HORIZON 2020
H2020 - INFRADEV-2019-3

D4.5 SLICES infrastructure and services integration with EOSC, Open Science and FAIR: Recommendations and design patterns (final report)

Acronym	SLICES-DS
Project Title	Scientific Large-scale Infrastructure for Computing/Communication Experimental Studies – Design Study
Grand Agreement	951850
Project Duration	24 Months (01/09/2020 – 31/08/2022)
Due Date	31 August 2022 (M24)
Submission Date	12 September 2022
Authors	Yuri Demchenko, Panayiotis Andreou, Brecht Vermeulen, Cedric Crettaz, Mathias Kirkeng, Christian Perez
Reviewers	All partners

Scientific Large-scale Infrastructure for Computing Communication Experimental Studies **Design Study**

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 951850. The information, documentation and figures available in this deliverable, is written by the SLICES-DS 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

This final WP4 deliverable summarizes all developments and describes design decisions made to ensure SLICES-RI interoperability and integration with EOSC and with other RIs and testbeds to enable new technologies and large-scale experimentation. The deliverable describes work done by the SLICES-DS project in developing architecture and technical solutions for integration and compatibility of the SLICES-RI with EOSC and other testbeds, as well as the adoption of the FAIR data principles in experimental data management.

The deliverable includes a short summary of the SLICES Interoperability Framework and recommendations to SLICES Architecture related to internal and external infrastructure interoperability for experimental data management and also interoperability with EOSC and external RIS that may include cloud and edge computing, data centre access network, distributed data network, IoT sensor networks, terrestrial network infrastructure, radio access network.

The deliverable provides an overview and reports on the assessment of the cloud native solutions and development platforms to ensure interoperability and fast integration of the SLICES services and infrastructure with EOSC and external RIs. A hybrid cloud deployment model is suggested that allows using public clouds such as AWS or Azure for development and solution testing, also for rapid experimentation, and distributed private cloud infrastructure to be created by SLICES that would be used primarily for running internal experimental infrastructure and supporting experimental data management. In this context, the deliverable analyses DevOps and cloud integration tools to define and manage design patterns for interoperability and integration. Basic design patterns to support interoperability and integration have been identified and tested in the laboratory development environment. Code examples are provided in appendices and available on the WP4 development github.

The deliverable analyses the required infrastructure and data management components to support the whole experimental research lifecycle and corresponding data management infrastructure and services. This includes both experiment workflow description, data modeling, and experimental data management, starting from the data collection and storing to data processing, data lineage/provenance, and data publication or archiving. Special attention is given to using experiment workflow definition and data description that allow for experiment reproducibility and portability. Options with github, Jupyter Notebooks and Common Workflow Language (CWL) were discussed.

The D4.5 Deliverable summarises requirements and proposed solutions for SLICES general data management services, procedures and policies, including metadata definition, management and publication/registration with a focus on Open Science and FAIR data principles adoption and compliance. The deliverable also reports on the updates made to the Data Management Plan (delivered in D4.1 in M6) based on the best practices in the DMP specification among European RIs and EOSC.

The presented results on the interoperability and integration with EOSC and external RIs, adoption of the FAIR data principles in the SLICES-RI have been supported by the active involvement of the project partners in the activity of EOSC, other related European and international initiatives to ensure the best use of existing services for the research community as well as smooth data sharing to increase the efficiency of research and address needs of the European research community.

Table of contents

1. Introduction

This final WP4 deliverable summarizes all developments and describes design decisions made to ensure SLICES-RI interoperability and integration with EOSC and with other RIs and testbeds to enable new technologies and large-scale experimentation. The deliverable describes work done by the SLICES-DS project in developing architecture and technical solutions for integration and compatibility of the SLICES-RI with EOSC and other testbeds, FAIR data management infrastructure and external RIs.

The final deliverable is based on the analysis and proposed solutions presented in the previously submitted Deliverables D4.2 SLICES infrastructure and services integration with EOSC and Open Science, D4.3 Definition of the SLICES metadata profiles to support FAIR principles and D4.4 SLICES infrastructure articulation with NGI and international testbeds.

The main focus in WP4 was on the interoperability with EOSC services that primarily include data exchange and sharing compliant with the FAIR data principles, EOSC services access and publication of the future SLICES services in the Catalog maintained by EOSC (and operated by EGI on behalf of EOSC).

The D4.5 Deliverable includes a short summary of the SLICES Interoperability Framework and recommendations to SLICES Architecture related to internal and external infrastructure interoperability for experimental data management and also interoperability with EOSC and external RIS.

Aspects related to interoperability and integration with external RIs include both infrastructure aspects and data exchange and sharing during the distributed and large-scale experiments that may include cloud and edge computing, data centre access network, distributed data network, IoT sensor networks, terrestrial network infrastructure, radio access network.

The deliverable provides an overview and reports on the assessment of the cloud native solutions and development platforms to ensure interoperability and fast integration of the SLICES services and infrastructure with EOSC and external RIs. A hybrid cloud deployment model is suggested that allows using public clouds such as AWS or Azure for development and solution testing, also for rapid experimentation, and distributed private cloud infrastructure to be created by SLICES that would be used primarily for running internal experimental infrastructure and supporting experimental data management. In this context, the deliverable analyses DevOps and cloud integration tools to define and manage design patterns for interoperability and integration. Basic design patterns to support interoperability and integration have been identified and tested in the laboratory development environment. Code examples are provided in appendices and available on the WP4 development github.

The deliverable analyses the required infrastructure and data management components to support the whole experimental research lifecycle and corresponding data management infrastructure and services. This includes both experiment workflow description, data modelling, and experimental data management, starting from the data collection and storing to data processing, data lineage/provenance and data publication or archiving. Special attention is given to using experiment workflow definition and data description that allow for experiment reproducibility and portability. Options with github, Jupyter Notebooks and Common Workflow Language (CWL) were discussed.

The D4.5 Deliverable summarises requirements and proposed solutions for SLICES general data management services, procedures and policies, including metadata definition, management and publication/registration with a focus on Open Science and FAIR data principles adoption and compliance.

The deliverable also reports on the updates made to the Data Management Plan (delivered in D4.1 in M6). This includes the adoption of the best practices in the DMP specification among European RIs and EOSC, better data stewardship functions definition for experimental data management and update on privacy and ethics.

The presented results on the interoperability and integration with EOSC and external RIs, adoption of the FAIR data principles in the SLICES-RI have been supported by the active involvement of the project partners in the activity of EOSC, other related European and international initiatives to ensure the best use of existing services for the research community as well as smooth data sharing to increase the efficiency of research and address needs of the European research community.

2. SLICES Interoperability Framework and supporting infrastructure elements

2.1. SLICES Interoperability Framework

SLICES Interoperability Framework (SLICES-IF) is proposed in the Deliverable D4.2 and defined a set of requirements, infrastructure services and recommendations to ensure SLICES-RI interoperability and future services integration with EOSC services and infrastructure. The SLICES-IF considers SLICES-RI as a programmable and remotely accessible digital research infrastructure comprising of geographically distributed compute, storage and networking (wired and wireless) elements. From the use case perspective, SLICES-RI will primarily enable large scale experimental research in the following categories:

- **Distributed cloud/edge computing**: under this category, experiments regarding infrastructure operation and management are considered (e.g., management of virtualized resources), as well as applications of Machine Learning and Artificial Intelligence that can run in a distributed manner (e.g., Federated Learning);
- **IoT verticals**: under this category of experiments regarding applications and ubiquitous IoT networking, interoperability of sensor infrastructure and integration of IoT infrastructure with the rest of the stack are envisioned to take place;
- **Wired/wireless networking**: experiments regarding prototyping next generation protocols for network interconnection (e.g., researching new waveforms and new frequencies, spectrum and wireless network management, formation of Heterogeneous networks) are foreseen under this category.

Figure 1 illustrates architectural elements of the SLICES-IF to support the listed above requirements. The left side illustrates the main components of the compute, storage network infrastructure which implementation will use cloud based and cloud native services model that allows services composition and deployment on demand benefiting from rich management and monitoring functionality of the modern cloud platforms, both public and private. The right side of the figure illustrates experimental facilities and testbeds which access for the research community will be provided by SLICES-RI: IoT verticals and edge computing, industrial automation testbeds and AI enabled services, experimentation facilities for Radio Access Network, external RIs and testbeds.

The core SLICES infrastructure services will ensure smooth services integration, resources management and on-demand provisioning. When using cloud native services model commonly accepted in 5G and modern networking technologies, all use cases can be combined in a common services provisioning and orchestration model using cloud based or edge-based services and microservices, benefiting from well-developed cloud automation and DevOps practices supported by industry accepted service architecture and numerous development and management platforms.

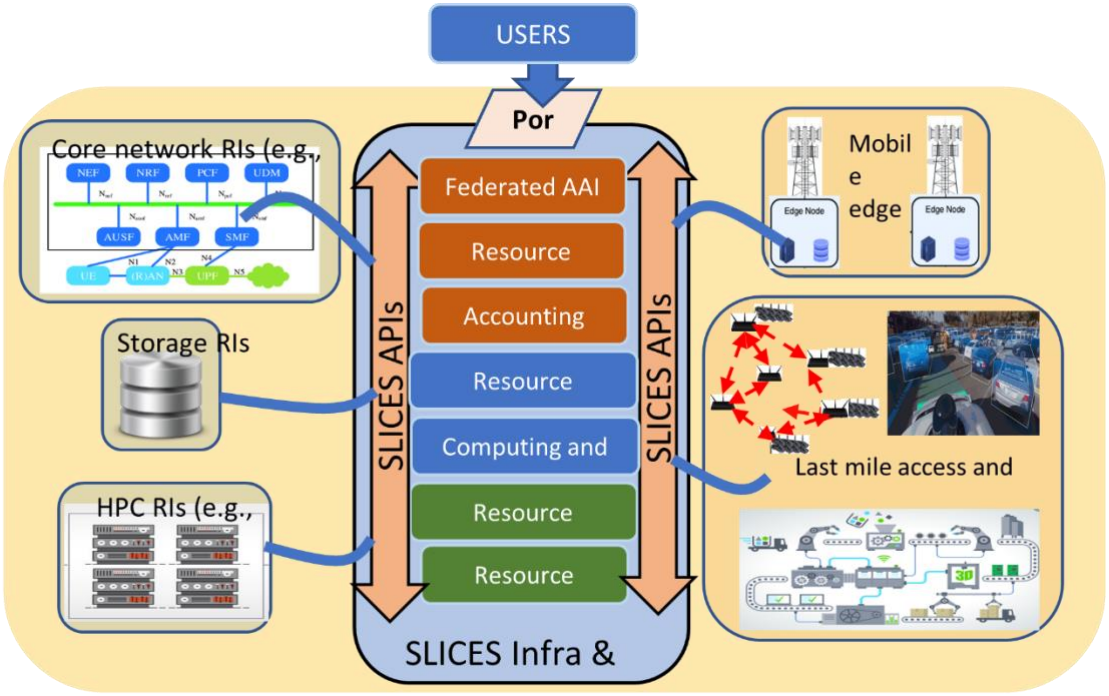


Figure 1 – SLICES-RI infrastructure components

The proposed in D4.2 SLICES Interoperability Framework enables the following features:

I. Support integration into a distributed pan-European research infrastructure that shall be accessible remotely.
II. Allow deploying and operating large scale multi-site experiments on SLICES infrastructure, including external services and interaction with other RIs and testbeds.
III. Use well defined, where possible standard, interfaces/API that support both services/resources interaction and data access and sharing (e.g., research data, computing and storage modules, providers and users)
IV. Comply with the FAIR data principles where interoperability is essential to ensure that the SLICES data can be integrated with the scientific workflows for analysis, storage and processing
V. Interaction with EOSC and other RIs to allow external services to be used in SLICES-RI and SLICES-RI services are accessible by external RIs and researchers; the goal is to consolidate the scattered experimental facilities and European research infrastructures.

SLICES-IF defines four groups of interoperability recommendations in compliance with the EOSC Interoperability framework:

- **Technical interoperability** is defined as the ability of different information technology systems and software applications to communicate with each other and seamlessly exchange data.

- **Semantic interoperability** refers to the ability that the exchanged data to be understood well and have a common meaning across different entities of the EOSC ecosystem.
- **Organizational interoperability** is focused on the alignment of organizational policies, functions, responsible people, documentation and processes across different EOSC service providers. The main emphasis is on defining a governance framework to achieve cross-organizations and cross-discipline interoperability.
- **Legal interoperability** primarily concerns data access governed by various forms of intellectual property rights (e.g., licensing, copyrights, etc.), general data protection regulation (GDPR), private and sensitive data and enabling legal instruments.

A detailed list of all recommendations is provided in Table 1, which is derived from the EOSC IF recommendations, while later in this document, we will focus on the technical and semantic interoperability that are relevant to experimental infrastructure and data management services.

Table 1 – SLICES Interoperability Framework Recommendations

Layer	Recommendation
Technical (services but not yet infra)	• Open specifications for SLICES services published on the SLICES Portal (adopting and ensuring interoperability with EOSC services description). • A common security and privacy framework (including Authorisation and Authentication Infrastructure). • Easy-to-understand Service-Level Agreements for all SLICES resource providers. • Easy access to data sources available in different formats. • Coarse-grained and fine-grained dataset (and other research objects) search tools. • Interoperability and integration with the EOSC PID infrastructure, compliance with EOSC PID policy.
Semantic (Metadata)	• Clear and precise, publicly-available definitions for all concepts, metadata and data schemas. • Semantic artifacts, preferably with open licenses. • Associated documentation for semantic artifacts. • Repositories of semantic artifacts, rules with a clear governance framework. • A minimum metadata model (and crosswalks) to ease discovery over existing federated research data and metadata. • Extensibility options to allow for disciplinary metadata. • Clear protocols and building blocks for the federation/harvesting of semantic artifacts catalogues.
Organisational	• Interoperability-focused rules of participation recommendations. • Usage recommendations of standardised data formats and/or vocabularies, and with their corresponding metadata. • Clear management of permanent organisation names and functions

Legal	• Standardised human and machine-readable licenses, with a centralised source of knowledge and support on copyright and licenses. • Permissive licenses for metadata (and preferably for data, whenever possible), where is CC0 preferred over CC BY 4.0. • Identification of different parts of a dataset with different licenses. • Clearly marked instances of expired or non-existent copyright, as well as for orphan data. • Compatibility with EOSC-recommended licenses and hosting national regulations. • Tracking of license evolution over time for datasets. • Harmonised policy and guidance to dealing with cases where patent filing or trade secrets may be compromised by disclosure. • GDPR-compliance for personal data. • Additional restrictions on access and use of data only applied in cases of applicable legislation or legitimate reasons. • Harmonised terms of use across repositories • Alignment between Member States national legislations and EOSC.

