slices SC

HORIZON 2020

H2020 - INFRAIA-2020-1

D2.4 Definition of common standards and practices for research reproducibility

Acronym	SLICES-SC
Project Title	Scientific Large-scale Infrastructure for Computing/Communication Experimental Studies – Starting Community
Grand Agreement	101008468
Project Duration	36 Months (01/03/2021 – 29/02/2024)
Due Date	31 August 2022 (M18)
Submission Date	14 September 2022 (M19)
Authors	Serge Fdida (SU), Hassane Rahich (SU), Sebastian Gallenmüller (TUM), Nikos Makris (UTH).
Reviewers	All partners

Scientific Large-scale Infrastructure for Computing Communication Experimental Studies **Starting Communities**

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101008468. The information, documentation and figures available in this deliverable, is written by the SLICES-SC project consortium and does not necessarily reflect the views of the European Commission. The European Commission is not responsible for any use that may be made of the information contained herein.

Executive summary

Repeated research is an important part of scientific methodology. Repeating or replicating research not only brings an independent perspective to the investigation, but also provides a basis for comparing different approaches and extending known results. The process of repeating others' research also allows us to learn, not only from their insights but also their experimental methodology. Replication is thus an important facet of scientific debate.

Up until recently repeating computer science research, while desirable in principle, was often difficult in practice. This is due to several well-identified factors such as the lack of incentives (e.g., in publication venues) that value such effort, lack of common research platforms and tools, and the difficulty of packaging the requisite digital artifacts in an easily shareable form and the mechanisms to combine and publish them. Change is needed.

SLICES, now on the ESFRI 2021 roadmap, ambitions to develop a framework to cope with the full research life-cycle. We are designing a solution for aligning with the best practices used in Open Science as well as an articulation with the EOSC initiative.

We present our preliminary thoughts and solutions to deals with open data and reproducible research, as well as relevant initiatives from other testbeds and relevant standardization efforts. A progressive approach will be taken with the goal to embed a native and robust solution that will align with the best practices adopted in other fields. In addition, international cooperation will be researched in order to support a common methodology across facilities.

Table of content

List of figures

1. Introduction

Advanced platforms are meant to support the discovery process in digital infrastructures. Indeed, it is often forgotten that a scientific instrument is not limited to a physical infrastructure but should also provide tools to support the full research life-cycle. In particular, it deals with open and FAIR data as well as reproducibility of experiments. The later has been a hotly debated topic but with little progress in our field. In addition, it is based on the very formal methodology, which is often associated to experimentally driven research.

Despite the challenge to support the full research life cycle, this ambition is part of SLICES RI. Although some preliminary tools developed in other projects can be considered for sharing data across testbeds, we target a more robust solution that will be fully integrated as a unified solution.

This is related to the ambition of the European EOSC (European Open Science Cloud) initiative. Researchers and research stakeholders nowadays require that research data is made available for other researchers to examine, experiment and develop further. Additionally, preserving the data in conjunction with how conclusions from the data were drawn, accelerates the discovery process, enable easier reproducibility of the results and thus supports evidence. It is then necessary to develop policies and procedures for regulating the management and publication of research data in order to make them interoperable and widely available.

In Europe, it is recommended to conform with the European Open Access policy, Open Research Data Pilot and FAIR principles in producing and managing research data.

We introduce and discuss some of the issues related to this important field of experimentally driven research, illustrated by example of projects that having addressed it. We highlight some recent initiatives at ITU in order to standardize some important APIs for the interoperability among testbeds, even if a more integrated solution is promoted in SLICES. We also propose a preliminary workflow that could be integrated in SLICES. This is developed with a full articulation with the work done in WP3.

2. The full research life-cycle

Experimentally-driven research should be grounded on a solid methodology that is understood and implemented by other disciplines and we will certainly benefit from the experience of those fields that have already developed some know-how and tools. This is somehow the ambition of the European EOSC (European Open Science Cloud) initiative. As a consequence, it shows that one should not only target the deployment of the instrument/facility but as importantly, address the full research life-cycle, including open data, data management and reproducibility.

Researchers and research stakeholders nowadays require that research data is made available for other researchers to examine, experiment and develop further. Additionally, preserving the data in conjunction with how conclusions from the data were drawn, accelerates the discovery process, enable easier reproducibility of the results and thus supports evidence. It is then necessary to develop policies and procedures for regulating the management and publication of research data in order to make them interoperable and widely available

In Europe, it is recommended to conform with the European Open Access policy, Open Research Data Pilot and FAIR principles in producing and managing research data. This requires defining appropriate metadata (including compatible experiment description) on the data produced by or integrated into the infrastructure with the objective to ensure eventually data accessibility, reusability and

interoperability with data produced by similar infrastructures/experiments for enabling complex experiments and multi-domain research.

Alignments with the relevant recommendations such as the ones published by EOSC FAIRs project, GO FAIR initiative and RDA for FAIR data management, and general European Open Access to research publications and Open Research Data Pilot policies, are of utmost importance.

The FAIR[1] (Findable, Accessible, Interoperable, and Reusable) Data Principles were developed to be used as guidelines for data producers and publishers, with regards to data management and stewardship. One important aspect that differentiates FAIR from any other related initiatives is that they move beyond the traditional data and they place specific emphasis on automatic computation, thus considering both human-driven and machine-driven data activities. Since their publication, FAIR became widely accepted and used.

To this end, advanced platforms should fully endorse and adopt the FAIR principles, in order to enable and foster the data-driven science and scientific data-sharing in this area.

For scientific research, replicability and reproducibility is crucial to validate and extends any experiment, this has traditionally been difficult with computer science research because of the availability and diversity of resources, both hardware, software and data.

Researchers need a solution to easily package their experiments in such a way that they can be repeated cost-effectively. In other word researchers are looking for a solution that makes it easy to:

- Run code in a stable, customizable and powerful environment;
- Provide documentation or explanation about the experiment;
- Share experimental code data and results.

We present the initial thoughts regarding this important topic and how it could be fully integrated in SLICES. Our scientific community recognizes the challenge and that very little progress has been achieved in the past. Therefore, it is highly important that a solid foundation is considered for SLICES. This is also tightly articulated with the work carried out in WP3.

3. FAIR and open data, Common standards

Understanding the data collected and processed within an advanced platform becomes essential to understand data usage from the target user groups. This should allow developing an appropriate information model that will represent the data collected from the platform, experimental equipment and applications. We consider that the datasets generated by a platform, involving hardware and software infrastructure, can be roughly organized into five main categories:

- **Observational Data:** will be collected using methods such as surveys (e.g., online questionnaires) or recording of measurements (e.g., through sensors). The data will include mostly data related to signal or performance measurements, and network or service log data that allow for experiment evaluation and reproducibility;
- **Experimental Data:** where researchers introduce an intervention and study the effects of certain variables, trying to determine their impact;

[1] Wilkinson, M., Dumontier, M., Aalbersberg, I. et al. The FAIR Guiding Principles for scientific data management and stewardship. Sci Data 3, 160018 (2016). https://doi.org/10.1038/sdata.2016.18 [Last accessed 25 August 2022]

- **Simulation Data:** is generated by using computer models that simulate the operation of a real-world process or system. These may use observational data;
- **Derived Data:** involves the analysis (e.g., cleaning, transformation, summarization, predictive modelling) of existing data, often coming from different datasets (e.g., the results of two experiments), to create a new dataset for a specific purpose;
- **Metadata:** concerns data that provides descriptors about all categories of data mentioned above. This information is essential in making the discovery of data easier and ensuring their interoperability.

In addition, an advanced open platform, will promote interoperability, thus non-proprietary, unencrypted, uncompressed, and commonly used by the research community formats should be adopted. In addition, the platform end users should have the ability to decide on a suitable license and attach it to their data.

It is important for the platform to derive a preliminary estimation of its dimensions (in number of single users as well as the data generated by the experiments in order to size the system needs in storage).

Data management raises several important concerns related to openness and privacy. It copes with the issues dealing with the governance of the data. Certainly, it will require to setup a data management framework to support the efficient and effective operation of the infrastructure and achieve the data dimension objectives. To accomplish this, the data management framework should clarify its own design goals, which are summarized below and presented in Figure 1:

- **Data Governance**: A systemic and effective Data Governance structure to support the data management operations through a hierarchical structure with appropriate roles (e.g., Data Manager, Data Protection Officer and Metadata administrator), implement all related policies and processes, and adopt standards and leading practices;
- **Data Architecture**: An agile Data Architecture that can perform efficiently to fulfill the infrastructure requirements, scales gracefully to accommodate for increased workloads, is flexible to integrate new processes and technologies, and is open to interact with other systems and infrastructures;
- **Data Quality**: Appropriate data transformation mechanisms to ensure Data Quality across multiple dimensions (e.g., accuracy, completeness, integrity), in order to improve data utility (e.g., further processing, analysis);
- **Metadata**: Appropriate metadata management mechanisms to facilitate collaboration between users by providing the means to share their data and also support FAIR data;
- **Interoperability**: Facilitate seamless interaction with other systems and infrastructures;
- **Analytics**: Deployment of statistical, machine learning and artificial intelligence techniques to draw valuable insights from data and appropriate visualisation techniques to interpret them;
- **Data Security:** Mechanisms to protect data from unauthorized access and protect its integrity;
- **Privacy**: Strict controls to manage the sharing of data, both internally and externally.