Figure 2 illustrates a conceptual view of the SLICES-IF to support listed above recommendations that include both SLICES-RI core services and services to support technical and semantic interoperability.

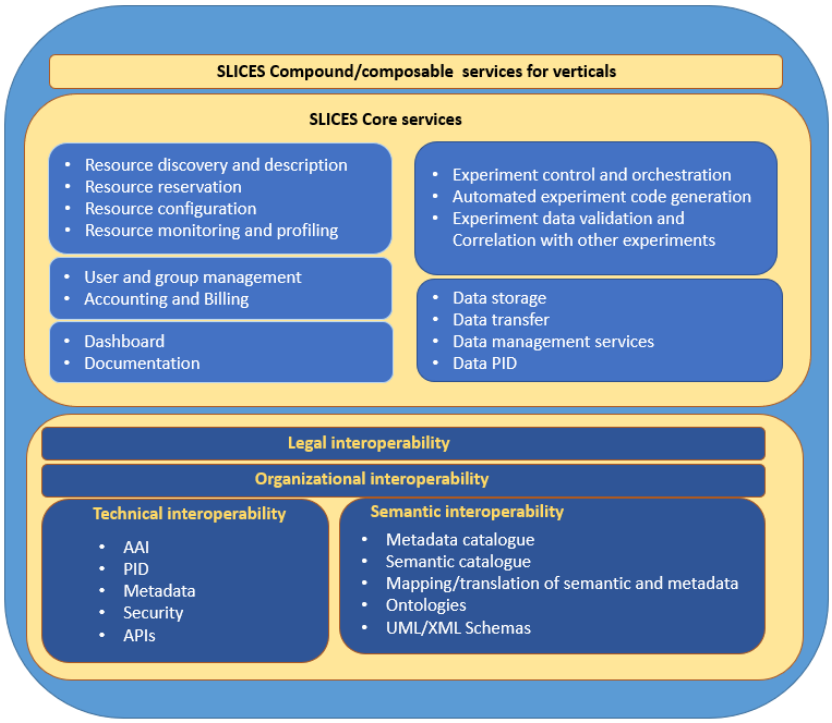


Figure 2 – Conceptual view of SLICES Interoperability Architecture

The following design principles and common building blocks are defined and should be implemented in the SLICES-RI to ensure interoperability and integration with EOSC and external RIs[1]:

For services interoperability:

a) Common services model that uses REST services as building blocks;
b) Correspondingly, services interactions are done via formally defined and published Application Programming Interfaces (APIs) implementing HTTP based REST protocol;
c) Authentication and Authorization Infrastructure (AAI) and Identity Provider (IDP) supporting standard credentials formats and protocols such as SAML 2.0, OpenID Connect, OAuth 2.0 and X.509v3, which are supported in EOSC AAI;
d) Cloud native and container based microservices architecture as a platform for services design, integration, deployment and operation supported by portable design artifacts.

For data interoperability:

a) FAIR Digital Object (FDO) for effective data discovery and management, including FDO policy defined for specific SLICES-RI experimentation domains;
b) Persistent IDentifiers (PID) and supporting infrastructure for reliable data access and transfer
c) Common metadata models and formats;
d) Experiment description using standard markup format or executable object is provided as a part of metadata to ensure full reproducibility and mobility;
e) Metadata registry to support FAIR data access and sharing;
f) Semantic artifacts to support consistent and interoperable data/metadata definition and rendering for basic use cases and experiments.

The Platform Research Infrastructure as a Service (PRIaaS) architecture proposed in the authors' research paper[2] can be used for the practical implementation of the SLICES Interoperability Framework. Refer to Appendix A. The Platform Research Infrastructure as a Service (PRIaaS) for a short summary on PRIaaS to section 4.1 for its relevance to experimental infrastructure workflow.

Common building blocks must be supported by services deployment templates and design patterns organized as a library for future reuse and infrastructure services for automated experiments deployment and management. Service deployment templates and popular deployment automation tools are discussed in section 3.

2.2. Recommendations to SLICES-RI Architecture (WP2)

WP4 and D4.2 primarily focused on technical and semantic interoperability aspects which have an impact on SLICES-RI Architecture. Below we summarise the technical and semantic requirements and architecture.

[1] It is assumed that external RIs comply with the general four-layers interoperability model (defined in the EOSC Interoperability Framework) and implement the standard REST services model and APIs

[2] Yuri Demchenko, Cees de Laat, Wouter Los, Future Scientific Data Infrastructure: Towards Platform Research Infrastructure as a Service (PRIaaS), Proc. The International Conference on High Performance Computing and Simulation (HPCS 2020), 10-14 Dec 2020.

Technical interoperability includes the following requirements and recommendations:

I. EOSC recommends that all the services should use open specifications whenever possible;
II. A common security and privacy framework including a common authentication, authorization mechanisms should be used;
III. A clear policy for persistent identifiers (PID) for research data, infrastructure and software, should be defined;
IV. Data should be made available in a different format for ease of accessibility.

Semantic interoperability includes the following requirements and recommendations:

I. All the concepts, metadata and schemas should use clear, precise and publicly available definitions which are referenced with PIDs;
II. A minimum metadata model should be used to describe all the research data for ease of discovery;
III. Metadata should have extensibility options for the inclusion of domain specific information;
IV. Semantic artifacts should contain the associated documentation.

2.3. EOSC Services and Resources for SLICES-RI

Implementing SLICES-IF and compliance with the EOSC-IF will ensure interoperability of the SLICES and EOSC services and will allow using services and resources offered by EOSC community (via EOSC Portal Marketplace and Services and Resources Catalog[3]) as part of the general SLICES-RI and in deploying individual experiments.

At this stage, we see the benefits of using a wide range of services offered by EOSC and European RIs in the SLICES infrastructure and experiments, which however can be short running and long running and requiring both persistent and on-demand resources. The following services and resources have been identified that potentially can be used and effectively integrated with the SLICES infrastructure and used on-demand in individual experiments (refer for details to the EOSC Services and Resources Catalog):

- EGI Foundation resources:
  - EGI DataHub: Access key scientific datasets scalable;
  - EGI Cloud Compute: Run virtual machines on-demand with complete control over computing resources;
  - EGI Cloud Container Compute: Run Docker containers in a lightweight virtualised environment;
  - EGI High-Throughput Compute: Execute thousands of computational tasks to analyse large datasets;
  - EGI Online Storage: Store, share and access your files and their metadata on a global scale;
  - EGI Check-In: Proxy service that operates as a central hub to connect federated Identity Providers (IdPs) with EGI service providers.

[3] EOSC Market Place website, https://marketplace.eosc-portal.eu/ and https://marketplace.eosc-portal.eu/services, [Last accessed 26 August 2022].

- GEANT Association network connectivity services
  - GÉANT Open, GÉANT Open, GÉANT Plus, GÉANT Lambda: High-performance interconnectivity for the most demanding networking requirements;
  - GÉANT MDVPN: Increased privacy and control - helping to build effective virtual teams across borders;
  - Clouds Service: Enabling Institutions to access virtualised commercial cloud services
  - eduGAIN: Unlocking global research and education collaboration;
  - eduroam: Seamless Wi-Fi access for research and education around the world.
- Other infrastructure services that can be used as external services at the initial stage of the SLICES-RI deployment but should be replaced by SLICES own services at later stage:
  - INDIGO Identity and Access Management (IAM), provided by Italian National Institute of Nuclear Physics (INFN)
  - B2HANDLE: Distributed service for storing, managing and accessing persistent identifiers (PIDs) and essential metadata (PID records) as well as managing PID namespaces.

To ensure interoperability between SLICES and EOSC information services and smooth data sharing, the SLICES-RI is planning to use Open-Source implementation of the EOSC Catalog for deploying the SLICES services catalog that can be federated with the EOSC Catalog. Such experience already exists in the EOSC community and some RIs.

Design patterns for consuming/integrating EOSC services via development, integration and deployment tools (CI/CD process) are discussed in section 3.

2.4. Interaction with external RIs and testbeds

Although EOSC is providing a common cooperative platform to facilitate interaction between European research infrastructure, organisations and projects, SLICES will need cooperation with multiple research and experimental facilities outside Europe. The work was done in WP4 and reported in the Deliverable D4.4 to identify and study existing European and international research infrastructure and testbeds, which are important for the future SLICES experimental research on advanced infrastructure and digital technologies, in particular the Next Generation Internet (NGI) European testbeds and US infrastructures. The report analysed different aspects of interaction and interoperability with external experimental facilities including technical connection, governance, ad-hoc collaboration and guided collaboration (based on coordinated funding). The following infrastructures were studied: Fed4FIRE – NGI, PlanetLab, EOSC, GENI, Emulab, CloudLab, Chameleon, FABRIC, PAWR, EMPOWER, PRACE.

The following forms of collaboration and technical interconnectivity with other experimental infrastructure and testbeds have been identified and analysed:

- **Governance (cooperation) agreement**: Different levels of collaboration can be defined at the governance level. As an example, SLICES is developing a collaboration effort at the international level, and in particular with the USA based infrastructures, testbeds and projects such as FABRIC, BRIDGES, Chameleon. Formalizing cooperation agreement can open wider cooperation and facilitate joint research.
- **Technical cooperation at the level of services interconnection**: All studied examples allow technical cooperation and interconnection based on standard API or community defined

standardized API, what is sufficient for most of cases, or using Open-Source software stacks for deeper services integration. It's important to note that large infrastructures do not use a single software stack for all sites but they tend to agree on e.g., common APIs or common user tools to make it easy for the users. Defining a common interoperability framework and set of common APIs will provide a ground for future proof collaboration.

- **Ad-hoc collaboration**: SLICES should be open for ad-hoc collaboration, e.g., for running dedicated demonstrations or tests, but structured collaboration and interconnection based on cooperation agreement and common interoperability framework provide a better choice.
- **Sponsored collaboration**: this kind of cooperation is established based on the funded project in one or both parties. This is often practice of funding priority collaboration like between US and Europe or Europe and Brazil.

Standardization can also be an enabler to interact with other infrastructures based on API definitions a.o. Based on the interaction/integration with existing infrastructures, this can also have an impact on the need to support one or more software stacks for the nodes.

3. Design patterns and infrastructure deployment platform

3.1. Cloud native services model and cloud platforms

Clouds are becoming a common platform for deploying and operating large scale infrastructures to support modern services and applications, such as 5G, mobile applications, Industry 4.0 as well as scientific applications and infrastructures. All these examples benefit from the native cloud features: on-demand provisioning and rapid scalability, high automation of services provisioning, detailed services monitoring, and not the least, the global reach of cloud infrastructures offered by major public cloud providers.

Evolution of cloud hosted services model went from cloud enabled and cloud-based services that migrated monolithic or traditionally tiered applications to cloud platform to modern cloud native services model and using containers and microservices architecture. Adopting cloud-native services model brings the following benefits:

- Fast services development and deployment;
- Easy services modification and continuous improvement to response user requirements;
- Availability of many integrated development environments and tools integrated with different cloud platforms;
- Easy infrastructure and services configuration and the possibility of creating reusable services deployment templates and design patterns (or artefacts) that can be composed to create necessary services.

Cloud native approach makes it easy to use hybrid cloud services deployment model where company's infrastructure includes private cloud part for sensitive and high-availability services and public cloud for large scale computations and global customer or user reach. Services and microservices can easily migrate between private and public cloud benefiting from virtually unlimited private cloud resources and global reach. This might be beneficial for large scale experimentations where SLICES can benefit from global features of the major cloud service providers such as AWS or Microsoft Azure clouds which

also offer vast free resources such as AWS Free Tier or Azure Free Services[4] that in many cases can be used for ad hoc experimentations.

When selecting private cloud platform, the best solution could be OpenStack which already used by majority of RIs and EOSC members as a cloud platform for services deployment. OpenStack have reached production maturity with the functionality comparable to major private cloud platforms. There is a rich experience with OpenStack services development and operation among European RIs and the research community[5].

Using OpenStack as a cloud platform would also simplify services and applications interoperability and integration. In a particular case with EOSC interoperability and integration, there is a benefit of using community developed services templates and design patterns, which can be reused as a whole deployable services stack in the SLICES-RI, individual/component services, or connected to well established EOSC infrastructure services via well-defined APIs as discussed above.

3.2. Development platforms and DevOps/SRE practices

To speed up the resources and services provisioning for on-demand provisioned experimental environment, the SLICES-RI will use modern technologies and tools for infrastructure and services deployment automation which are also supported by DevOps practices and environment. This is also a practice in many European RI and EOSC, in particular, ENVRI[6] uses DevOps for virtual infrastructure (VI) deployment and to ensure FAIR data services compliance. Typical DevOps process/pipeline for VI provisioning includes:

- Obtain and configure resources, compose infrastructure, test services;
- Deploy VI and test in pre-production environment, run adoption tests, test security aspects;
- Deploy VI and applications to the production environment, setup VI monitoring and dashboard;
- Monitor operation and provide feedback to the development team, explore sustainability options.

Essential for the successful building of SLICES-RI and its sustainable operation is to enable the whole DevOps pipeline extended with the Site Reliability Engineering (SRE) practices that allow continuous improvement of the products and services with the ultimate goal to improve quality of services and user experience. DevOps-SRE development and operation environment relies on the generic cloud-based environment for services deployment, operation and management (also called “cloud native” environment or approach) which is commonly adopted by 5G and modern telecom sector.

The development and deployment process are supported by a number of Integrated Development Environment (IDE) platforms, which are typically connected to a selected cloud platform, in particular one of the most popular public cloud platforms Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), or Open-Source cloud platforms such as OpenStack or CloudStack. It is important

[4] Azure services, https://azure.microsoft.com/en-us/pricing/free-services/, [Last accessed 26 August 2022].

[5] AWS, https://aws.amazon.com/free/, [Last accessed 26 August 2022].

[6] EOSC DevOps framework and virtual infrastructure for ENVRI-FAIR common FAIR data services https://eosc-hub.eu/research-communities/eosc-devops-framework-and-virtual-infrastructure-envri-fair-common-fair-data, [Last accessed 26 August 2022].

to mention that all cloud platforms support EC2 and S3 interfaces, which are originally defined for compute and storage resources in AWS cloud, for managing compute and storage resources.

WP4 investigated and assessed existing cloud platforms and tools supporting services deployment that can be easily adopted/configured to different cloud platforms. The primary goal was to test and get experience with the tools and platforms for automated services deployment with using one of the popular cloud platforms, in particular case AWS.