The SLICES-SC Data Management Plan, which corresponds to the deliverable D3.1, presents the metadata management. The main goal of the metadata management is to ensure the reusability, the interoperability and the data quality. Indeed, it is important that the published open data are described with metadata allowing the compliance with the FAIR principles.

Several metadata format standards were already studied: AGLS, AGRkMS, EAD, OAIS, PREMIS, OpenDOAR, Dublin Core, DataCite, MINSEQE, DDI, EML, ISO 19115, FGDC-CSDGM, FITS and MIBBI. Finally, the results conducted by the SLICES consortium are demonstrated that the Dublin Core standard is the most appropriate to describe the metadata generated by the SLICES Research Infrastructure. Indeed, the Dublin Core standard is already widely used by the research community and its level of maturity is high.

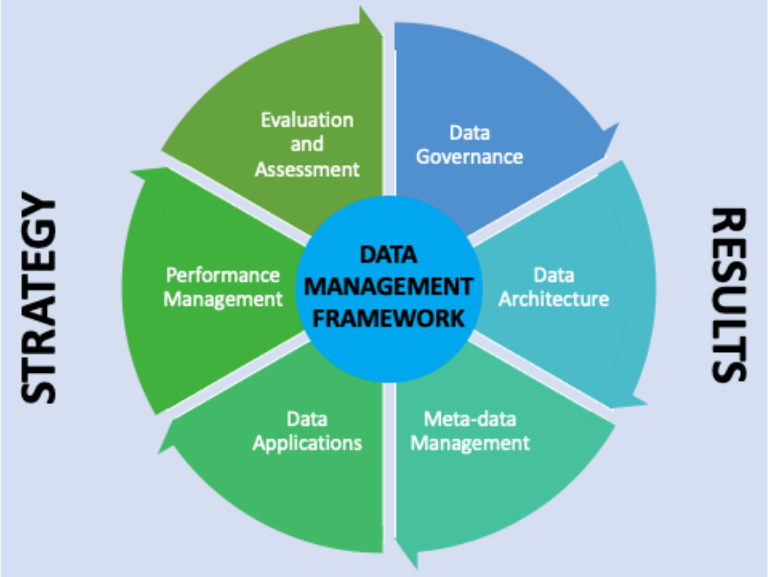


Figure 1: Data Management Framework

4. Interoperability with EOSC

EOSC has established itself as an important pillar in Europe for the implementation of Open science concepts by accelerating the adoption of the FAIR data practices among researchers in the European Union. Integration of the platform into European Research Infrastructure via EOSC will facilitate data sharing and reuse among the platform partners and the larger European researchers' community. This is particularly true for the European platforms but it could also provide a guideline for the cooperation on this topic with the US, as shown in recent talks from Kate Keahey[2].

Since aligning EU and US platforms will mobilize experimental research platform by jointly utilizing the geographically dispersed computing, storage and networking infrastructures, it is highly important that the different facilities interacting in the experimental workflow are interoperable with each other. Similarly, existing research needs to be accessible and directly pluggable to the platform's services and sites. It is thus necessary that resource description, availability, execution and data exchanges are effective. This can only be assured if a common interoperability framework is adopted across the platforms ecosystem so that different subsystems have a common understanding of resources, data/metadata and are on the same page with respect to the licensing, copyright and privacy requirements. The platforms infrastructure should be designed to ensure compatibility and integration with EOSC, and be ready to offer advanced ICT infrastructure services to other RIs and projects, with the special focus on the FAIR data management and exchange.

[2] Chameleon Reproducibility Project, presentation of Isabel Brunkan, https://www.youtube.com/watch?v=zOLpNBurQD4, [Last accessed 25 August 2022]

Therefore, it is of utmost importance to design the integration framework of the research platforms with EOSC in such a way that the data exchange between the platforms and EOSC is interoperable for scientific workflow management for data storage, processing and reuse.

Interoperability is an essential feature of EOSC ecosystem as a federation of services and data exchange is unthinkable without interoperability among different EOSC constituents. The meaningful exchange and consumption of digital objects is necessary to generate value from EOSC which can only be realized if different components of the EOSC ecosystem (software/machines and humans) have a common understanding of how to interpret and exchange them, what are the legal restrictions, and what processes are involved in distribution, consumption and production of them. To facilitate this, an EOSC interoperability framework (EOSC-IF) should be defined as a generic framework for all the entities involved in the development and deployment of EOSC. The EOSC interoperability framework is a set of *not-so-specific* guidelines to ensure smooth integration of infrastructure services and seamless exchange of research data across the EOSC ecosystem. EOSC-IF is derived from the European Interoperability framework that defines interoperability of an information technology system by four key elements, i.e.., technical, semantic, organization and legal interoperability.

The FAIR principles, federated resource and user management and legal compliances pertaining to privacy, licensing and governance by European Commission are at the core of the EOSC-IF framework.

The interface should be built upon the foundations led by the European Interoperability Reference Architecture (EIRA), where interoperability is classified at four layers, namely: (i) technical, (ii) semantic, (iii) organizational; and (iv) legal.

FAIR Digital Object (FDO) is the core building block of EOSC-IF. Here, Digital Object refers to the kind of objects that allow binding all critical information about any entity. In EOSC, a digital object can be research data, software, scientific workflows, hardware designs, protocols, provenance logs, publications, presentations, etc., as well as all their metadata (for the complete object and for its constituents). *Figure **2*** shows a schematic of FDO. An FDO should conform to all the four layers of interoperability introduced earlier in this document by following the FAIR guidelines.

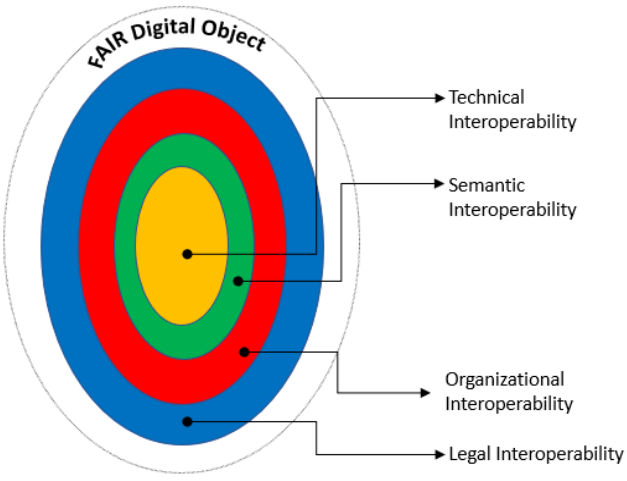


Figure 2: EOSC FAIR Digital Object illustration

FDO (FAIR Digital Object) and PID (Persistent Identifier) are key components of the EOSC architecture and supporting federated data infrastructure that ensures consistent implementation of the FAIR data principles. FAIR principles are realized through FAIR Digital object, and PID services and infrastructure

provide facilities to access and manipulate FDO. The FDO Forum provides the following definition of the FDO[3]: "A FAIR digital object is a unit composed of data and/or metadata regulated by structures or schemes, and with an assigned globally unique and persistent identifier (PID), which is findable, accessible, interoperable and reusable both by humans and computers for the reliable interpretation and processing of the data represented by the object."

According to FDO Forum, the FAIR Digital Object concept brings together FAIR guiding principles and Digital Object that supports interoperability across existing and evolving data regimes/frameworks using mechanisms of the structured machine-readable identifiers and principles of persistent binding for digital object of different types. It is important to mention that activities currently coordinated by FDO Forum are including essential results and current activities at the RDA (Research Data Alliance) on Data Types and Data Types Registries, Data Factories, others, and are aligned with EOSC technical development what is reflected in the EOSC Architecture and EOSC Interoperability Framework.

The following main requirements to FDO services and infrastructure defined in the FDO Framework:

- General requirements include machine actionability, technology independence, persistent binding, abstraction and structured hierarchical encapsulation, compliance with standards and community policies;
- FDO is identified by PID; there are possible multiple PID frameworks defined by PDI scheme, namespaces, ontologies or controlled vocabularies;
- A PID resolves to a structured record (PID record) with attributes that are semantically defined within a (data) type ontology (which may be defined for different application or science domains);
- The structured PID record includes at least a reference to the location(s) where the FDO content and the type can be accessed, and also metadata describing FDO can be retrieved;
- Each FDO is accessed via API by specifying PID and additionally point. API must support basic operation with FDO: Create, Read, Update, Delete (referred to as CRUD), however subject to access control and policy;
- Metadata used to describe FDO properties should use standard semantics and registered schemes to allow machine readability and actionability.

5. Illustration of a full-research life cycle

In order to realize the vision of FAIR research, supporting the full research lifecycle, we consider two consider two simples, although incomplete, example borrowed from the Chameleon test facility (https://www.chameleoncloud.org/) and the Fed4Fire+ federation of testbeds (https://www.fed4fire.eu/).