The following cloud automation tools have been reviewed and tested:

- Reviewed cloud deployment automation tools (all allow creating reusable templates for Infrastructure as Code provisioning):
  - AWS CloudFormation[7];
  - Azure Resource Manager[8];
  - OpenStack Heat Templates[9]
- Reviewed general purpose infrastructure deployment and configuration tools
  - Ansible[10] – the most popular among EOSC and RIs community;
  - Terraform[11] – popular among professional cloud-based services developers (primarily used with Azure cloud);
  - Kubernetes[12] – originally developed by Google, Kubernetes is a container orchestration platform for automating the deployment, scaling, and management of containerized applications. In fact, it's established itself as the de facto standard for container orchestration and is the flagship project of the Cloud Native Computing Foundation.
- Tested use of AWS CloudFormation for simple services deployment on AWS cloud comprising few basic services such as EC2 VM/compute, S3 Storage, database, Virtual Private Cloud (VPC) network configuration, monitoring services.
- Tested Ansible infrastructure deployment and configuration tools for both cloud-based services and general infrastructure services.
  - Sample service templates (which are called playbooks in Ansible) have been created and tested with AWS that can further be used as templates for intended design patterns.

The following sections provide more information about Ansible, Cloudformation together with examples of CloudFormation templates and Ansible playbooks for simple services and/or experiments are collected are available on the WP4/activity github https://github.com/slicesdata[13].

[7] AWS CloudFormation, https://aws.amazon.com/cloudformation/, [Last accessed 26 August 2022].

[8] Azure Resource Manager documentation, https://docs.microsoft.com/en-us/azure/azure-resource-manager/, [Last accessed 26 August 2022].

[9] Red Hat Training: Chapter 2. Understanding Heat Templates, https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/advanced_overcloud_customization/sect-understanding_heat_templates, [Last accessed 26 August 2022].

[10] Red Hat – Ansible website, https://www.ansible.com/, [Last accessed 26 August 2022].

[11] Terraform website, https://www.terraform.io/, [Last accessed 26 August 2022].

[12] Kubernetes website, https://kubernetes.io/, [Last accessed 26 August 2022].

[13] Github SLICES, https://github.com/slicesdata, [Last accessed 26 August 2022].

3.3. Cloud automation tools tested

3.3.1. AWS CloudFormation

AWS CloudFormation is a cloud-based Infrastructure as code (IaS) tool by AWS, which enables the use of templates to deploy digital infrastructure called 'stacks' using AWS resources. These templates define which resources should be present, how they should be configured and which dependencies exist between the resources. It also allows for referencing other resources within a template as well as information about these resources. Besides this the template can contain references to pseudo parameters, which provide information about the current stack as well as parameters which are provided by the user during stack creation.

These reusable stack templates are written in either JSON or YAML and can be created manually or with the help of CloudFormation Designer which is an AWS tool that provides a more visual way to create or edit templates. The template language can also be extended using custom resources, modules and hooks. Custom resources allow the use of provisioning resource types not listed by default; modules allow for small sections of preconfigured templates to be used in other templates and hooks are pieces of custom logic that inspect the template before provisioning to enforce requirements and guidelines. These extensions can also be listed in the extension registry.

From these templates multiple stacks can be launched. These stacks can also be deleted which reverses the provisioning. A stack can be created from a template alone or with existing resources, which adds existing resources into the stack along with resources defined in the template. Existing stacks can be changed using direct updates or by using stack sets. Direct updates immediately apply the changes, while update sets allow the user to the first review in which ways the resources will change before applying the update. Changes to stacks might result in a resource being recreated completely or updated in place.

3.3.2. Ansible

Ansible is a DevOps/IT automation tool that can be used to provision and configure digital infrastructure. It works using a push-based model meaning there is a central node from which remote connections are established to the client nodes, called hosts, and command line scripts are executed to change the configuration or run programs. The remote connection is usually executed via SSH, but other methods are available. A consequence of the push-based model is that the hosts don't require a special agent to be installed and only need to be set up for the remote connection.

The desired state of a given host is described in lists of tasks called a play. A task describes an action to be taken on that host. These actions are generally declarative (for example, setting the state of a package to be present), but can also be imperative (for example, by executing a command directly using the "command" module). Multiple of these plays can then be combined into a playbook, which when executed, can set up multiple hosts at the same time.

The tasks in a play are described using modules. For example, some modules are the "command" module which executes a command on the target host, the "copy" module to copy files from the central node to the target host, the "file" module to create/remove/set attributes of files and directories on the target host and the "apt" module to manage installed packages on a target host using apt. Modules are combined into collections. The main collection is "built-in" but there are also

collections for services such as AWS[14], Google Cloud[15], Cisco services[16], Kubernetes as well as community developed modules. These modules are generally developed using python[17], but can be written in any language.

Ansible also allows for setting variables for use during the execution of a playbook. These variables can be used to customize the playbook by changing parts of the configuration of tasks for each run, for example, by changing the number of hosts provisioned. They can also be used to change the way a playbook is executed by for example, changing the way the connection is established. Finally, variables can also be used with the "template" module to modify files sent to the hosts at run-time, by for example, setting the IP in a configuration file to the IP of the machine it is sent to. These variables can be set per playbook, per play or associated with specific hosts. Setting the variables can be done in a file, from the command line or be set from the output of previously executed modules in a playbook.

Ansible target hosts are managed using an inventory system. This inventory is generally defined using a static inventory file where hosts are listed in a hierarchical tree of groups. In this inventory, the DNS name or IP address of the hosts along with other arbitrary information about the hosts is saved. This inventory is evaluated using a plugin, the two most used plugins are the YAML and INI plugins, however plugins exist to read from a number of different data sources. Some plugins also enable the use of a dynamic inventory where the inventory is retrieved from a remote source at run-time. For example, the aws_ec2[18] plugin uses the boto3[19] AWS python API to automatically get all EC2 instances associated with an AWS account when running a playbook.

3.3.3. Sample experiment

To test the use of infrastructure deployment applications for digital experimentation, a simple experiment was designed in which three AWS EC2 instances communicate using the MQTT[20] IoT protocol. This experiment is split into four steps, experiment setup, experiment execution, data processing and experiment cleanup. These are illustrated in the diagram listed in Appendix D: Deployment templates for a sample experiment

In general, one instance, referred to as the publisher, emulates an IoT sensor by generating random sensor data and publishing this data on a certain topic to another instance acting as the MQTT broker. The third instance, called the subscriber, is also connected to the broker and subscribed to this topic. Thus, the data generated at the publisher is sent to the broker, which forwards it to the subscriber. The subscriber subsequently saves each sensor datapoint to an AWS DynamoDB[21] table.

In the first step, all experiment resources are provisioned. These include AWS EC2 Security Group, the three EC2 instances, an AWS DynamoDB table and an EC2 key pair (only when using Ansible). The

[14] AWS, https://aws.amazon.com/,[Last accessed 26 August 2022].

[15] Google Cloud, https://cloud.google.com/, [Last accessed 26 August 2022].

[16] Certified integration: Ansible and Cisco, Https://Www.Ansible.Com/Integrations/Networks/Cisco, [Last accessed 26 August 2022].

[17] Python, https://www.python.org, [Last accessed 26 August 2022].

[18] EC2 inventory, https://docs.ansible.com/ansible/latest/collections/amazon/aws/aws_ec2_inventory.html, [Last accessed 26 August 2022].

[19] Boto3 documentation, https://boto3.amazonaws.com/v1/documentation/api/latest/index.html, [Last accessed 26 August 2022].

[20] MQTT Specifications, https://mqtt.org/mqtt-specification/,[Last accessed 26 August 2022].

[21] Amazon DynamoDB, https://aws.amazon.com/dynamodb/, [Last accessed 26 August 2022].

security group is assigned to all three instances and ensures that they can communicate with each other and allows for SSH access. The publisher and subscriber instances are configured to have the Paho python MQTT client[22] installed as well as having a python script which runs the experiment. The subscriber instance also has the AWS boto3 python package installed to allow it to send data to the DynamoDB table. The broker instance has a Mosquitto MQTT broker[23] configured and installed. Next, the DynamoDB table has a single uuid string as a partition key. Finally, the EC2 key pair is only created when provisioned using the ansible playbook as it needs SSH access to the instances to configure them.

For this experiment, an Ansible playbook as well as a CloudFormation template were created. The ansible playbook makes use of the 'aws_ec2' dynamic inventory plugin to keep track of created AWS resources. Since this inventory is only evaluated once when the playbook is initiated but new resources are provisioned during the playbook execution the 'meta: refresh inventory' module is used to re-evaluate the current state of the inventory. As the AWS plugins use the boto3 python package to connect to AWS, this package needs to be installed on the host node and the authentication credentials need to be configured in the environment variables. The private key of the created EC2 key pair is saved to the host node so it can be used to connect to the EC2 instances. After provisioning, the playbook configures the EC2 instances, adding configuration files and installing packages.

In the CloudFormation template, the resources to be created are defined with the relevant dependencies among them. For example, the EC2 instances are dependent on the security group being created so it can be assigned to them. Configuration of the instances is handled using the AWS cfn helper scripts[24] which are installed and run using the 'UserData' field, which defines a bash script which will be executed when the instance is created and running. The cfn helper scripts read the metadata section of the template where configuration like installed packages and existing files are listed. Parameters are used to allow the user to pass the AWS credentials needed for the subscriber script to connect to the DynamoDB table.

For the experiment execution step, another Ansible playbook is created which runs the python scripts on the subscriber and publisher instances. For the CloudFormation scenario the scripts were started manually using the AWS Management console. As CloudFormation is more focused on provisioning infrastructure, it does not provide a good way to execute certain code on command. It is, however, possible to schedule this during stack creation in the 'UserData' script or having the scripts run during creation as part of the setup, though care needs to be taken to set the right dependencies and use the 'cfn-signal' helper script to inform CloudFormation when the configuration of an instance is finished.

The data processing step is described in 4.3.2. For the experiment cleanup step, another playbook was created, which simply terminates or deletes any resources created during the setup step. For CloudFormation the created stack can simply be deleted and any created resources are terminated or deleted.

[22] ECLIPSE - Paho Python Client, https://www.eclipse.org/paho/index.php?page=clients/python/index.php, [Last accessed 26 August 2022].

[23] Mosquitto, https://mosquitto.org/, [Last accessed 26 August 2022].

[24] CloudFormation helper scripts reference, https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-helper-scripts-reference.html, [Last accessed 26 August 2022].

3.4. Interacting with EOSC compatible services

3.4.1. Interacting with EOSC Catalog: Provider and Resource APIs

The EOSC Provider and Resource APIs (https://providers.eosc-portal.eu/openapi)[25] are the external APIs that allow registration and discovery of a provider and a resource in the EOSC Registry serving the EOSC Catalog. Includes API for provider-controller, resource-controller, vocabulary-controller. Technical details on the EOSC Provider and Resource APIs are provided in Appendix B. EOSC Provider and Resource APIs

3.4.2. Interacting with EOSC Marketplace

The EOSC Marketplace (MP) enables establishing provider/consumer relations on published services, with the maximum process and negotiation automation. This process is supported by Order Management System (OMS) SOMBO[26] that interacts with users/customers via Catalog, Offer Configuration and User Space management, as illustrated in Figure 3. Communication between the user as a consumer and a provider offering service is supported by Marketplace APIs.

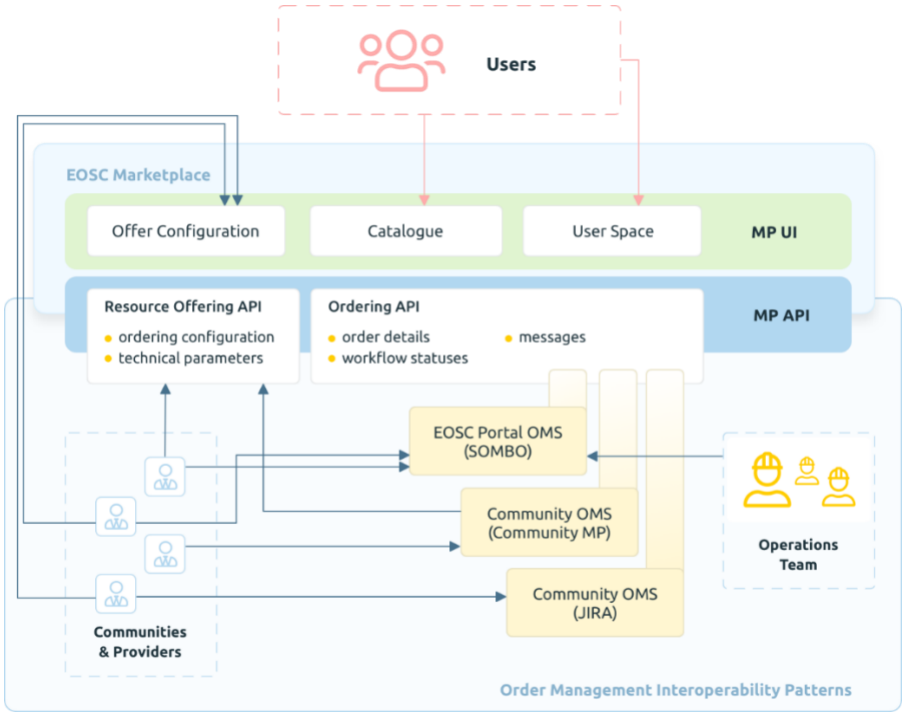


Figure 3 – EOSC marketplace API architecture

The API of the EOSC marketplace enables two functions: offering and ordering by resource/service providers and communities. The users are presented with a consistent EOSC marketplace User Interface while providers, communities and operations can utilize the APIs to ensure interoperability.

[25] EOSC Provider and Resource APIs, https://providers.eosc-portal.eu/openapi, [Last accessed 26 August 2022].
[26] "SOMBO, the order management system for EOSC Portal, is now in production", https://eosc-portal.eu/news/sombo-order-management-system-eosc-portal-now-production, [Last accessed 26 August 2022].

Providers and communities (aka Integrators), can choose from a minimum integration level facilitated by the marketplace UI through the SOMBO for order management or can opt for more complex API based integration with the EOSC marketplace APIs for a fine-grained order and offering management.

The SOMBO reference implementation[27] is based on the JIRA system[28] (https://github.com/cyfronet-fid/oms-adapter-jira) and contains Marketplace API client which is responsible for communication with the Marketplace Ordering API during the order management process (see Figure 4).