The Chameleon solution is of utmost interest as, as far as we know, it is the only solution developed lately and proposing reproducibility methodology for the cloud and networking testbeds founded in the US. We are aware of solutions for supporting the full research life cycle but in other disciplines of Science. Although the approach proposed in Chameleon is at an early stage, it is considered by other platforms and a dialogue started with SLICES as interoperability for platforms on both sides of the Atlantic will be a must.

[3] FAIR Digital Object Framework, https://www.go-fair.org/today/fair-digital-framework/, [Last accessed 25 August 2022]

Chameleon, as a shareable, cloud testbed, is designed for experimental collaboration. The Jupyter Notebook integration allows to mix explanatory text with actionable code, replay experiments, or break them down cell by cell. Whether to avoid future troubleshooting or provide educational value, the Chameleon-Jupyter Notebook combination unifies cloud capabilities with code and text.

With the integration with Zenodo, a digital publishing platform, and the introduction of a feature called **Trovi** that can be considered as "google drive" for chameleon experiments, Chameleon allows publishing a digital representation of an experiment and provides a digital object identifier (DOI) that gives the possibility to reference the experiment from a paper.

Packaging experiments for reproducibility in Chameleon can be easily achieved by following the next four steps:

Step 1: Creating the experimental environment (Container)

To begin any experiment on Chameleon, the first step is creating an environment. This can be done using any orchestration method, but it is particularly convenient to do from a Jupyter Notebook.

Step 2: Running the Experiment

Once the experimental container is ready, it's time to start coding up what it takes to run the experiment. Depending on the complexity of the experiment, the code can be put in a separate notebook (and then reuse the experimental container for multiple different experiments). A big advantage of using Jupyter is that the integration of code with text gives the possibility to explain the experimental steps, justify choices, and discuss alternative approaches.

Step 3: Data analysis

The reason why Jupyter is such a handy tool is that it is also a conducive environment for data analysis — after the experiment is finished, the results can be downloaded locally to analyze and visualize them within Jupyter. Jupyter offers a graphical and image visualization that can also be included in the notebook, so the container, experiment, and results are all in the same place.

Step 4: Sharing the experiment

Ultimately, implementing experiments within the Chameleon-Jupyter Notebook environment allows replicating the environment and experiment exactly — saving the container-building commands and experiment code in one place. In doing this, others — whether collaborators, reviewers, or other researchers interested in this area — can easily repeat the experiment by launching the notebook.

Below are some examples of experiments that followed the Chameleon approach for reproducibility:

1. **Application-based QoS Support with P4 and OpenFlow:**

   A networking experiment showing an early use of Jupyter notebook for packaging -- while no published Juypyter notebook for this one exists.

2. **Image Classification with AlexNet on the Stanford Dogs Dataset:**

   A machine learning experiment packaged in Jupyter Notebook, designed to be run with tools available within Chameleon and OpenStack. The packaged notebook is available on Zenodo, making it easy to reproduce in ~1 hour and perfect to use to teach machine learning or how to use the Chameleon testbed.

3. **Tiny-Tail Flash: Near Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs Reproduction:**

For this experiment, a Jupyter notebook is packaged and available from Zenodo, reproducing the Dev Tools Release experiment from this paper.

The second example is illustrated by the solution proposed in the context of the EU Fed4Fire federation of testbeds. In the Fed4FIRE+ project (2017-2022), a specific task was dedicated to the topic of Experiment-as-a-Service, data retention and reproducibility of experiments. The following tools and frameworks were discussed:

- ESpec (Experiment Specification): a framework to provision resources and (complex) software/start up scripts. As flexible to e.g., install OpenStack on a bunch of bare metal nodes
- Expo: a lightweight experiment orchestration tool to be used during the experiment phase to coordinate an experiment on multiple nodes
- Jupyter notebooks to help in reproducing experiments
- A metadata bundler to make it easy to collect a lot of information on the used resources
- Distrinet: a tool to scale up network experiments while making them reproducible as well

The Experiment Specification format is not a replacement for the RSpec format (defined by the GENI/Fed4FIRE APIs). An ESpec contains an RSpec and combines it with other files.

- The purpose of an RSpec is to define which resources are needed.
- The purpose of an ESpec is to additionally define which files should be placed where, and which scripts should be started.

The current ESpec specification already allows some complex ESpecs. However, the base idea when using an ESpec should be to keep it simple, and to put as little as possible in the ESpec. It is meant for easily bootstrapping your experiment. It is not meant for running experiment logic.

It is preferable to keep ESpecs so simple that a user not familiar with the format, will be easily able to manually execute your experiment using tools that do not support ESpec.

The ESpecs can be used with the jFed client tool (https://jfed.ilabt.imec.be), both in the GUI version and the command line version.

We defined e.g., an ESpec that deploys OpenStack via the EnOS framework.

For a full description, see https://jfed.ilabt.imec.be/espec, see some complex examples below:

```
version: 1.0-basic
rspec:
   - bundled: 3-nodes.rspec
upload:
   - exp-files-set1.tar.gz
   - bundled: exp-files-set2.tar.gz
     path: /tmp
     nodes: [central, exp1]
   - download: http://example.com/exp-files-set3.tar.gz
   - direct: |
        You can also directly specify the content of a file. This text will thus be stored on all nodes in /tmp/
        Check the yaml syntax of "literal-blocks" for details about syntax and removing indentation
     path: /tmp/demo.txt
execute:
   - bundled: setup-central-node.sh
     nodes: central
   - bundled: setup-exp-node.sh
     nodes: [exp1, exp2]
   - local: /work/repo/start-exp.sh
     nodes: [exp1, exp2]
```

Figure 3: ESpec example with new functionality of 'direct' for literal blocks of data

```
version: 1.0-basic
rspec: my-experiment.rspec
dir:
   - path: /work/ansible/
     content: ansible
ansible:
    host:
       type: EXISTING
       name: control
       galaxy-command: /usr/local/bin/ansible-galaxy
       playbook-command: /usr/local/bin/ansible-playbook
       execute:
          - my-custom-ansible-install.sh
    galaxy:
       - download: http://example.com/ansible-requirements.yml
       - my-ansible-requirements.yml
    playbook:
       - bundled: setup-software.yml
         debug: 2
       - run-1st-experiment.yml
       - run-2nd-experiment.yml
    group:
       servers:
         - server1
         - server2
       clients:
         - client1
         - client2
```

Figure 4: ESpec example with ansible scripting, supporting the new item 'galaxy' for ansible galaxy (pre-packaged units of work)

ExpO is short for "Experiment Orchestrator". It allows you to run time-sensitive experiments over multiple machines in a very light way and can be used after the provisioning phase (manual provisioning through a tool/script or automated e.g., with ESpec).

ExpO consists out of two pieces of lightweight software:

- the **ExpO slave** which is present on all machines participating in the experiment, waiting for instructions on when to execute commands
- the **ExpO director** which executes experiments defined in an *Experiment Orchestration definition*

See for the full details at: https://gitlab.ilabt.imec.be/ilabt/expo

An example of an orchestration definition:

```
version: 1.0
nodes:
  node1: [groupA]
  node2: groupB
  node3:
    - groupA
    - groupB
  node4
commands:
  - command: "iperf -s"
    groups: [groupA]

  - after: 10
    id: iperf_client
    command: "iperf -c server"
    groups: [groupB]

  - command: echo "Hello $EXAMPLE_NAME"
    args:
      chdir: /tmp
      environment:
        EXAMPLE_NAME: 'World'
    nodes: node4
```

Figure 5: Example of an ExpO orchestration definition (including the new 'environment')

And all functionality is described here:

The file format

`version` Currently always 1.0

`nodes` List of nodes expected in the experiment. Optionally you can specify to which groups the node belongs in this experiment

`commands` : List of commands to be executed

`Command` format

`command` : the command to execute

`after` : number of seconds after the previous command that this command must be executed (optional, if omitted, it will be executed immediately after the previous command)

`daemon` : whether to keep this command running in the background. Used by the CLI to know if it should wait for the command to finish or not.

`groups` : the groups which must execute the command (optional if `nodes` has been specified)

`nodes` : the nodes which must execute the command (optional if `groups` has been specified)

`id` : a custom ID to identify MQTT-messages which relate to this command, such as the result, stdout, stderr, stdin (optional, if omitted, the index of this command in the `commands` -list is used as an id)

`stderr` : whether to stream the stderr as MQTT-messages with topic `expo/<experiment_id>/node/<node_id>/cmd/<command_id>/stderr` (defaults to True)

`stdout` : whether to stream the stdout as MQTT-messages `expo/<experiment_id>/node/<node_id>/cmd/<command_id>/stdout` (defaults to False)

`stdin` : whether to stream data sent to MQTT topic `expo/<experiment_id>/node/<node_id>/cmd/<command_id>/stdin` or `expo/<experiment_id>/group/<group_id>/cmd/<command_id>/stdin` to the stdin of the process (defaults to False)

Warning: the order in which data is streamed to the stdin of a process is indeterminate when mixing both topics

Figure 6: Documentation of the ExpO orchestration definition file

Some of the testbeds, in particular imec's GPULab and Inria's GRID'5000 offer jupyter notebooks to the experiments which help in reproducing and sharing experiments.

Jupyter notebooks at imec's GPULab testbed

See https://doc.ilabt.imec.be/ilabt/jupyter/index.html, you can use GPUs or only CPUs to use the notebooks. This is typically used for first steps in machine learning, classes and quick experimentation (GPULab with a job-based workflow is then used for longer running more advanced jobs).

Below you can see a screenshot of the environment, with an example class on machine learning that has been used.

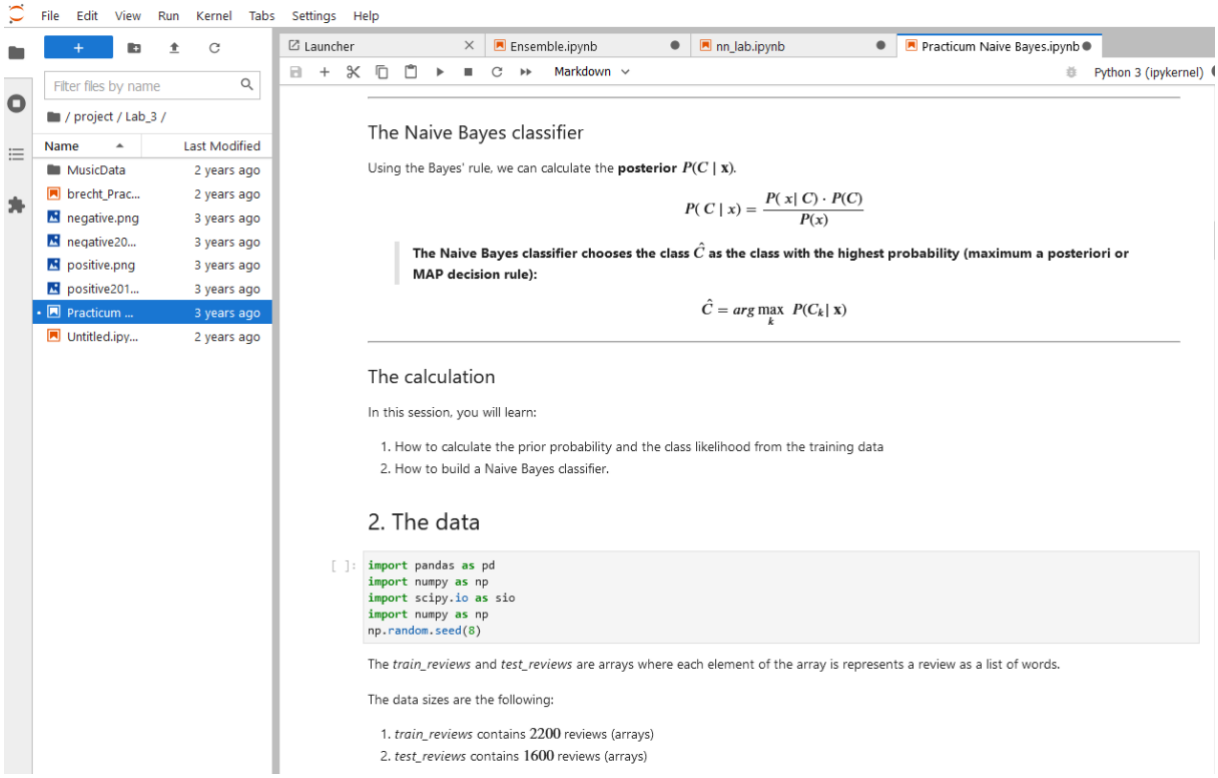


Figure 7: Jupyter notebook at imec's GPULab testbed

Jupyter notebooks on Inria's GRID'5000[4]

Computer science testbeds often require extensive experiment automation to be used efficiently. Jupyter notebooks can contain the scientific reasoning, experiment orchestration, and experiment results in an easy to read, easy to execute format. However, some level of support is required from the testbeds to facilitate notebook use on their platforms. Additionally, this format can be used for more than driving experiments, and different uses have different requirements in terms of support.

During research a lot of code is often generated to collect and process data. Often this code was unpublished, researchers simply described the process in the corresponding article, and even when it was published the code was often relatively undocumented and hard to follow. In proposing their notebook format integrating code and prose, the Jupyter team hoped to foster readable code for reproducibility.

Jupyter notebooks combine ideas of literate programming, interactive programming, and laboratory notebooks. Notebook files contain code cells interspersed with text. Additionally, outputs generated by the executed code cells are also saved in the notebook. The jupyter-notebook application is designed to view, edit, and run notebooks and the execution of code cells is handled by Jupyter kernels.

We decided to deploy a single JupyterHub instance for the whole of g5k. This instance is placed on a service machine in the internal g5k network, with a reconfigurable proxy executed on the same machine as the hub. To allow users access to the reconfigurable proxy, and thus the hub and labs, a

[4] An extensive description can be found at https://hal.archives-ouvertes.fr/hal-03233095 as well, [Last accessed 25 August 2022]

route is added to g5k's pre-existing Apache proxies. This frontline proxy is already used for other g5k services, can handle user authentication, and will mitigate any weakness the reconfigurable proxy might have.

The authentication module used in our hub is `jhub_remote_user_authenticator`[5], which removes the authentication page and instead relies on incoming HTTP headers for authentication. This allows our frontline proxy to perform authentication and pass on authentication information to the hub through http headers. This is advantageous as it uses g5k already established authentication infrastructure without involving a new service.

The spawner module is a custom module implemented specifically to match g5k's needs, called `G5kSpawner`. From the user point of view the spawner requires the user select a site and whether to start the lab on a front-end or a node. If a node is selected, the users can specify the OAR resource tree to request, the wall time of the OAR job and other information required by OAR to service the request.

For the execution of lab instances on compute nodes the hub delegates the execution of the `jupyter-labhub` program to OAR, that it controls through g5k's REST API.

The REST API allows the hub to interact with OAR instances of every site and as a service operated by g5k, the hub is provided an SSL client certificate allowing it to make calls to the API in the name of the user requesting the lab instance. When requesting resources from OAR the hub provides the lab command to execute. OAR will execute the command automatically once the requested resources become available on the main node of the request. If the jupyter-labhub command ends before the end of the OAR job, the job will be ended immediately by OAR. As such the lifecycle of the OAR job and the lifecycle of the lab instance are closely interlinked and the hub treats them as one and the same. Under this mode of operation, the three main functions operate as follows:

The close interlink between the lab and the corresponding OAR job, the REST API, and the existence for the python-grid5000 library[6] used to interact with the API greatly simplifies the code used when instantiating labs on nodes.

When running experiments on Grid'5000, users generate metadata across multiple services. This metadata is useful for reproducibility purposes or scientific dissemination. The `g5k-metadata-bundler` is a tool designed to retrieve metadata across all the different services and bundle them in a single archive. The bundle only retrieves metadata generated by Grid'5000 services, the collection of data generated by the users' experiment is beyond the scope of this application.

This can be found at https://www.grid5000.fr/w/Grid5000_Metadata_Bundler

[5] C. Waldbieser et al. (2019). Jupyterhub remote user authenticator, https://github.com/cwaldbieser/jhub remote user authenticator, [Last accessed 25 August 2022]

[6] M. Simonin. (2019). Python-grid5000, https://gitlab.inria.fr/msimonin/python-grid5000, [Last accessed 25 August 2022]

Usage

G5k-metadata-bundler is installed on every site frontend in Grid'5000. It can only be executed from the site frontends.

```
g5k-metadata-bundler -s SITE -j JOBID [-o NAME]
  -v, --version                    Print g5k-metadata-bundler version
  -s, --job-site SITE              [MANDATORY] Grid'5000 site from which to extract
  -j, --job-id JID                 [MANDATORY] Job id of the OAR jod to extract
  -o, --output NAME                Bundle name to use for the directory/archive
```

Users do **not** need to operate the bundler on the same frontend as the site the jobs was executed on. The bundler download all data pertaining to the queried job and bundle in a archive named `g5k-bundle-` `SITE` `-` `JID` `.tar.gz` or if an output name has been provided `NAME` `.tar.gz`. The bundle is provided in as a tar.gz archive which can be manipulated by using the following commands:

- **Listing**

  `tar -tzf` `NAME` `.tar.gz` lists all files contained within the bundle

  Extraction

  `tar -xzf` `NAME` `.tar.gz` extracts all files to a directory with the same name as the bundle

Users operating on older versions of Windows might require thrid party software to unpack the bundle. (often 7-zip)

Example usage

```
user@fsophia:~$ g5k-metadata-bundler -s nancy -j 3003030
 Running g5k-metadata-bundler for job 3003030 at nancy
 Downloading https://api.grid5000.fr/stable/sites/nancy/jobs/3003030
 Downloading https://api.grid5000.fr/stable/sites/nancy/clusters/graoully/nodes/graoully-
1?version=7f6b81c2621c6ed3a4fac632f213436813495755
 Downloading https://api.grid5000.fr/stable/?version=7f6b81c2621c6ed3a4fac632f213436813495755&deep=true
 Downloading https://api.grid5000.fr/stable/sites/nancy/metrics?job_id=3003030&nodes=graoully-1
 Generating README
 Compressing bundle
 Bundle created at g5k-bundle-nancy-3003030.tar.gz
user@fsophia:~$ ls -lh g5k-bundle-nancy-3003030.tar.gz
 -rw-r--r-- 1 user g5k-users 456K Jul 19 09:50 g5k-bundle-nancy-3003030.tar.gz
user@fsophia:~$ tar -tzf g5k-bundle-nancy-3003030.tar.gz
 g5k-bundle-nancy-3003030/
 g5k-bundle-nancy-3003030/g5k-oarjob-nancy-3003030.json
 g5k-bundle-nancy-3003030/README
 g5k-bundle-nancy-3003030/g5k-resource-nancy-graoully-1-7f6b81c2621c6ed3a4fac632f213436813495755.json
 g5k-bundle-nancy-3003030/g5k-monitoring-nancy-graoully-1-3003030.json
 g5k-bundle-nancy-3003030/g5k-refapi-7f6b81c2621c6ed3a4fac632f213436813495755.json
```

Figure 8: Grid'5000 metadata bundler usage and example

The bundle contains these different files types:

- **g5k-oarjob-SITE-JID.json**: Job files

  Contains the information for a given OAR job JID at Grid'5000 site SITE such as:

  - submission, start, and end dates;
  - user and group (group granting access) of the job;
  - job types and properties;
  - command executed by the job;
  - list of resources attributed to the job;
  - OAR events for the job.