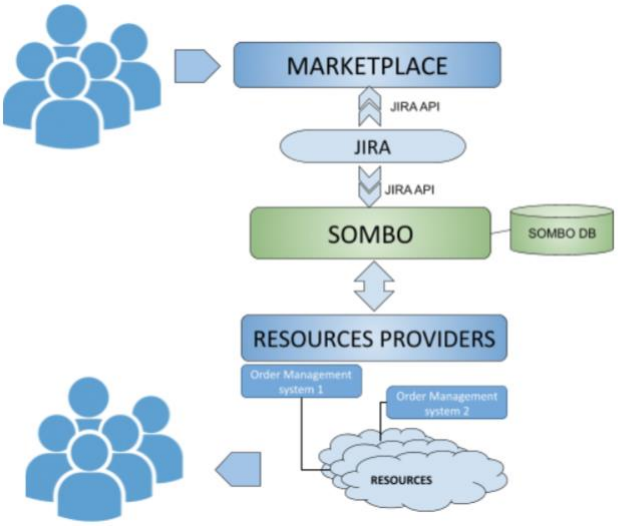


Figure 4 – EOSC marketplace Order Management System

Both the ordering and offering APIs are HTTP based and use JSON format for request and response. Authorization tokens are used to authorize both APIs. In the case of ordering API, a token is issued while OMS (order management system) registration is performed. Technical details on the APIs specification/description and API call examples are provided in Appendix B. EOSC Provider and Resource APIs and Appendix C: API calls to EOSC Catalog.

4. Data Management Infrastructure for Experimental Data

4.1. Experiment deployment and supporting infrastructure

SLICES will provide access to unique experimental facilities and testbeds operated by SLICES members and supported by infrastructure services to allow experiments management/control during the whole experiment lifecycle, including the support of the full cycle of the experimental data collection, processing, visualization and publication (and archiving). Experimental data produced in the SLICES experiments will become an important resource for researchers working on corresponding research topics.

[27] Service Order Management Backoffice (SOMBO) https://wiki.eoscfuture.eu/display/PUBLIC/Order+Management+Architecture+and+Interoperability+Guidelines, [Last accessed 26 August 2022].

[28] JIRA Adapter (https://github.com/cyfronet-fid/oms-adapter-jira) allows using the JIRA functionality for managing the process of order handling and negotiation between provider and consumer.

Good example of the experimental workflow support is provided by the Chameleon cloud platform[29] for Computer Science research which is a large-scale, deeply reconfigurable experimental platform built to support Computer Sciences systems research. Chameleon Infrastructure (CHI - Cloud++) is a cloud platform powered by OpenStack with bare metal (re-)configuration (Ironic), using OpenStack Blazar reservation service for experimental resources reservation.

Chameleon/CHI experimental workflow includes stages related to resource discovery, allocation, dynamic configuration, orchestration and monitoring. The workflow management services is supported by a rich library of orchestration templates and images created by the research community.

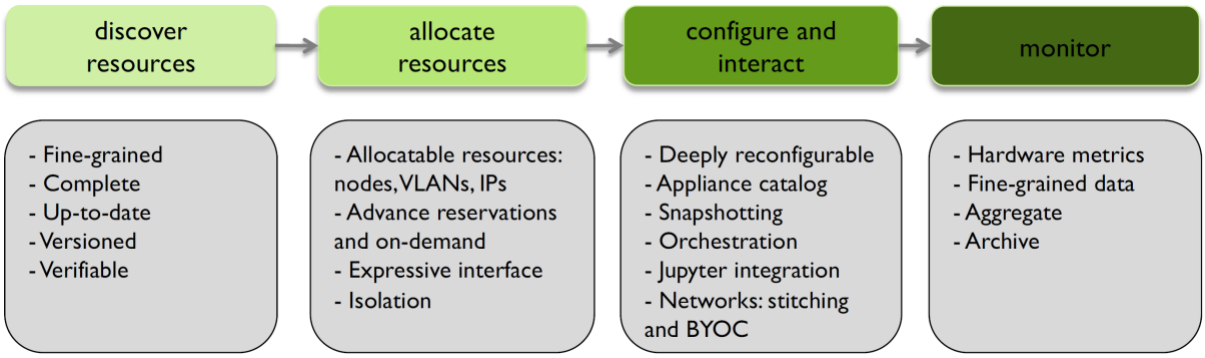


Figure 5 – Chameleon experimental infrastructure workflow[30]

Chameleon provides Jupyter integration for orchestration via JupyterLab service/portal[31] what allows creating and managing reproducible experiment workflows via Jupyter Notebooks[32] that can be created and shared by researchers (or research teams).

4.2. Experiment lifecycle and experiment workflow description

4.2.1. Github

Github is widely used for managing scientific code and data and in particular proprietary code for running experiments and processing experimental data. In this case, Github is used by the scientific programmers in the same way as software developers, also benefiting from powerful functionality for code sharing, integration and deployment (also referred to CI/CD process of Continuous Integration and Continuous Deployment).

However, using github for experiment automation is limited by code portability which depends on the individual scientific programmer style and may not provide sufficient code structuring and formalised interfaces to infrastructure computational and storage services.

[29] Chameleon cloud, https://www.chameleoncloud.org/about/chameleon/, [Last accessed 26 August 2022].
[30] https://www.chameleoncloud.org/media/filer_public/8d/a8/8da8b517-fd99-46ce-94ea-61d4edd94531/cluster.pdf, [Last accessed 26 August 2022].
[31] JupyterLab Documentation, https://jupyterlab.readthedocs.io/en/stable/index.html, [Last accessed 26 August 2022].
[32] https://www.chameleoncloud.org/blog/2019/10/25/reproducible-workflow-jupyter-chameleon/, [Last accessed 26 August 2022].

4.2.2. Jupyter Notebook

Jupyter Notebooks are widely used in the scientific community for scientific data analysis and reporting, however recent developments and uses are targeting the full scientific research cycle, including the full cycle of the experiment development and exploration.

Chameleon infrastructure provided a good example of using Jupyter Notebooks in defining and running experimental workflows, however in Chameleon, Jupyter Notebooks are used primarily for running experiments on already provisioned experimental infrastructure which is provisioned in the infrastructure provisioning workflow. Paper by M.Beg, et al[33] describes other cases of using Jupyter Notebooks for supporting reproducible experimental research.

SLICES-RI will leverage the Grid'5000[34] experience to support using Jupyter Notebooks for different aspects/activities in the experiment automation and experimental research data management[35]:

1. Notebooks as experiment drivers. These notebooks run the experiments from beginning to end, starting with resource reservations and going at least to data collection;
2. Notebooks as experimental payload. The code contained within these notebooks is the core of experiments. These notebooks run on the reserved resources, and either contain or control the computation that is the subject matter of the experiment;
3. Notebooks for post-processing. These notebooks are executed after an experiment to process the results. Supporting this usage will be dependent on your testbed's infrastructure and the type of post-processing expected;
4. Notebooks for exploratory programming. Notebooks for exploratory programming are used by users as a form of enhanced interactive shell in order to create new code through trial and error;
5. Notebooks as tutorials. Notebooks as tutorials are notebooks provided to the users by teachers that aim to present and explain to the users some specific concepts.

To achieve experiments reproducibility, the experimental platform must provide well-defined interfaces to experimental resources and data services that can be connected to the Jupyter programming environment. This is available in Chameleon scientific cloud and provided by the major public cloud and Big Data infrastructure providers such as AWS with their SageMaker Studio Notebook[36] and Microsoft Azure Data Studio Notebook service[37] that is also supported by the Azure DevOps for Data Science platform[38].

---

[33] Marijan Beg, Juliette Taka, Thomas Kluyver, Alexander Konovalov, Min Ragan-Kelley, Nicolas M. Thiéry, Hans Fangohr, Using Jupyter for reproducible scientific workflows [online] https://ieeexplore.ieee.org/document/9325550, https://arxiv.org/ftp/arxiv/papers/2102/2102.09562.pdf, [Last accessed 26 August 2022].

[34] Grid'5000 a computer science testbed based in France - https://www.grid5000.fr/w/Grid5000:Home, [Last accessed 26 August 2022].

[35] Luke Bertot, Lucas Nussbaum, Leveraging Notebooks on Testbeds: the Grid'5000 Case, Proc IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2021 [online] https://ieeexplore.ieee.org/document/9484501, [Last accessed 26 August 2022].

[36] AWS SageMaker Studio Notebook – https://docs.aws.amazon.com/sagemaker/latest/dg/notebooks.html, [Last accessed 26 August 2022].

[37] Azure Data Studio - https://azure.microsoft.com/en-gb/services/developer-tools/data-studio/, [Last accessed 26 August 2022].

[38] Team Data Science Process for data scientists, Article, 2022, https://docs.microsoft.com/en-us/azure/architecture/data-science-process/team-data-science-process-for-data-scientists, [Last accessed 26 August 2022].

4.2.3. CWL

To achieve experiment workflow portability (in addition to experiment reproducibility), the scientific community uses scientific workflow languages. Of many workflow languages introduced in the past, the Common Workflow Language is gaining popularity in recent times.

The Common Workflow Language (CWL) [39]is a specification to describe digital workflows. It describes how multiple steps in a computational workflow and their connections should be defined. CWL itself is only a specification, so a user needs a program to execute workflows called a runner. A reference implementation of such a runner exists called cwltool[40], however several workflow management systems also implement CWL support, for example, Apache Airflow[41], StreamFlow[42] and Toil[43]. Other workflow management systems offer partial or experimental support, such as Galaxy[44].

The standard defines CWL tools, which are described in '.cwl' files using a subset of YAML. These tools can execute command line tools, evaluate javascript expressions or define abstract operations to be implemented by a specific CWL runner. For each tool, the inputs and outputs need to be defined. It is also possible to list requirements such as required software, the ability to process inline javascript or certain files or directories to be present during a run time, among other requirements.

These steps can be combined into workflows, also defined in '.cwl' files using a subset of YAML. Each workflow contains a list of steps, with each step having defined inputs and outputs. These steps are not necessarily supposed to be executed in order but rather according to their dependencies on other steps. Independent steps can thus also be run in parallel. These steps execute CWL tools, either defined in the workflow itself or referencing a tool defined in a separate CWL file. The steps can also execute other CWL workflow, which allows for workflows to be nested. This works because for each workflow the inputs and outputs need to be defined. Finally, arbitrary metadata and metadata according to certain schemas can be defined in the workflows as well. The inputs of a workflow or tool are listed in a separate '.yml' file provided to the CWL runner at run time. A tool or workflow step can also be set to scatter, meaning it runs multiple times for each element of an array of inputs.

4.2.4. CWL for a sample experiment

The data processing step in the sample experiment was implemented using CWL, specifically using the reference implementation of a CWL runner. The full workflow takes as input the AWS access credentials and the name of the DynamoDB table containing the sensor data. After execution the workflow has produced the sensor data in CSV format sorted by date time and a description as well as a line chart of the sensor data. This is illustrated in the third step of the experiment diagram in Appendix D: Deployment templates for a sample experiment.

The first step of the processing runs a CWL tool which retrieves the MQTT sensor data from the DynamoDB table using the boto3 python module. This tool takes as input the AWS credentials to authenticate the client as well as the name of the table from which to retrieve the sensor data and outputs the sensor data in JSON format. The next step in the workflow converts the data from the

[39] https://www.commonwl.org/v1.2/, [Last accessed 26 August 2022].
[40] https://github.com/common-workflow-language/cwltool, [Last accessed 26 August 2022].
[41] https://github.com/Barski-lab/cwl-airflow, [Last accessed 26 August 2022].
[42] https://streamflow.di.unito.it/, [Last accessed 26 August 2022].
[43] https://github.com/DataBiosphere/toil, [Last accessed 26 August 2022].
[44] https://galaxyproject.org/, [Last accessed 26 August 2022].

JSON format to a CSV file using the JQ[45] command line tool. This CSV file is then sorted by the data time of the sensor measurements in the next workflow step. The sorted CSV is one of the outputs of the workflow as well as the input to a CWL tool that uses the python pandas[46] library to create a description of the data, including information such as the mean and standard deviation. This description is the second output of the workflow. Finally, the sorted CSV is also used as input to a CWL tool which uses the gnuplot[47] application to create a line chart of the sensor data.

The following code shows the contents of the CWL workflow used for the data processing step of the sample experiment with comments explaining the code. The tools used in each of the steps are defined in separate CWL files and referenced in the workflow code.

```
#!/usr/bin/env cwl-runner

cwlVersion: v1.0
class: Workflow

# The inputs of the workflow as a whole
# These are referenced in the first workflow step
inputs:
  AWS_ACCESS_KEY_ID: string
  AWS_SECRET_ACCESS_KEY: string
  table_name: string

# In the following list the workflow steps are defined
steps:
  # the first step, called "get_data" gets the sensor data from the DynamoDB table
  get_data:
    run: ../tools/get-dynamodb-data.cwl # the CWL tool is defined in this file
    # the following list defines the inputs to the CWL tool
    in:
      AWS_ACCESS_KEY_ID: AWS_ACCESS_KEY_ID
      AWS_SECRET_ACCESS_KEY: AWS_SECRET_ACCESS_KEY
      table_name: table_name
    # the output of this workflow step is defined as "dynamodb_data"
    out: [dynamodb_data]

  # the second step of the workflow converts the sensor data from JSON to CSV
  convert_to_csv:
    run: ../tools/json-to-csv.cwl
    in:
      # the input is the output of the previous step, "dynamodb_data"
      json_file: get_data/dynamodb_data
    out: [csv_file]

  # the third step sorts the sensor data in CSV format
  sort_csv:
    run: ../tools/sort.cwl
    in:
      file_to_sort: convert_to_csv/csv_file
      sort_field:
        default: 2 # which column to sort by
    out: [sorted_file]

  # the 4th step creates a description of the data
```

[45] Stedolan, https://stedolan.github.io/jq/, [Last accessed 26 August 2022].

[46] Pandas, https://pandas.pydata.org/, [Last accessed 26 August 2022].