  This information is extracted from the jobs API

- **g5k-resource-SITE-NODE-VERSION.json**: Resource files

Contains information about a single NODE, such as:

- Node architecture, bios, ram, and cpu information;
- network, storage, and monitoring devices;
- base configuration information.

The bundle will contain one such file for each of the nodes involved in a job.

This information is extracted from the reference API

The VERSION of the information contained in this file will match what it was on the day the job was executed.

- **g5k-refapi-VERSION.json**: Reference API files

Contains a full copy of the reference API at VERSION

This can be used to look up information about nodes not directly used by the bundled oar jobs.

The resources files are a subset of this file.

- **g5k-monitoring-SITE-NODE-JID.json**: Monitoring files

Contains all the monitoring measurements made by Kwollect for a NODE during the oar job JID.

The contents will vary depending on how much monitoring was enabled for a given job. See default metrics in Monitoring_Using_Kwollect.

Often the heaviest files in the bundle

This information is extracted from Kwollect

- **README**: The readme file

Contains information pertaining to the execution of the bundler such as:

- Bundler version
- Execution date
- List of warnings and errors that happened during bundling
- List of files included in the bundle with a short description

User will also find at the end of every file a small bundler info segment. This segment contains the date at which the file was generated, warnings raised by the file generation and a list of references indicating how this file relates to other files in the bundle.

As the bundler is still in alpha version, we welcome comments and feature requests.

The following is a list of features we are already looking at:

- Concerning bundle contents:
  - Bundling multiple jobs in a single archive;
  - Bundling based on job names;
  - Better management of monitoring information when it is too big for download;
  - Adding information concerning the standard environment to the bundle;
  - Adding image deployments to the bundle:
    - Adding information concerning the deployed images.

- Concerning bundler operation;
  - No-compress mode where the bundle is left as a directory containing all files;
  - Appending new files to an existing bundle;
  - Reduce memory footprint;
  - Assess viability of parallel downloads.

On the Fed4FIRE testbeds it is possible to run large scale networking experiments, but they are restricted to the physical limits (e.g., only up till 11 network cards on nodes), so it's not infinite scale. Mininet on the other hand is a tool for network simulation but is limited to a single machine. If we use the power of the testbeds with a lot of nodes and deploy a distributed version of mininet on this, that would scale to really huge networks.

The objective of Distrinet was to enhance network experiment tools in order to address the orchestration of large-scale experiments on grid and cloud environments and make it easier to automatically reproduce experiments. This section introduces Distrinet-HiFi: a Distrinet plug-in to monitor fidelity of emulated experiments based on measurement of packet delays.

This info can be found at https://github.com/distrinet-hifi

The scripts given here use apssh and asynciojobs to remotely run parallel commands on a number of nodes. First make sure you have a recent version of Python (>= 3.6), then install those on your computer:

```
pip3 install apssh asynciojobs
```

You also need to have a slice in R2Lab and your computer must be able to log onto the gatewy node. If this is not the case already, you can ask to register for an account.

If you would rather use your own cluster of computers to deploy Distriniet-HiFi and/or run the proposed experiments manually, you can ignore this step.

To set up the experiment, the R2Lab nodes must be correctly configured and running the latest stable version of Distrinet. You can use the already available images to set up your testbed, with one master node and one or more worker nodes:

```
rhubarbe load -i u18.04-distrinet_hifi_leader $LEADER_NODE
rhubarbe load -i u18.04-distrinet_hifi_worker $WORKER_NODE_1 $WORKER_NODE_2
...
```

You can also manually install the testbed if you do not wish to use R2Lab. Make sure to have a recent Linux Kernel (the tool has been tested on v4.15.0) on all your nodes then install bcc and download and install Distrinet. Then copy hifi.py to the mininet code directory in your master node (~/Distrinet/mininet/mininet/), and the rest of the files to a ~/experiment/ directory you would have created in all the nodes of your testbed.

First import the Distrinet-HiFi library in your Distrinet script:

```
from mininet.hifi import Monitor
```

and wrap your experiment in the monitoring process:

```
monitor = Monitor(net)
monitor.start()
monitor.wait()
# run your experiment...
```

```
monitor.stop()
monitor.receiveData()
monitor.analyse()
```

Before running your experiment, initialise the monitoring agents on each node of your cluster:

```
python3 agent.py --ip=NODE_IP --bastion=LEADER_IP
```

6. ITU-T Focus Group on Testbeds Federations for IMT-2020 and beyond (FG-TBFxG)

In December 2021, the ITU-T Study Group 11 (SG11) "Signalling requirements, protocols, test specifications and combating counterfeit products" has established the ITU-T Focus Group on Testbeds Federations for IMT-2020 and beyond (FG-TBFxG)[7]. The creation of this ITU-T Focus Group was supported by the Fed4FIRE+ project which included several partners of the SLICES Research Infrastructure. Indeed, ITU-T SG11, ETSI TC INT and members of the Fed4FIRE+ project have worked together to write the Recommendation ITU-T Q.4068 "Open APIs for interoperable testbed federations". In parallel, a joint SDOs brainstorming workshop on testbeds federations for 5G and beyond was organised in March 2021 by ITU-T SG11, ETSI TC INT and IEEE. The topics of the workshop were the interoperability, the standardisation, the reference model and APIs in the context of testbeds federations. One of the main takeaways of the successful submission of the Recommendation ITU-T Q.4068 and the joint workshop was to build a dedicated and open ITU-T Focus Group to work deeper on the different aspects of the testbed federations.

This ITU-T Focus Group is constructed as an exchange platform open to each individual from a country which is a current member of ITU-T. So, the methodology to contribute is very simple in fact: during one year, several meetings of the ITU-T Focus Group on Testbeds Federations for IMT-2020 and beyond are organised and each time, a deadline is given to receive any contributions following the template provided by ITU-T. Every contribution is examined in the plenary meeting to check its pertinence and is incorporated to the related deliverables of the Focus Group if there is no objection or remark. This methodology permits to integrate different proposed solutions in the Focus Group deliverables. The targeted stakeholders for the ITU-T Focus Group FG-TBFxG are the users of the testbeds, the testbed providers, the research community, the SDOs, the fora, the communication service providers (CSPs), the network operators, the infrastructure providers and the open-source projects.

The objectives of the ITU-T Focus Group FG-TBFxG is to harmonise specifications concerning the testbeds across the main SDOs and fora working in the testbed federations. This work includes the development of APIs which are aligned with the testbeds federations reference model described in the Recommendation ITU-T Q.4068. New use cases and services, notably linked to the concept of Testbed as a Service (TaaS), can be proposed by the contributors; the use cases will be afterwards reported in the related Focus Group deliverables.

The ITU-T Focus Group on Testbeds Federations for IMT-2020 and beyond (FG-TBFxG) is structured into three Working Groups (WGs):

- **WG1 Use Cases, Applications and Industry Demand, Business Models.** This Working Group defines the scenarios for testbeds federations with a strong focus on models, ecosystems and value-chains. Furthermore, it studies the market and the industry demand concerning the

[7] ITU-T Focus Group on Testbeds Federations for IMT-2020 and beyond (FG-TBFxG) website: https://www.itu.int/en/ITU-T/focusgroups/tbfxg/Pages/default.aspx, [Last accessed 25 August 2022]

testbeds. The establishment of use cases, within the business models and value-chains models, is realised in the Working Group 1.

- **WG2 Testbeds as a Service.** The Working Group is elaborating the requirements and the reference model dedicated to the concept of Testbeds as a Service. Different aspects such as the user interfaces, services and remote access, are addressed by the WG2. The interoperability, the integration and the extensibility of the Testbeds as a Service are also examined in this Working Group.
- **WG3 APIs, Reference Model Instantiations.** This Working Group is defining the APIs complementing the Recommendation ITU-T Q.4068. It provides the guidance to instantiate the testbeds federations reference model. Furthermore, a framework is scheduled to be provided to ensure that existing IMT-2020/5G testbeds are fully aligned and compatible with the APIs specified in the Focus Group.