[47] Gnuplot, http://www.gnuplot.info/, [Last accessed 26 August 2022].

```
  describe_data:
    run: ../tools/describe-csv.cwl
    in:
      # the input is the sorted CSV file from the previous step
      csv_file: sort_csv/sorted_file
    out: [data_description]

  # the 5th step generates a line plot
  generate_graph:
    run: ../tools/graph-csv.cwl
    in:
      # the input is also the sorted CSV file from the 3rd step
      csv_to_plot: sort_csv/sorted_file
    out: [plot]

# the outputs of the workflow as a whole are the sorted CSV file from the third
# step, the data description from the 4th step and the line chart from the 5th
# step
outputs:
  data_csv:
    type: File
    outputSource: sort_csv/sorted_file
  description:
    type: File
    outputSource: describe_data/data_description
  plot:
    type: File
    outputSource: generate_graph/plot
```

Appendix D: Deployment templates for a sample experiment provides details on the experiment deployment and execution deployment with Ansible playbooks and CloudFormation infrastructure component templates. The solution has been deployed and tested on the AWS cloud and proved that use of templates both for cloud resources and infrastructure and for experiment workflow provides effective instrument and platform for SLICES experiments automation for the whole experiment lifecycle.

4.3. Experimental Data Management Infrastructure and Components

4.3.1. Experimental Data Management stages

Management of experimental data is an important aspect of the SLICES-RI and it includes a number of services that must support all stages of the experimental data lifecycle, as illustrated in Figure 6 which also includes reference to SLICES Data Storage and Management Infrastructure and activities typical for experimental research such as Experiment planning and deployment (well explained in the previous sections), the discovery of data from internal data archives and external sources that are needed for correct experiment planning and setup as well as data publication and sharing. Also, another important aspect of data management such as data curation and quality assurance, must be supported by infrastructure tools. The figure also indicates that the SLICES Data Management infrastructure should be inter-connected with the EOSC Scientific Data Infrastructure for data sharing and access.

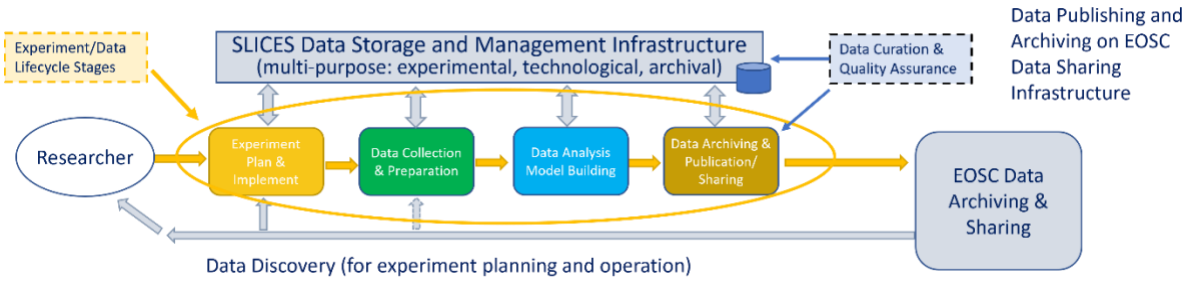


Figure 6 – SLICES Data Management stages and supporting infrastructure components.

Each Data Lifecycle stage: experiment setup, data collection, data analysis, and finally data archiving, - typically works with its own data set, which are linked and their transformation must be recorded in the process that is called lineage (that can also be extended to provenance for complex linked scientific data). All staged datasets need to be stored for the purpose and possibly reused in later processes. Many experiments and research require already existing datasets that will be available in the SLICES data repositories or can be obtained/discovered in EOSC data repositories

4.3.2. Infrastructure components to support the experimental data management

The following are requirements to robust data management infrastructure (DMI) for experimental data that follows from best practices and use cases analysis done in previous deliverables:

RDM1. Distributed data storage and experimental data(set) repositories should support common data and metadata interoperability standards, in particular common data and metadata formats. Outsourcing of data storage to the cloud must be protected with appropriate access control and compliant with the SLICES Data Management policies.
RDM2. SLICES DMI should support the whole data lifecycle. It should provide interfaces to research/experiment workflow and staging
RDM3. SLICES DMI shall provide PID (Persistent IDentifier) and FDO (FAIR Digital Object) registration and resolution services to support linked data and data discovery that should be integrated with EOSC services.
RDM4. SLICES DMI must support (trusted) data exchange and transfer protocols that allow policy-based access control to comply with the data protection regulations.
RDM5. SLICES DMI must enforce user and application access control and identity management policies adopted by SLICES community that can be potentially federated with the EOSC Federated AAI
RDM6. Procedures and policies must be implemented for data curation and quality assurance.
RDM7. Certification of data and metadata repositories should be considered at some maturity level following certification and maturity recommendations by RDA

The following infrastructure components, services and procedures must be implemented at different infrastructure levels to ensure interoperability and FAIR compliance:

A. Data Infrastructure

- Distributed data storage and repositories (for experimental data collection, processing and archiving or publishing), allowing cross SLICES DMI access
- PID/FDO infrastructure for data discovery and data linking
- Data Catalogs and Metadata Catalogs organized as central or distributed hierarchical and federated, that should allow/support data exchange with EOSC data infrastructure.

B. Data Management

- Data modeling that includes data model definition and deployment as distributed database or storage, in particular using standard SQL or NoSQL databases
- FAIR data principles and metadata registries (schemas) that are supported by corresponding infrastructure and user tools
- Metadata definition for sharing and publication of data produced by SLICES
- Metadata definition for the resource and experimental facilities description

C. Data Governance

- Organisational and legal aspects in data handling
- Policies and IPR, personal data protection
- Data Management Plan and definition of the Data Steward's organizational role and responsibilities (focused on maintaining DMP and data quality)

Strategy for practical SLICES DMI deployment must include well defined procedures for distributed data storage integration and linking to ensure data is discoverable/findable and accessible across all SLICES-RI. This should also relate to using external community and cloud-based storage and clear procedure should be developed for data migration. The technical solution for this can be adopting Data Spaces concept that is used in industry and promoted by International Data Spaces Association (IDSA) with their Reference Architecture Model[48] and recently also discussed at RDA19 BoF on Data Spaces Taxonomy[49].

SLICES will consider connecting to and using EOSC community services for coordinating hybrid data management infrastructure that may include both own data storage, as part of the private cloud, and external data storage offered by EOSC and EGI community. The use of public cloud storage and file sharing services will be considered and regulated by data management policies.

One of the candidate data management services OneData[50] offers scientific data spaces management solution and can be considered as a native solution among scientific community. Another assessed service is EGI DataHub[51] that allows enabling simple and scalable access to distributed data for computation, and to publish a dataset and make it available to a specific community, or worldwide, across federated sites, and it is actually based on the Onedata technology.

The main features offered by DataHub are:

- Data(set) sources and storage registration and discovery of data via a central portal.
- Access to data conforming to required policies which may be: unauthenticated open access; access after user registration or access restricted to members of a Virtual Organization (VO).
- Access to data via GUI, POSIX, CDMI
- Support for many backends (CEPH, S3, GlusterFS, POSIX, etc)

[48] IDSA Reference Architecture Model version 3.0, 2019 [online] https://internationaldataspaces.org//wp-content/uploads/IDS-Reference-Architecture-Model-3.0-2019.pdf [Last accessed 26 August 2022].

[49] RDA19 BoF on Data Spaces Taxonomy [online] - https://www.rd-alliance.org/research-data-spaces-taxonomy [Last accessed 26 August 2022].

[50] OneData - https://onedata.org/#/home [Last accessed 26 August 2022].

[51] EGI DataHub - https://www.egi.eu/service/datahub/ [Last accessed 26 August 2022].

- Metadata and shares management
- Data import and data caching based on file popularity
- Replication of data from data providers for resiliency and availability purposes. Replication may take place either on-demand or automatically.
- Authentication and Authorization Infrastructure (AAI) integration between the EGI DataHub and with other EGI components and with user communities existing infrastructure.

5. SLICES Data Management and Interoperability aspects

5.1. Adoption of Open Science and Open Access, FAIR data principles

Open science is a policy priority for the European Commission and the standard method of working under its research and innovation funding programmes as it improves the quality, efficiency and responsiveness of research. When there is active collaboration between academia, industry, public authorities and citizen groups in the research and innovation process, creativity and trust in science increases and also new problems/ideas emerge leading to further research and innovation actions. That is why the European Commission encourages or even requires that beneficiaries of research and innovation funding, make their publications available in open access and make their data as open as possible. Furthermore, it encourages linking with the European Open Science Cloud (EOSC) to enable not only storing, curating and sharing data but also facilitating access to other digital objects, such as tools, methods, software and services. However, to achieve these objectives, several challenges must be addressed, with perhaps one of the most important ones, the interoperability and reuse of data, as well as the careful consideration of issues such as Intellectual Property Rights (IPR) and licensing agreements.

Reusability of data is a key enabler for Open Science. Open data, which can be accessed without restrictions, may not necessarily be structured in such a way to serve research and innovation activities that require computational use of data. It is for that reason that the FAIR Principles were developed. However, FAIR is not equal to open and that is why some open data may not be FAIR. FAIR data must be geared towards human and machine readability and machine accessibility. Additionally, there are various constraints that limit the “openness” of FAIR data, such as privacy and security etc.

The need for open and fair data is also evident in the new Open Science policy governing all funding provided by Horizon Europe. All projects that collect, generate, process and re-use data, need to clearly define appropriate research data management and a detailed data management plan. The data needs to be open by default, unless specific exceptions apply. The data also need to be FAIR by providing persistent identifiers, using trusted repositories, machine-readable licenses and many more. The use of the European Open Science Cloud (EOSC) is compulsory in many work programmes.

There are different approaches/practices and platforms used to support research data management, focusing on features such as API functionality, query and retrieval mechanisms, and supported communities. The results of our investigation are detailed in D4.1, which summarizes the key requirements to maximize interoperability:

- FAIR principles: implementation should support FAIR principles;
- APIs: development of APIs (e.g., REST) for exposing metadata information to end-users;

- Repository Interoperability: support at least one metadata harvesting protocol (e.g., Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH)) to streamline data dissemination;
- Flexible Referencing: develop a flexible engine for transforming the implemented metadata format to the majority of well-known formats;
- Serialisation: support XML, JSON and YAML for data serialisation, while being extendible to support other formats in the future;
- Complex Querying: support both simple and advanced queries that allow users to use free-text search, combine filters and drill down to the individual components of each metadata property;
- User Experience: develop a user friendly, intuitive UI for designing and enhancing the user experience according to user experience design (UXD) principles. The UI must seamlessly offer the functionality of the underlying infrastructure in an efficient and pleasant manner.

We found that amongst the most predominant platforms that support research data management is Figshare and Zenodo, supporting the aforementioned features and more. An important characteristic of Zenodo is that it is hosted by CERN, which increases the probability for long term sustainability. A thorough comparison of these platforms can be found in[52], according to which the only additional feature that Zenodo supports is the ability to upload from Github, the predominant code hosting platform for version control and collaboration.

In conclusion, SLICES can design its own dedicated research data management platform, use an existing research data management platform, such as Zenodo or Figshare or combine both approaches (e.g., use a repository only for publications). The decision needs to take into account the cost of developing the platform and the final requirements from all stakeholders. If the decision is to implement a new platform, then the aforementioned objectives need to be fully met, so that the solution is more easily adopted by the community.

In the sections below, we provide specific requirements for the adoption of open science and FAIR data management and demonstrate how these were incorporated into the design of future SLICES-RI.

5.2. Adoption of the FAIR data principles

5.2.1. FAIR data principles and metadata management

FAIR (Findable, Accessible, Interoperable, and Reusable) Data Principles were developed[53] to ensure the longevity of data and to facilitate easier access to the wider research community to facilitate further knowledge discovery and research transparency. Moreover, with the rapid expansion of the digital ecosystem, the use of machines to process the vast volume of available data is crucial, as humans cannot efficiently and effectively perform the relevant data processing (e.g., find, access, reuse), without additional computational support. 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 machine-actionability, thus considering both human-driven and machine-driven data

[52] Mislan, K.A.S. & Heer, Jeffrey & White, Ethan. (2015). Elevating The Status of Code in Ecology. Trends in Ecology & Evolution. 31. 10.1016/j.tree.2015.11.006.

[53] 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 30 August 2022]

activities. The table below provides an overview of the four guiding principles and their main components:

Table 2 – FAIR Guiding Principles

Code	Principle
Findable	
F1	(meta)data are assigned a globally unique and persistent identifier
F2	data are described with rich metadata (defined by R1 below)
F3	metadata clearly and explicitly include the identifier of the data they describe
F4	(meta)data are registered or indexed in a searchable resource
Accessible	
A1	(meta)data are retrievable by their identifier using a standardized communications protocol
A1.1	the protocol is open, free and universally implementable
A1.2	the protocol allows for an authentication and authorization procedure, where necessary
A2	metadata are accessible, even when the data are no longer available
Interoperable	
I1	(meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation
I2	(meta)data use vocabularies that follow FAIR principles
I3	(meta)data include qualified references to other (meta)data
Reusable	
R1	meta(data) are richly described with a plurality of accurate and relevant attributes
R1.1	(meta)data are released with a clear and accessible data usage license
R1.2	(meta)data are associated with detailed provenance
R1.3	(meta)data meet domain-relevant community standards

To adhere to the above principles and to achieve efficient and effective sharing and reuse of data within and outside SLICES, appropriate metadata should be designed and developed. High-quality metadata are as important as data, and adequate resources should be allocated to cater for metadata operations. This means that appropriate protocols and procedures should be put in place on metadata operations (i.e., creation and operational workflows), ensuring proper management and stewardship. These protocols should also incorporate automated metadata production (e.g., metadata extraction) whenever possible and appropriate, decreasing both human effort and errors (e.g., use of predefined lists from established vocabularies), thus further contributing to enhancing the quality of data.

Metadata management should also be considered as an evolutionary process. Metadata attributes may be transformed, adapted or enhanced over time to cater to new or revised requirements. As such, metadata should be flexible and open, allowing for enrichment. Furthermore, metadata change management should be part of data governance's procedures utilizing appropriate mechanisms, such as versioning.

As such, several design principles should be considered to facilitate the management and sharing of the research data in an intuitive and safe way, respecting the creators' rights, while ensuring compliance with open access and FAIR principles. SLICES considers several design objectives for effective and efficient metadata management, which are detailed in D4.3 and are summarized below:

- Flexible Hierarchical Metadata Model design, which supports: (a) compulsory domain-agnostic attributes and can describe any digital object, ensuring compliance with FAIR principles; and (b) optional, domain-specific attributes to further enhance the description of the object for specific domains;
- Hybrid Metadata Production, which supports automated machine generated metadata to streamline metadata production and manual, human-based entries to personalize the information;
- Wide-ranging Interoperability, by incorporating attributes to support different levels of interoperability: *semantic*, which will allow internal and external systems to discover and understand what the underlying object is; legal, which describes the restrictions in the data; and technical, which will enable systems to communicate effectively using appropriate catalogues and services;
- Enhanced Discovery, by supporting both manual (e.g., keywords) and automatic descriptors (e.g., creation date, file size) but also using built-in data analysis functions (e.g., term document frequency) to allow for efficient discovery by external systems. Facilities for transforming to other metadata formats should be supported also;
- Long-term Reusability, to cater to enhanced or revised requirements for the metadat throughout the lifetime of the project. The metadata model must incorporate appropriate elements to describe the metadata model itself, including specific attributes, such as the metadata version, and utilize services where systems or users can obtain this info. Vocabularies and Standards utilized by specific attributes should also be versioned. This will also enable mappings between different versions of the same metadata records further enhancing reusability and interoperability;
- Metadata Governance, to ensure metadata quality, proper maintenance and audit.

5.2.2. Technical aspects in implementing FAIR data principles

Besides Data Management Plan and Data Governance Policy which are the main instruments to implement and maintain the FAIR principles, a number of dedicated infrastructure services must be implemented and available.

The following technical aspects need to be addressed when implementing FAIR data principles:

To ensure data Findability, the following services must be available:

- Metadata and PID – infrastructure and tools;
- Metadata registries, mapping, tools;
- PID and handles registries and resolution service;
- Source and usage policies, SLA management.

To ensure data Accessibility, the following services must be available:

- Repositories and data storage: infrastructure and management;
- Data protection, compliance, privacy and GDPR;
- Data access protocols;
- Data usage policies and access control: infrastructure and API management.

To ensure data Interoperability, the following services must be available:

- Standard formats for data, metadata and API;
- FAIR maturity level compliance and certification.

To ensure data Reusability, the following services must be available:

- Data provenance and lineage; data origin and experiment description to ensure reproducibility;
- Reliable and persistent data storage and databases, data backups.

5.3. Metadata description and registration

5.3.1. Results achieved in the Deliverable D4.3

SLICES wants to fully endorse and adopt the FAIR principles through appropriate metadata profiles that enable efficient and effective interoperability and cross-disciplinary research.

The D4.3 deliverable presented the initial proposal of the SLICES metadata profiles, primarily Provider and Resource profiles as defined by the EOSC Catalog. The report provided an overview of several important metadata standards and highlighted their purpose. Measures and solutions to comply with the FAIR guiding principles are discussed and implemented when defining the metadata profiles. The report also presented considerations for the transformability and management of the metadata that should address the task of integrating data from different experiments and tools.

SLICES proposes a flexible hierarchical metadata model consisting of three levels. The first level consists of compulsory domain-agnostic information that can describe any digital object (e.g., data, services), ensuring that it conforms to FAIR principles and beyond. Additionally, to enhance machine actionability for specific commonly used types of digital objects, such as data, services, and software, SLICES employs the second level of compulsory metadata attributes that are type specific. Finally, the third level incorporates optional domain-specific attributes to further enhance interoperability for specific communities. Where appropriate, SLICES will support additional optional metadata attributes accompanied by their metadata model to further enhance the description of the object. The model comprehensively describes data objects and also allows to be easily extended with new attributes or new types/categories as well as with new additional hierarchy levels.

5.3.2. MS10 progress and results

The SLICES metadata profiles have been specified and their conformance to the FAIR principles has been positively evaluated. Additionally, the preliminary EOSC provider digital object, which can be used to register SLICES RI on EOSC has been prepared and tested.

For the SLICES Provider profile, the following metadata are provided:

- Code – attribute code having prefix EPP.x (predefined by EOSC)
- Attribute Name
- Value
- Recommendation
- Definition

- Type
- Multiplicity
- Required
- Public
- Example

All metadata are grouped in the following blocks:

- Basic Information
- Contact Information
- Maturity Information
- Other Information

SLICES metadata templates are stored in the project document store.

5.3.3. SLICES Metadata Implementation

The preliminary design of the SLICES metadata profiles has been implemented with a focus on flexibility, modularity, and extensibility. The adopted approach allows to support future evolution of the metadata profiles as well as ensuring their portability across various platforms and deployment scenarios. The implementation consists of 3 key parts: the SLICES FAIR Digital Object (S-FDO) definition, conversion to and from other standards (currently supporting DublinCore and DataCite) and a test user interface to access and test the aforementioned functionality.

```
public sealed class SfdoResource
{
    #region Primary

    [SlicesCode(SlicesFieldCategory.PrimaryInformation, 1)]
    [SlicesParticipation(SlicesParticipationType.Required)]
    [SlicesAccessModifer(SlicesAccessModiferType.PU)]
    public SfdoIdentifier Identifier { get; set; }

    /// <summary>
    /// Internal Code created within SLICES which can be resolved by the SLICES portal.
    /// </summary>
    [SlicesCode(SlicesFieldCategory.PrimaryInformation, 2)]
    [SlicesParticipation(SlicesParticipationType.Required)]
    [SlicesAccessModifer(SlicesAccessModiferType.PU)]
    public SfdoIdentifier InternalIdentifier { get; set; }

    /// <summary>
    /// An alternate identifiers to the resource
    /// </summary>
    [SlicesCode(SlicesFieldCategory.PrimaryInformation, 3)]
    [SlicesParticipation(SlicesParticipationType.RequiredIfExists)]
    [SlicesAccessModifer(SlicesAccessModiferType.PU)]
    public List<SfdoIdentifier> AlternateIdentifiers { get; set; } = new();

    /// <summary>
    /// The creator(s) of the digital object in priority order. May be a corporate/institutional or personal name.
    /// </summary>
    //[SlicesCode(SlicesFieldCategory.PrimaryInformation, 4)]
    [SlicesParticipation(SlicesParticipationType.Required)]
    [SlicesAccessModifer(SlicesAccessModiferType.PR)]
    public List<SfdoCreator> Creators { get; set; } = new();
```

Figure 7 – SLICES Fair Digital Object (preliminary model)

The S-FDO is represented as a single class that contains all the metadata fields/attributes as illustrated in Figure 7. The model supports specific constraints, such as the nullable constraint, where a field can be marked as “absent” with an optional reason (e.g., the value could not be parsed during import). The fields are annotated with standardized metadata, which can be used for generation and programmatic consumption of meta-metadata of the SLICES metadata profiles to facilitate machine actionability. This

meta-metadata includes information, such as the SLICES code of each field, access options, required resource type for a specific field and others as illustrated in Figure 8.

```
#region Classification

/// <summary>
/// The branch of science, scientific discipline that is related to the digital object.
/// </summary>
[SlicesCode(SlicesFieldCategory.ClassificationInformation, 1)]
[SlicesParticipation(SlicesParticipationType.Required)]
[SlicesAccessModifer(SlicesAccessModiferType.PU)]
[SlicesAssociatedResourceTypes(
    SfdoResourceType.Publication,
    SfdoResourceType.Provider,
    SfdoResourceType.Data
)]
public SfdoOptional<List<string>> ScientificDomains { get; set; }
```

Figure 8 – Example of modifiers/annotations for each attribute in S-FDO

Conversion to and from other standards is facilitated via a dedicated module for each standard, providing flexibility for both development and deployment. Each module consists of 3 main components – serializer, importer and exporter. The serializer is responsible for converting between transferable representation (e.g., json, xml) and an in-memory representation. The importer is responsible for mapping the in-memory representation of the external standard to the SLICES model, while the exporter is responsible for the reverse process. Finally, a converter class acts as a wrapper to combine the 3 parts into a single interface for other code to consume.

The conversion system supports one or more serialized formats for each external (not SLICES) standard. As such, standards with multiple formats will likely require multiple serializers. For convenience, a default format is defined for most standards. Finally, the converter can be queried for supported formats (e.g., to present a choice for the user in a UI).

The users or machine interacts with the system via “frontends” – any code that’s able to accept user input. At the moment, CLI and desktop GUI frontends have been implemented for testing purposes, but this can be expanded to any form in the future, such as web applications, mobile applications, etc. Likewise, the location of the source and output of the conversion can be anything – in-memory strings, files on disk, etc. Additionally, the first 2 parts (S-FDO definition and converters) are versioned with the intention of being duplicated for each iteration of the SLICES standard. This allows multiple versions of SLICES to be used simultaneously (e.g., to convert a single record between them).

6. Data Management Plan (revision and update)

The final data management plan records the last actions performed for the data generated and managed throughout the project. It includes a breakdown of the different types of data collected, such as project related data, datasets and software, and details on how they are managed through the project. It also provides information about certified digital repositories that were utilized to store the project data, such as Zenodo for deliverables and GitHub for software code.

In collaboration with WP3 and within the context of this design study, the role of the data management director has been defined within the Coordination and Management Office (CMO) to manage all data-related aspects of the future SLICES-RI. The governance structure described in D3.4 positions the data management director as a thematic director directly below the CEO. D4.1 provides a description of the director's main responsibilities.

Intra- and inter-Interoperability issues and key design choices are also described in the DMP and detailed in D4.2, including integration with EOSC. The deliverable describes the key decisions made for the key semantic and technical building blocks such as Authentication and Authorization (AAI), persistent identifiers (PID) policy and application programming interfaces (APIs). These design choices are selected in such a way that they fulfill the interoperability requirements. Additionally, the data management plan provides a description of the implementation of the metadata profiles as these were described in D4.3. In particular, the description demonstrates how the SLICES FAIR Digital Object (S-FDO) was implemented, adhering to the proposed hierarchical design. The conversion to and from other standards (currently supporting DublinCore and DataCite) is also described.

The DMP also describes matters of protection of personal data and privacy rights, including a description of the technical and organisational measures that were implemented to safeguard the rights and freedoms of the data subjects/research participants. The deliverable, in collaboration with D7.1 also covers relevant security aspects in line with the measures that were implemented to prevent unauthorised access to personal data.

In conclusion, the final version of the DMP, covers all aspects required for effective data management for the SLICES-DS project and key decisions taken for the future SLCIES-RI project. The DMP considers recent development in FAIR principles, Open Science and data management best practices and details how the project effectively addresses data management responsibilities and resources, data collection and reuse, data quality assurance procedures, metadata models and management, storage considerations, legal and ethical requirements and data sharing, interoperability and long-term preservation.

7. Conclusion

The presented final deliverable summarises work done in the project to identify different aspects to ensure SLICES-RI interoperability with EOSC and other Research Infrastructures. Throughout the project execution, WP4 provided recommendations to WP2 on architecture issues related to SLICES internal and external interoperability and specifically interoperability with EOSC.

The report includes overview and assessment of the cloud native solutions and development platforms that are important to ensure interoperability and fast integration of the SLICES services and infrastructure with EOSC and external RIs. A hybrid cloud deployment model is suggested that allows using a private cloud to deploy SLICES infrastructure and occasionally using public clouds for development and solution testing, also for rapid experimentation

The deliverable analises DevOps and cloud integration tools to define and manage design patterns for interoperability and integration. Basic design patterns to support interoperability and integration have been identified and tested in the laboratory development environment. Code examples are provided in appendices and available on the WP4 development github.

To extend interoperability from infrastructure and data to the whole experimental research lifecycle, WP4 analysed general experiment workflow and corresponding infrastructure and services. This includes both experiment workflow description, data modeling, and experimental data management, starting from the data collection and storing to data processing, data lineage/provenance and data publication or archiving. Special attention is given to using experiment workflow definition and data description that allow for experiment reproducibility and portability. Options with github, Jupyter Notebooks and Common Workflow Language (CWL) were discussed.

The D4.5 Deliverable summarises WP4 developments on research data management services, procedures and policies, including metadata definition, management and publication/registration. SLICES Provider metadata profile was defined and registered with the EOSC Provider Catalog. As SLICES-RI develops and new services become available and mature, they will be registered with the EOSC Services and Resources Catalog.

The deliverable also reports on the updates made to the Data Management Plan (delivered in D4.1 in M6). This includes the adoption of the best practices in the DMP specification among European RIs and EOSC, better data stewardship functions definition for experimental data management and update on privacy and ethics.

In general, the WP4 delivered consistent view and technical recommendations on the interoperability and integration with EOSC and external RIs, and adoption of the FAIR data principles in the SLICES-RI. The WP4 activity was supported by the active involvement of the project partners in the activity of EOSC, and other related European and international initiatives to ensure the best use of existing services for the research community as well as smooth data sharing to increase the efficiency of research and address needs of the European research community.

8. Appendix A. The Platform Research Infrastructure as a Service (PRIaaS)

The Platform Research Infrastructure as a Service (PRIaaS) has been proposed in the authors' research paper as an architectural solution for future RI leveraging platform model to deliver specialized and community-oriented services.

A.1. Proposed PRIaaS Architecture Model

We propose the PRIaaS Architecture for FutureSDI as illustrated in Figure A.1. This model contains the three generalized layers:

- **Virtualised Resources (VR):** Virtualised general compute, storage and network resources that are composed to create infrastructure components and are used by other services and applications.
- **Actualisation Platform:** This is the main component and layer that enables provisioning, monitoring and operating fully functional instant Virtual RIs for specific scientific domains, projects, or communities.
- **Virtual (Private) RI (VirtRI):** Virtual RI provisioned on demand that contains a full set of services, resources and policies needed to serve the target scientific community and create full value change of data handling. VirtRI is operated by the specific community and uses services provided by the Actualisation platform, including the possibility of cross-platform data sharing.

Users and external resources include researchers, developers and operators, and external datasets.

Federation Access Infrastructure and Tenants Management FAI&TM) layer serves as interface layer enabling communication between distributed Actualisation Platform resources and services and generally distributed and multiorganizational VirtRI. FAI&TM is also the place where VirtRI and Actualisation Platform policy are enforced and managed.

A.2. Actualisation Platform Components

Actualisation Platform includes the following groups of services required to develop, deploy, manage and operate the Virtual RI during its whole lifecycle, including resources and users that can be grouped into Virtual Organisations.

- Core Infrastructure Services (IaaS & PaaS) including compute, storage, network, IoT&Edge, blockchain, Access Control and Federated Identity management, infrastructure security;
- Data Services including directory, metadata/PID, lineage/ provenance, FAIR &QA, semantic data lakes, data analytics and AI tools;
- Management and Operation including Service Catalog and Lifecycle Management, orchestration and management;
- Service provisioning and Fulfilment including user provisioning, SLA management and policy provisioning;
- Development Environment and Tools that support DevOps process related to platform and VirtRI development, provisioning and operation; this group also maintains the repository of API, containers and design templates that can facilitate VirtRI design and provisioning.

VirtRI provisioning process is based on well-known and commonly used DevOps tools and is supported by the Management and Operation functions. As the PRIaaS platform progresses, the repository of the

design patterns, templates and containerized applications and functions will grow. A starting point for such a repository can be the EOSC Catalog which already contains information about API for applications and services offered by existing RIs and service providers.