Each Working Group has a set of deliverables to be published. They will contain all the contributions received during the FG-TBFxG meetings. The list of the Focus Group FG-TBFxG is shown below:

Deliverables	Title
D0.1	Technical Specification: High-Level Taxonomy
D1.1	Technical Specification: Taxonomy of Use Cases for Federated Testbeds covering Verticals, Technologies, and Business Scenarios; Focus on Synergies and Commonalities
D1.2	Technical Specification: Consolidated Technical Requirement Set for Federated Testbeds Use Cases; Focus on Stakeholders Models (SMEs, ISVs, CSPs, …)
D1.3	Technical Specification: Use Case Description on Use of Federated Testbeds in Testing Federated Autonomic Management and Control (AMC) Operations by ETSI GANA Components within and Across Multiple 5G Network Operators
D2.1	Technical Specification: User requirements and reference model for Testbed as a Service
D2.2	Technical Specification: Testbed as a Service API and interoperability specifications
D3.1	Technical Specification: APIs Definitions on Testbeds Federations, and APIs Invocations Framework
D3.2	Technical Report: Instantiations of the Testbeds Federations Reference Model, Transformation of existing IMT-2020/5G related Testbeds APIs
D3.3	Technical Report: Guide on Development and Maintenance of ONPs (Open Networking Platforms) and Federations for IMT-2020/5G & Beyond
D3.4	Technical Specification: Definition of KPIs specific to Testbed Federations
D3.5	Technical Report: Use of Open-Source & Open Hardware Projects/Products in Testbed Federations for IMT-2020/5G and Beyond

The contributions made by the different partners of SLICES-SC to the ITU-T Focus Group on Testbeds Federations for IMT-2020 and beyond (FG-TBFxG) are reported in the deliverable D6.2 “Mid-term Report on Dissemination, outreach, community building and standardisation”.

Overall, this is an important and valuable effort to push these needs into standardization organizations. Lessons will be learned from this initiative and outcomes will be taken onboard SLICES with the required adaptations necessary to support the centralized and integrated model suited to ESFRI.

7. Reproducibility of experiments leveraging virtualization of resources

Recent advances in networking including Network Functions Virtualization can assist towards strengthening experiment reproducibility and repeatability. NFV can significantly empower network flexibility and scalability, by wrapping the experiment services/applications in appropriate format (e.g., Virtual Machines, or containers such as docker), regardless of the underlying hardware. By adopting such approaches, experiment evaluation can be reproducible across different test platforms, with either similar or completely different hardware. By adopting the same workflow defined within the experiment, the results can be repeated from different groups.

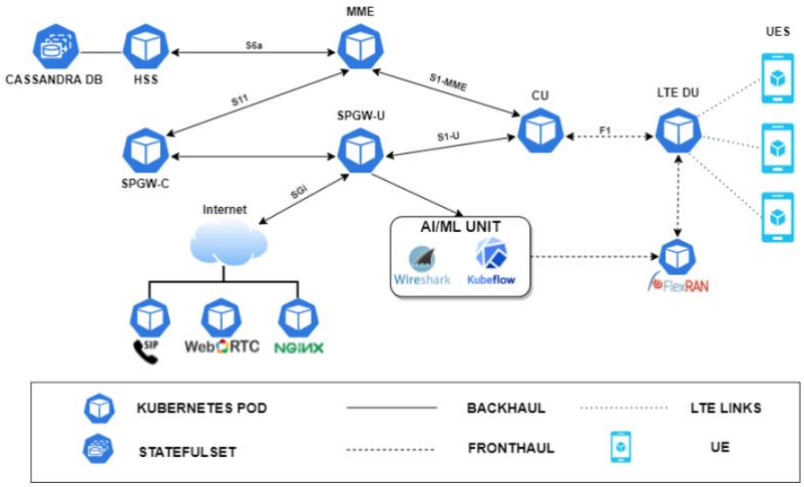


Figure 9: Example experiment achieving reproducibility and repeatability through the adoption of virtualization

As example we illustrate the technologies needed for executing the experiment defined in [8]. The core experiment is using Machine Learning driven optimizations to a Telecom network. The experiment requires the extended use of resources for online training and validation of the prediction models, which appropriately feed into a near-real time controller that manages the Radio Access Network. Towards ensuring the reproducibility and repeatability of the experiment, the different experimental resources needed have been wrapped appropriately in docker containers. Similarly, for the Machine Learning optimizations, the Kubeflow framework is used, which appropriately allocates the training workloads to multiple containers, dispersed over the available computational resources. For the overall orchestration of the experiment (and networking setup across different components – e.g. core network to RAN, ML to radio controllers) the Kubernetes framework was used. The experiment

[8] T. Tsourdinis, I. Chatzistefanidis, N. Makris and T. Korakis, "AI-driven Service-aware Real-time Slicing for beyond 5G Networks," IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2022, pp. 1-6, doi: 10.1109/INFOCOMWKSHPS54753.2022.9798391.

was able to be reproduced from an entirely different group, while the same experimental results were obtained, using the available resources provided from the experiment authors [9]. As a result, the experiment was awarded a "reproducibility award" in the CNERT 2022 conference.

8. Reproducibility using the pos experiment workflow

In SLICES, we propose to explore a novel approach toward reproducible research—a structured experimental workflow that allows the creation of reproducible experiments without requiring additional efforts of the researcher. This concept is realized in the **plain orchestrating service**[10] (pos), which enables the creation of such experimental workflows. The experiment is documented via a fully scripted experiment structure. Further, pos considers the entire experimental workflow from experiment orchestration, to data measurement, to result evaluation. In addition, pos provides scripts that enable the automation of the bundling and release of all the created experimental artifacts. An example for such a workflow is publicly available[11].

The workflow of an experiment according to the pos workflow is depicted in Figure 10. There, the experiment is split in three phases, described in the following.

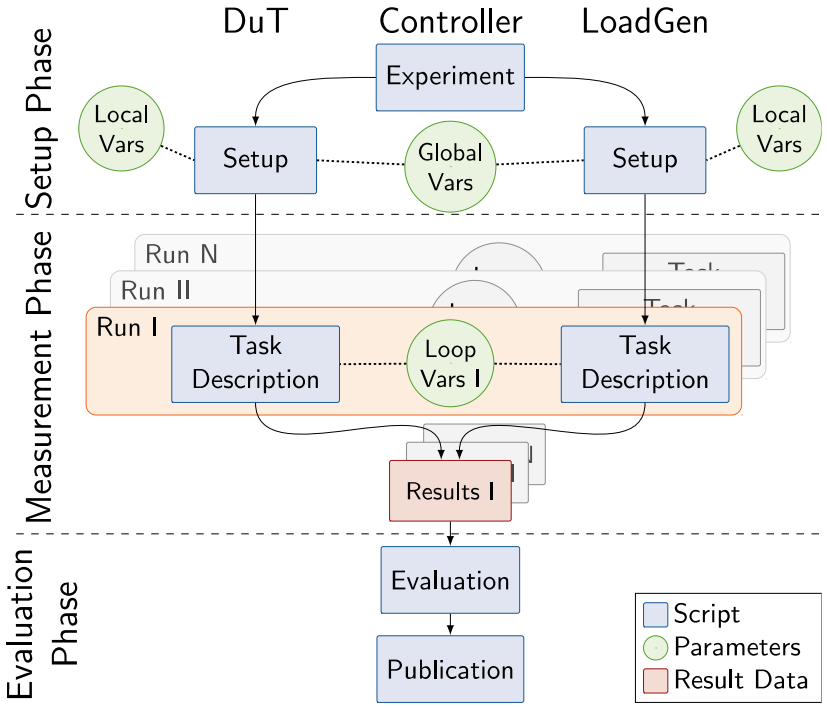


Figure 10: Experiments according to the pos workflow

[9] Github (app aware 5g), https://github.com/teo-tsou/app_aware_5g, [Last accessed 30 August 2022]

[10] Sebastian Gallenmüller*, Dominik Scholz*, Henning Stubbe, Georg Carle, "The pos Framework: A Methodology and Toolchain for Reproducible Network Experiments," in The 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '21), Munich, Germany (Virtual Event), Dec. 2021. https://dl.acm.org/doi/pdf/10.1145/3485983.3494841, [Last accessed 30 August 2022]

[11] Github, Pos-artifacts-experiment, https://github.com/gallenmu/pos-artifacts/tree/gh-pages/experiment [Last accessed 30 August 2022]

Setup Phase

The testbed controller host executes the main *experiment* script that interacts with the pos API to execute multiple actions. First, it allocates the desired devices, e.g., the DuT and LoadGen. The testbed uses a calendar-based reservation system for the experiment hosts. Devices are configured by loading the *global*, *local*, and *loop vars.* Moreover, a live image and boot parameters can be set for every device. The experiment script instructs pos to start the devices, whereby the boot is executed using the initialization interface. Once the experiment hosts have finished booting, pos deploys a set of utility tools before the setup scripts can be loaded and executed to complete the setup phase. These tools can be used in the setup or task description scripts; read or set *vars* and synchronize the different experiment hosts.

The scripts in pos workflow can be written in any programming language that can be executed on the respective experiment hosts. We recommend using either bash or python because of their wide adoption, that ensures that the scripts can be easily read and adopted by other researchers. For the configuration of the experiment nodes, specialized management tools exist. The high flexibility of pos allows using such specific configuration management systems, e.g., Ansible[12], Chef[13], or Puppet[14]. However, the user can also choose to use a simple shell script for the configuration of experiment hosts.

Measurement Phase

After the setup phase has finished, pos executes different *runs*, i.e., multiple executions of the *task description* script for every host. This task description file contains the commands that should be investigated during the experiment such as commands starting a piece of software on the on the device under test or the execution of measurement tools. The number of executions of the task description depends on the number of individual parameters in the *loop vars* file. pos experiments perform measurements for each possible combination of loop parameters. If lists are used as parameters, pos automatically generates the cross product over all parameter values. For every set of values contained in the calculated cross product, it executes the task description script once. The complete output of the experiment script is captured and stored in the result folder of the experiment. This enforced central collection of artifacts, including the utility tools output, executed scripts, *vars*, device hardware, and topology information, guarantees publishability.

Evaluation Phase

The evaluation script typically processes the result files after all runs have been completed. For each *run*, pos creates separate result files and metadata, containing the loop parameters. Based on this metadata, the evaluation script can filter or aggregate specific parameters and values. Our plotting scripts can create throughput figures and latency distributions out-of-the-box using a set of different representations, e.g., line plots or histograms. Our structured experimental workflow allows all artifacts linked to an experiment to be connected, i.e., executed scripts, generated results, and created plots. The *publication* script bundles these artifacts into a release format, e.g., archive or repository. In addition, it generates a website and inserts all the collected artifacts documenting the experimental structure in a format that researchers can easily read.

[12] Ansible website, https://www.ansible.com/, [Last accessed 30 August 2022]

[13] Progress Chef website, https://www.chef.io/, [Last accessed 30 August 2022]

[14] Puppet website, https://puppet.com/ [Last accessed 30 August 2022]

Comparison to other testbeds

There already exist a number of testbeds and frameworks to execute experiments, such as Chameleon[15], CloudLab[16], or FABRIC[17]. European initiatives such as fed4fire[18] established a federation concept to make experiments portable across different testbeds. However, these testbeds typically offer a low-level API that allows allocating resources in testbeds but does not provide support to realize sophisticated experiments using complex software setups or experiment workflows. If needed researchers can create experiment-specific software to take care of the high-level experiment workflow. Reproducible experiments can be created in these existing environments. However, due to a lack of standardized structures or workflows there is no process that ensures that experiments created and executed in the existing environments are reproducible.

There exist solutions, called experiment controllers, that allow the realization of complex experiments in a well-defined way. One of these examples is OMF[19], a framework that allows specifying experiments using a domain specific language. There is also the chameleon cloud testbed that uses Jupyter notebooks to specify and control experiments[20] as illustrated in section 5.

However, both approaches differ significantly from the pos approach. The pos approach is designed to make the experimentation process reproducible without any additional effort for the researcher. This can be achieved by relying on live images for the experiment nodes. By providing only live images, testbed users know from the beginning that they need to take care to make their experiments repeatable, i.e., using scripts to document experiment setup and execution. Testbed users are highly motivated to do so because the configuration and the state of the experiment nodes is lost after a reboot of the experiment nodes. Relying on live images further ensures that, after a reboot, each experiment is starting from a well-defined state.

Encouraging users to automate experiments and using live images to avoid residual state between experiments thereby ensure the creation of repeatable experiments from the beginning. With access to the pos controller other researchers can easily reproduce these fully automated experiments. We call this feature reproducibility-by-design.

Applying the pos experiment workflow to SLICES

The pos experiment is a refined concept that allows the specification and execution of reproducible experiments. The pos experiment controller that understands and executes these experiments can therefore be seen as a generic API to create reproducible experiments. This API act as a reference

[15] Chameleon website, https://www.chameleoncloud.org/, [Last accessed 30 August 2022]

[16] Duplyakin, Dmitry, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller et al. "The Design and Operation of {CloudLab}." In *2019 USENIX annual technical conference (USENIX ATC 19)*, pp. 1-14. 2019.

[17] Baldin, Ilya, Anita Nikolich, James Griffioen, Indermohan Inder S. Monga, Kuang-Ching Wang, Tom Lehman, and Paul Ruth. "Fabric: A national-scale programmable experimental network infrastructure." *IEEE Internet Computing* 23, no. 6 (2019): 38-47.

[18] Fed4FIRE+ website, https://www.fed4fire.eu, [Last accessed 30 August 2022]

[19] Rakotoarivelo, Thierry, Maximilian Ott, Guillaume Jourjon, and Ivan Seskar. "OMF: a control and management framework for networking testbeds." *ACM SIGOPS Operating Systems Review* 43, no. 4 (2010): 54-59.

[20] Anderson, Jason, and Kate Keahey. "A case for integrating experimental containers with notebooks." In *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 151-158. IEEE, 2019.

implementation for reproducible experiments: a potential candidate for a future SLICES experiment API supported by the participating testbeds.

At TUM, pos currently runs on a testbed and uses a custom backend to control and execute experiments. We are currently in the process of extending pos to different backends. A highly attractive backend is the geni-lib. Geni-lib is a widely used library to control and manage testbeds, supported by testbeds such as CloudLab, OneLab, or the Chameleon testbeds. Figure 11 shows the planned pos experiment stack. The *pos experiment* on top is executed on the *pos experiment controller*, using the *geni-lib* as its backend, which itself is hosted by one of the previously mentioned testbeds.

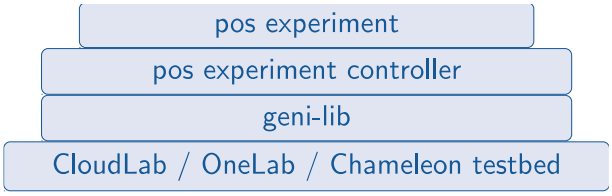


Figure 11: Pos experiment stack

Different deployment models for the pos controller are currently investigated:

- The first deployment model installs the experiment controller on a testbed hosting an experiment. In this case, the pos experiment controller is deployed inside a testbed for the execution of a single experiment. After the experiment is executed, the pos experiment controller is removed from the testbed;
- The second deployment model uses a pos experiment controller that executes an experiment remotely. There, the pos controller is hosted on a server outside of the actual testbed hosting the experiment. The pos controller controls the testbed and the experiment workflow remotely. In this deployment model the controller does not need to be deleted after an experiment execution;
- The third deployment model uses a pos experiment controller that is part of the host testbed. In this case the testbed provides the facilities to run the experiment controller as separate machine not part of an experiment. There the same controller can be used for multiple experiments over a longer period of time.

Additional research is necessary to find the best deployment model. There is also the possibility to support multiple deployment models in parallel. Testbeds part of SLICES may tightly integrate the pos experiment controller and therefore use the third deployment model. Non-European testbeds that prefer their own experiment controller may be used as part of a SLICES experiment using the first or the second deployment model.

Some proofs of concept are currently being developed in order to better identify the challenges ahead for a broader adoption.