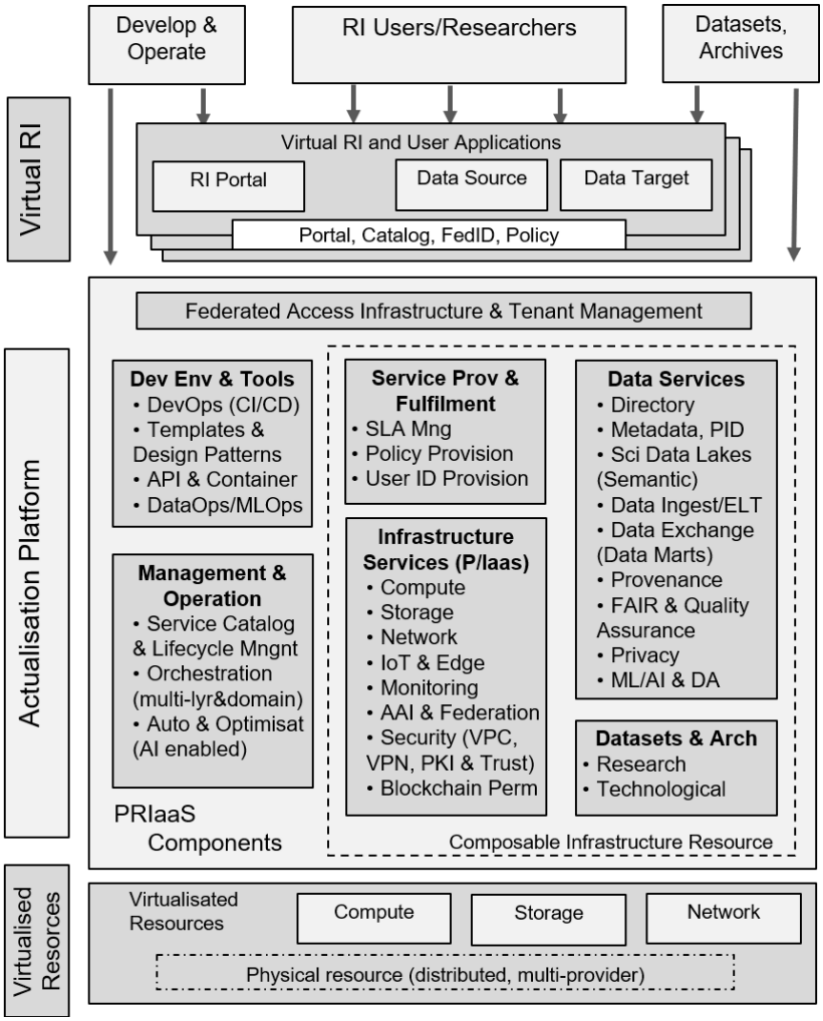


Figure A.1. The proposed PRIaaS architecture

The policy provisioning, management and enforcement are important functions of the Actualisation Platform that can be attributed to the Fulfilment function. The policy that is defined by the target community is provisioned as a part of VirtRI provisioning. Policy management and enforcement infrastructure should support policy roaming and combination for the multi-domain distributed resources and tenants.

9. Appendix B. EOSC Provider and Resource APIs

EOSC provider and Resource APIs (https://providers.eosc-portal.eu/openapi) are the external APIs that allow registration and discovery of provider and resource in the EOSC registry.

B.1. Provider Controller (External) APIs

API method	Function	Response
PUT /provider	Updates the Provider assigned the given id with the given Provider, keeping a version of revisions	200(OK) 201 (created) 401 (unauthorized) 403 (forbidden) 404 (not found)
GET /provider/all	#Filter a list of Providers based on a set of filters or get a list of all Providers in the Catalogue #	200/401/403/404
GET /provider/services/{id}	#Get a list of services offered by a Provider with the given id.#	200/401/403/404
GET /provider/{id}	#Returns the Provider with the given id#	200/401/403/404

B.2. Resource Controller (External) APIs

APIs	Function	Response
POST /resource	Create a new resource	200/201/401/403/404
PUT / resource	Updates the Resource assigned the given id with the given Resource, keeping a version of revisions	200/201/401/403/404
GET / resource /all	Filter a list of Resources based on a set of filters or get a list of all Resources in the Catalogue	200/401/403/404
GET /resource/by/{field}	Get all Resources in the catalogue organized by an attribute, e.g., get Resources organized in categories	200/401/403/404
POST /resource/validate	Validates the Resource without actually changing the repository	200/201/401/403/404
GET /resource/{id}	Get the most current version of a specific Resource, providing the Resource id	200/401/403/404
GET /resource/{id}/{version}	Get the specified version of a Resource, providing the Resource id and version	200/401/403/404

B.3. EOSC marketplace APIs

The EOSC marketplace API enables two functions: offering and ordering by resource/service providers and communities.

Both the ordering and offering APIs are HTTP based and uses JSON format. Authorization tokens are used to authorize both the APIs. In case of ordering API, token is issued while OMS (order management system) registration is performed. Following methods are used to access the API resources:

GET: to get the status of a resource

POST: To create a resource

Patch: To update the selected fields of a resource

DELETE: To remove a resource

B.3.1. EOSC marketplace offering API:

a. Resource APIS

1. /api/v1/resources /api/v1/resources # lists resources administered by user#
2. /api/v1/resources/{id} # retrieves an administered resource#

b. Offers APIS

3. GET /api/v1/resources/{resource_id}/offers # lists offers for an administered resource with a given resource ID#
4. POST /api/v1/resources/{resource_id}/offers # creates an offer for an administered resource#
5. GET /api/v1/resources/{resource_id}/offers/{id} # retrieves an offer for an administered resource#
6. PATCH /api/v1/resources/{resource_id}/offers/{id} #update an offer……#
7. /api/v1/resources/{resource_id}/offers/{id} # deletes an offer for an administered resource#

B.3.2. EOSC marketplace ordering API

Ordering API forms a message bus which enables tri-directional communication among the users, providers and EOSC portal operations team

Events

API method	Description	Required inputs	Response
GET /api/v1/oms/{oms_id}/events	List events	oms_id	200 (events found) 400 (bad request) 401 (user not recognized)

Messages

API method	Description	Required inputs	Response
GET /api/v1/oms/{oms_id}/messages	List messages	Project_id oms_id	200 (messages found)

			401 (user not recognized) 403(user not authorized to access this OMS) 404(project item not found)
POST /api/v1/oms/{oms_id}/messages	Create a message	oms_id	201 (message created) 401, 403, 404
PATCH /api/v1/oms/{oms_id}/messages/{m_id}	Update a message	oms_id message_id	200 (message updated) 400, 401, 403, 404(message not found)

Project Items

API method	Description	Required inputs	Response
GET /api/v1/oms/{oms_id}/projects	List projects	oms_id	200 (project found) 401 (user not recognized) 403 (user not authorized to access this OMS) 404 (OMS not found)
GET /api/v1/oms/{oms_id}/projects/{p_id}	Retrieve a project	Project_id oms_id	200 401 403 404

Order Management System (OMS)

API method	Description	Required inputs	Response
GET /api/v1/oms	List OMSs	NA	200 (OMSs found) 401 (user not recognized)
GET /api/v1/oms/{oms_id}	Retrieve an OMS	oms_id	200 401 403 404 (OMS not found)

10. Appendix C: API calls to EOSC Catalog

C.1. API calls to EOSC Catalog to get information on a Provider

This example shows how to retrieve the information about an existing provider from the EOSC Catalog. The provider is Mandat International and the API request is GET /provider/mi.

The corresponding cURL command is: curl -X GET --header 'Accept: application/json' 'https://api.eosc-portal.eu/provider/mi?catalogue_id=eosc'

The request URL is https://api.eosc-portal.eu/provider/mi?catalogue_id=eosc with the following header:


```
{
  "Accept": "application/json"
}
```


The response obtained is composed by a response code, response headers and a response body:

- Response code: 200
- Response headers:


```
{
  "cache-control": "no-cache, no-store, max-age=0, must-revalidate",
  "content-type": "application/json",
  "expires": "0",
  "pragma": "no-cache"
}
```


- Response body:


```
{
  "id": "mi",
  "abbreviation": "MI",
  "name": "Mandat International",
  "website": "https://mandint.org/",
  "legalEntity": true,
  "legalStatus": "provider_legal_status-non_governmental_organisation",
  "hostingLegalEntity": "provider_hosting_legal_entity-mandati",
  "description": "Mandat International is an independent and apolitical foundation with Special consultative status to the UN and other international organisations. The foundation aims at promoting international dialogue and cooperation. Throughout the years, the foundation has been working to facilitate the participation of delegates in international conferences and encourage international cooperation through various projects. The foundation also promotes international research projects with an emphasis on information technologies and sustainable development.",
  "logo": "https://mandint.org/wp-content/uploads/2021/07/mandint-logo_520x111.png",
  "multimedia": [],
  "scientificDomains": [
    {
      "scientificDomain": "scientific_domain-engineering_and_technology",
      "scientificSubdomain": "scientific_subdomain-engineering_and_technology-other_engineering_and_technology_sciences"
    }
  ],
  "tags": [
```

```
    "ICT, IoT, Internet, IPv6, SDGs, standardisation, GDPR"
  ],
  "structureTypes": [
    "provider_structure_type-single_sited"
  ],
  "location": {
    "streetNameAndNumber": "Avenue de Sécheron 15",
    "postalCode": "1202",
    "city": "Geneva",
    "region": "Geneva",
    "country": "CH"
  },
  "mainContact": null,
  "publicContacts": [
    {
      "firstName": null,
      "lastName": null,
      "email": "contact@mandint.org",
      "phone": "",
      "position": null
    }
  ],
  "lifeCycleStatus": "provider_life_cycle_status-operational",
  "certifications": [],
  "participatingCountries": [
    "CH"
  ],
  "affiliations": [],
  "networks": [],
  "catalogueId": "eosc",
  "esfriDomains": [
    "provider_esfri_domain-data_computing_and_digital_research_infrastructures"
  ],
  "esfriType": "provider_esfri_type-node",
  "merilScientificDomains": [],
  "areasOfActivity": [
    "provider_area_of_activity-applied_research"
  ],
  "societalGrandChallenges": [
    "provider_societal_grand_challenge-environment",
    "provider_societal_grand_challenge-energy",
    "provider_societal_grand_challenge-europe_in_a_changing_world"
  ],
  "nationalRoadmaps": [],
  "users": null
}
```


C.2. API call to publish information about Provider

The message format for publishing information on EOSC Catalog about Provider (PUT action)


```
{
  "id": "string",
  "name": "string",
  "abbreviation": "string",
  "website": "https://example.com",
  "legalEntity": false,
  "legalStatus": "string",
  "description": "string",
  "logo": "https://example.com",
  "multimedia": [
    null
```

```
  ],
  "scientificDomains": [
    {
      "scientificDomain": "string",
      "scientificSubdomain": "string"
    }
  ],
  "tags": [
    "string"
  ],
  "location": {
    "streetNameAndNumber": "string",
    "postalCode": "string",
    "city": "string",
    "region": "string",
    "country": "string"
  },
  "mainContact": {
    "firstName": "string",
    "lastName": "string",
    "email": "string",
    "phone": "string",
    "position": "string"
  },
  "publicContacts": [
    {
      "firstName": "string",
      "lastName": "string",
      "email": "string",
      "phone": "string",
      "position": "string"
    }
  ],
  "lifeCycleStatus": "string",
  "certifications": [
    "string"
  ],
  "hostingLegalEntity": "string",
  "participatingCountries": [
    "string"
  ],
  "affiliations": [
    "string"
  ],
  "networks": [
    "string"
  ],
  "structureTypes": [
    "string"
  ],
  "esfriDomains": [
    "string"
  ],
  "esfriType": "string",
  "merilScientificDomains": [
    {
      "merilScientificDomain": "string",
      "merilScientificSubdomain": "string"
    }
  ],
  "areasOfActivity": [
    "string"
  ],
  "societalGrandChallenges": [
```

```
    "string"
  ],
  "nationalRoadmaps": [
    "string"
  ],
  "users": [
    {
      "email": "string",
      "id": "string",
      "name": "string",
      "surname": "string"
    }
  ]
}
```

11. Appendix D: Deployment templates for a sample experiment

This Appendix provides example code for a simple experiment that includes experiment workflow written in CWL and corresponding CloudFormation template and Ansible playbook. They are available in the slicesdata project on github (https://github.com/slicesdata).

1) CWL workflow template (file MQTT_data_processing.cwl)

```
#!/usr/bin/env cwl-runner

cwlVersion: v1.0
class: Workflow

# The inputs of the workflow as a whole
# These are referenced in the first workflow step
inputs:
  AWS_ACCESS_KEY_ID: string
  AWS_SECRET_ACCESS_KEY: string
  table_name: string

# In the following list the workflow steps are defined
steps:
  # the first step, called "get_data" gets the sensor data from the DynamoDB table
  get_data:
    run: ../tools/get-dynamodb-data.cwl # the CWL tool is defined in this file
    # the following list defines the inputs to the CWL tool
    in:
      AWS_ACCESS_KEY_ID: AWS_ACCESS_KEY_ID
      AWS_SECRET_ACCESS_KEY: AWS_SECRET_ACCESS_KEY
      table_name: table_name
    # the output of this workflow step is defined as "dynamodb_data"
    out: [dynamodb_data]

  # the second step of the workflow converts the sensor data from JSON to CSV
  convert_to_csv:
    run: ../tools/json-to-csv.cwl
    in:
      # the input is the output of the previous step, "dynamodb_data"
      json_file: get_data/dynamodb_data
    out: [csv_file]

  # the third step sorts the sensor data in CSV format
  sort_csv:
    run: ../tools/sort.cwl
    in:
      file_to_sort: convert_to_csv/csv_file
      sort_field:
        default: 2 # which column to sort by
    out: [sorted_file]

  # the 4th step creates a description of the data
  describe_data:
    run: ../tools/describe-csv.cwl
    in:
      # the input is the sorted CSV file from the previous step
      csv_file: sort_csv/sorted_file
    out: [data_description]

  # the 5th step generates a line plot
  generate_graph:
```

```
    run: ../tools/graph-csv.cwl
    in:
      # the input is also the sorted CSV file from the 3rd step
      csv_to_plot: sort_csv/sorted_file
    out: [plot]

# the outputs of the workflow as a whole are the sorted CSV file from the third
# step, the data description from the 4th step and the line chart from the 5th
# step
outputs:
  data_csv:
    type: File
    outputSource: sort_csv/sorted_file
  description:
    type: File
    outputSource: describe_data/data_description
  plot:
    type: File
    outputSource: generate_graph/plot
```

2) CWL supporting files

All workflow files are organized in the following way

```
cwl/input/MQTT_data_processing_input.yml
cwl/tools/get-dynamodb-data.cwl
cwl/tools/json-to-csv.cwl
cwl/tools/sort.cwl
cwl/tools/describe-csv.cwl
cwl/tools/graph-csv.cwl
cwl/workflow/MQTT_data_processing.cwl
```

3) Experiment deployment on AWS cloud with Ansible and Cloud Formation

Functional and workflow diagrams below illustrate a simple experiment implementation and its three stages: Setup, Execution, and Cleanup.