Jupyter notebooks and pos experiments

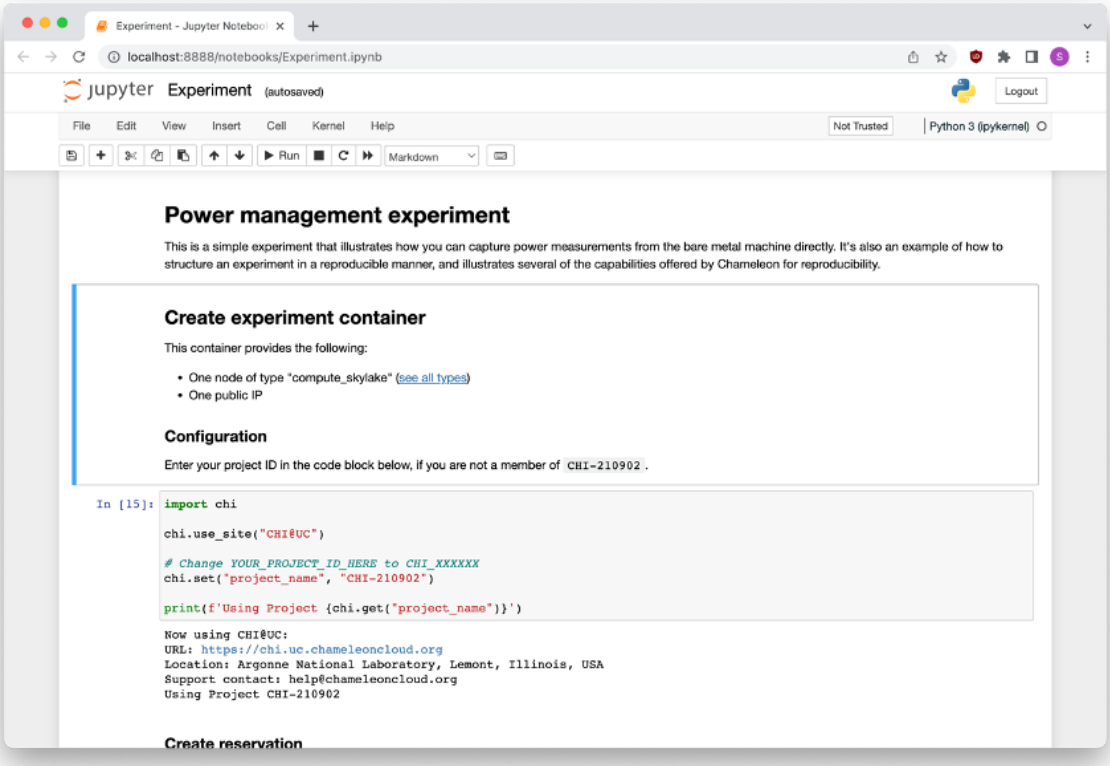


Figure 12: Network experiment in a Jupyter notebook

Figure 12 shows a typical example of a Jupyter notebook. Jupyter notebooks are typically opened and viewed via web browser. The content of the Jupyter notebook consists of subunits called cells. The content of these cells is highly flexible, typical examples include text, graphics, or executable code. The example in Figure 12 shows a text cell highlighted with a blue border. The cell below the highlighted one shows a short Python script with a light gray background. In addition to displaying code, Jupyter notebooks also can execute the code contained in its cells. Figure 12 shows the output of the script directly below the code cell. Jupyter notebooks are not limited to Python scripts, shell scripts are also supported, as are a number of other languages[21]. Another typical use case for Jupyter is the evaluation and direct graphical evaluation of data in the browser[22]. In summary, Jupyter notebooks provide a way to bundle, execute, and present textual and visual information in a web browser.

Because of these previously mentioned, attractive features, the Chameleon testbed supports Jupyter notebooks as format for specifying, executing, and evaluating experiments[23]. In addition, the ability to display and execute Jupyter notebooks in a browser lowers the entry barrier for new testbed users. The chameleon authors claim that Jupyter notebooks simplify the reproduction of experiments as

[21] Github (Jupyter kernels), https://github.com/jupyter/jupyter/wiki/Jupyter-kernels [Last accessed: 06 September 2022]

[22] Github (pos-artifacts/plot scripts), https://github.com/gallenmu/pos-artifacts/tree/gh-pages/plot_scripts [Last accessed: 06 September 2022]

[23] Anderson, Jason, and Kate Keahey. "A case for integrating experimental containers with notebooks." 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, 2019.

users can reexecute the Jupyter notebooks of previously specified experiments. The example in Figure 12 shows an example experiment of the Chameleon testbed, that is publicly available for others to reproduce.

A typical pos experiment is distributed across different files[24]. These files reflect the pos workflow shown in Figure 1. In this workflow, specific files are used for a specific purpose, e.g., separate scripts are used for setup and measurement. There also is a clear separation between scripts and variable files, separating the code from the parameters of an experiments. With each file in the pos workflow having a distinct purpose we try to make experiments more easily understandable for users. The Chameleon approach does not follow such a well-structured approach, i.e., each Jupyter notebook is structured in a different way.

Chameleon and pos follow different approaches regarding the specification of experiments: bundled in Jupyter notebooks vs. distributed across different files. However, both approaches are not mutually exclusive. The separate files of the pos experiment format can be specified as cells of a Jupyter notebook, thereby combining both approaches. Due to the fact that Jupyter notebooks support shell scripts within cells, other pos specific information, such as variables, also can be specified via separate cells and handed over to scripts in other cells. Thereby, the clear structure of pos experiment workflows can be kept inside of the Jupyter notebook experiment file. The basic compatibility of both approaches is demonstrated as part of the pos reference example. This example already contains Jupyter notebooks as specific parts of the workflow, specifically experiment evaluation to create plots[25].

Regarding reproducibility Chameleon and pos follow a similar approach, that strongly encourages users to automate the entire workflow of the experiment, from specification, execution and finally evaluation. However, using live images, pos users are highly aware of the need to automate experiment configuration and execution. This is an additional push towards reproducibility unique to the pos approach.

To conclude, Chameleon and pos pursue the same goals making experiments reproducible and easily understandable but follow different paths. The Jupyter-driven approach can help to present fully automated experiments in a simple and highly flexible manner. The pos-driven approach helps users that are familiar with its structure to get a quick overview, using live images as a powerful motivator towards automated experiment execution. Despite their differences, a combination of pos' and Chameleon's approaches is possible and should be closer investigated in the future.

[24] Github (pos-artifacts/plot scripts), https://github.com/gallenmu/pos-artifacts [Last accessed: 06 September 2022]

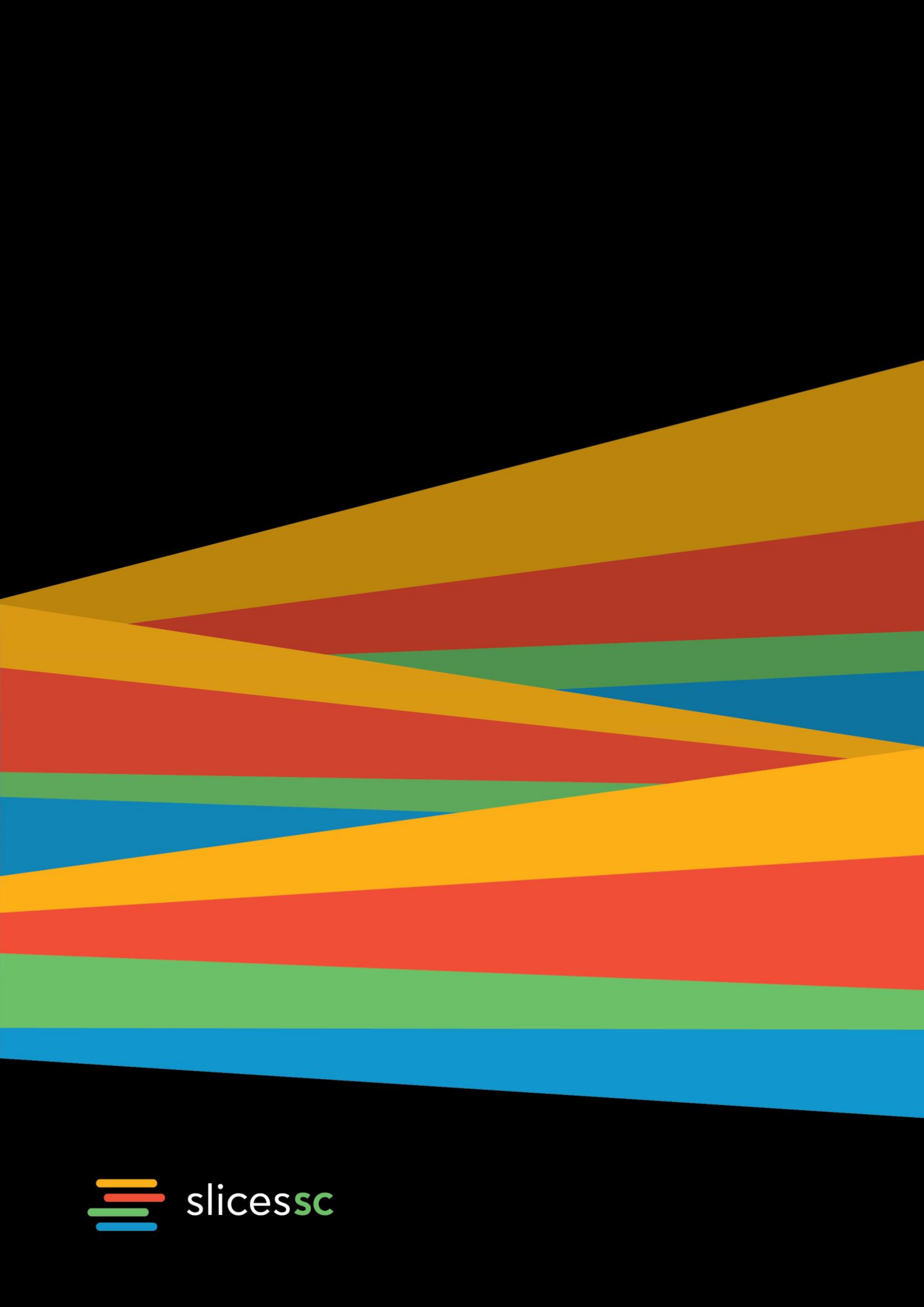
[25] Github (pos-artifacts/plot scripts), https://github.com/gallenmu/pos-artifacts/tree/gh-pages/plot_scripts [Last accessed: 06 September 2022]

9. Conclusion

A scientific instrument, a test platform, is not limited to a testing infrastructure but should also onboard the full-research life cycle. This includes difficult issues such as open and fair data, data management, liaison with EOSC and reproducibility.

Some initiatives have developed in the field but with limited impact for obvious reasons related to the complexity of the problem and the lack of reward in engaging resources into this task. Nevertheless, aligning with the best practices as developed in other more mature domain of sciences provides incentives for dealing more seriously with this concern.

In particular, the FAIR and EOSC framework in Europe could provide a catalyst and methodology to progress in this domain. This is the ambition of SLICES to propose a full framework to support reproducibility. This is a huge and longer-term challenge that will certainly require the adoption of incentives and policies in order to fully engage our research community.


slicessc