Experiment deployment directory and template files are organized in the following way

```
deployment-templates/AWS CloudFormation/MQTT_template.yml
deployment-templates/ansible/MQTT-sample-playbooks/aws_ec2.yml
deployment-templates/ansible/MQTT-sample-playbooks/experiment-cleanup.yml
deployment-templates/ansible/MQTT-sample-playbooks/experiment-execution.yml
deployment-templates/ansible/MQTT-sample-playbooks/experiment-setup.yml
deployment-templates/ansible/MQTT-sample-playbooks/readme.md
deployment-templates/ansible/MQTT-sample-playbooks/templates/mosquitto_config.conf
deployment-templates/ansible/MQTT-sample-playbooks/templates/mqtt_pub.py.j2
deployment-templates/ansible/MQTT-sample-playbooks/templates/mqtt_sub.py.j2
```

The listing below provides an example of the Ansible playbook to run an experiment (playbook file experiment-execution.yml)

```
- name: Run the subscriber script
hosts: tag_Name_ansible_MQTT_subscriber
tasks:
- command: nohup python3 mqtt_sub.py &
args:
chdir: /home/ubuntu
async: 10000
poll: 0

- name: Run the publisher script
hosts: tag_Name_ansible_MQTT_publisher
tasks:
- command: nohup python3 mqtt_pub.py &
args:
chdir: /home/ubuntu
async: 10000
poll: 0
```

Figures below illustrate experiment deployment stages:

- Experiment setup
- Experiment execution and data processing workflow
- Experiment termination and clear down.

These stages and operations are described with separate Ansible playbooks that are available on the project github https://github.com/slicesdata

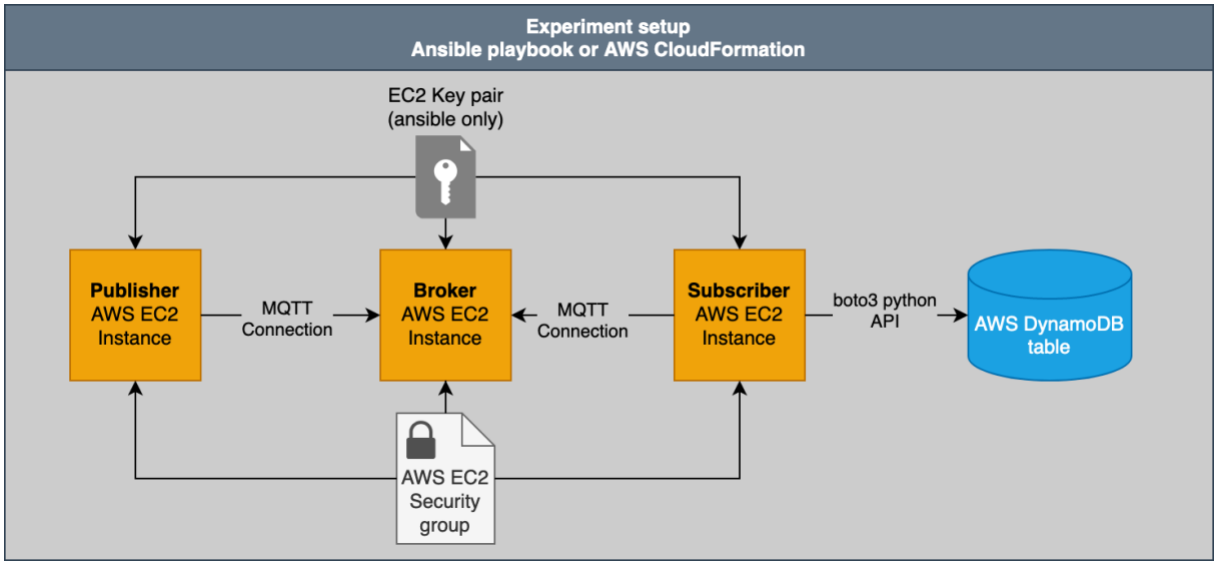


Figure D.1. Ansible playbook structure for Experiment setup.

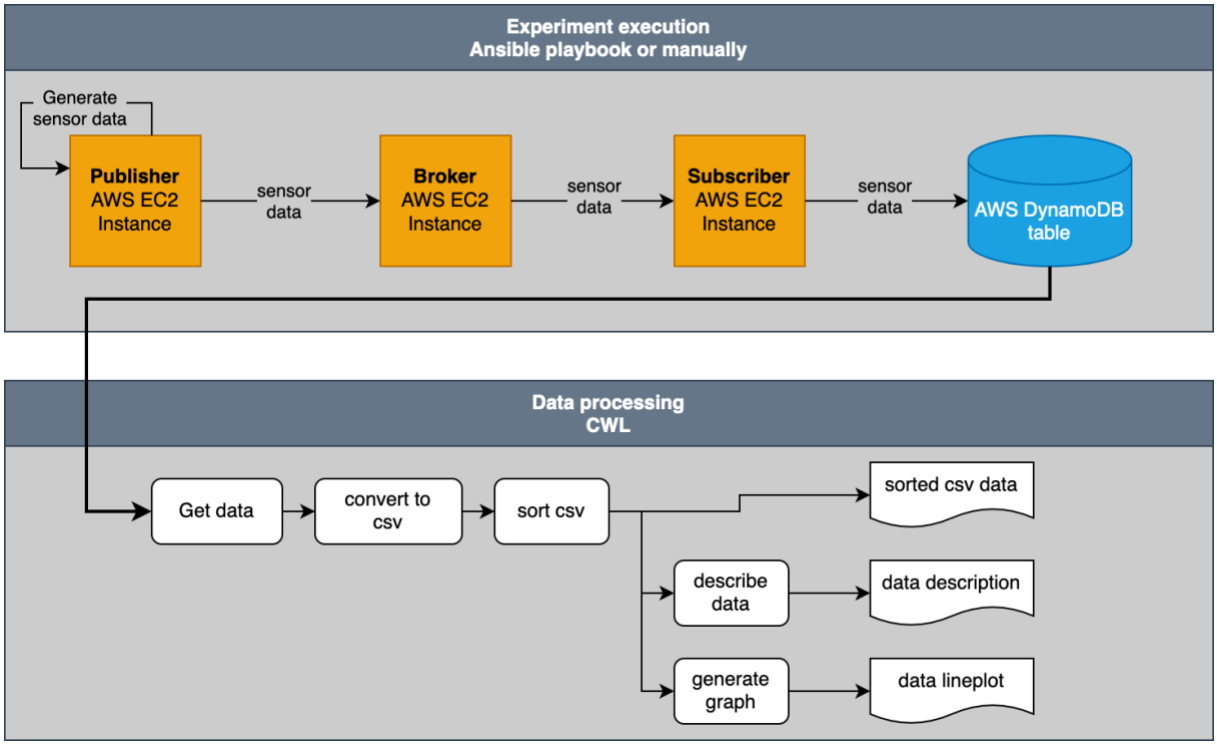


Figure D.2. Ansible playbook structure for Experiment execution and data processing workflow described with CWL.

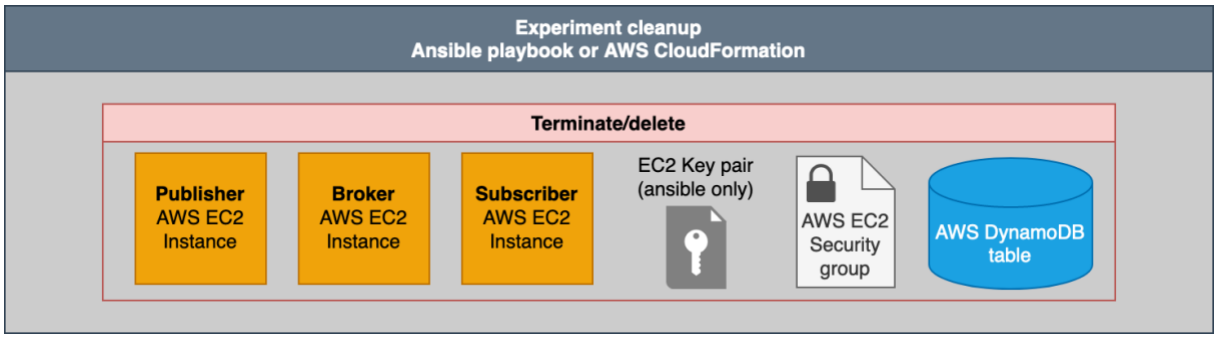


Figure D.3. Ansible playbook structure for Experiment termination and cleanup.

12. Appendix E. SLICES Alignment with the FAIR data principles (Excerpt from D4.1)

With the advancement of technology and the plethora of electronic data being generated and available online, there is a need to ensure the longevity of such data as well as its access to the wider research community. Wide access to scientific data can facilitate further knowledge discovery and research transparency. Moreover, with the rapid expansion of the digital ecosystem the use of machines to process the vast volume of data available is crucial, as humans cannot efficiently and effectively perform the relevant data processing (e.g., find, access, reuse), without additional computational support.

It is with the above in mind that the FAIR (Findable, Accessible, Interoperable, and Reusable) Data Principles were developed. The FAIR principles are intended 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.

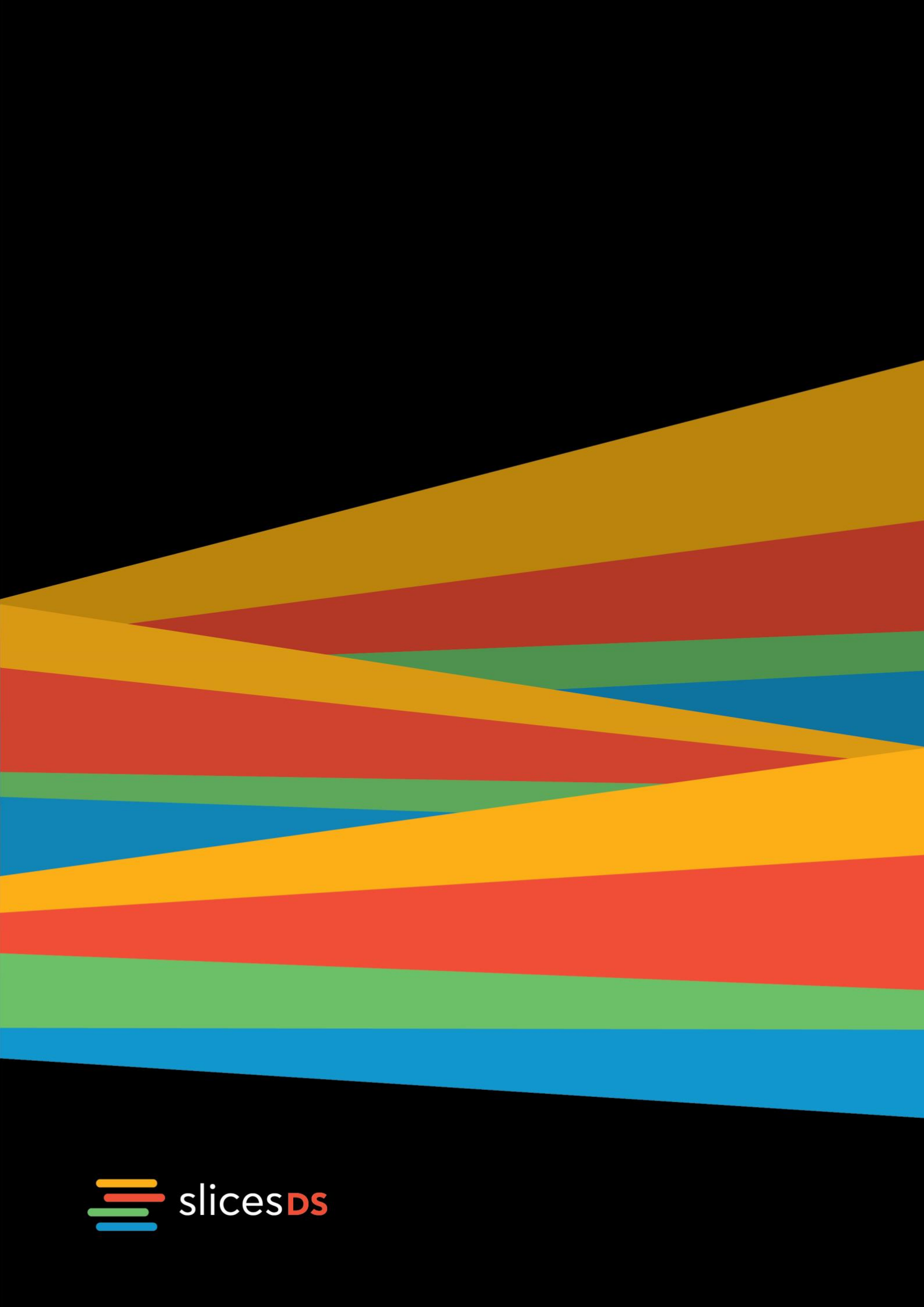
To this end, SLICES wants to fully endorse the FAIR principles and act as a catalyst to enable and foster the data-driven science and scientific data-sharing in this area by fully endorsing and adopting FAIR. In what follows, we provide how SLICES adheres to the FAIR guiding principles using the categorisation provided in [54].

Code	Principle	SLICES adherence to principle
Findable		
F1	(meta)data are assigned a globally unique and persistent identifier	The data will be assigned a Digital Object Identifier (DOI) at the time of upload.
F2	data are described with rich metadata (defined by R1 below)	Metadata will be provided in a consistent format with appropriate properties based on an enhanced version of the well-established format DublinCore. Furthermore, automatic metadata generation, e.g. using Machine Learning techniques, should also be used to provide even more reusability and scalability for interacting with other infrastructure and external digital libraries and their collections.
F3	metadata clearly and explicitly include the identifier of the data it describes	The data will be assigned a Digital Object Identifier (DOI) at the time of upload. The metadata will include the identifier in a dedicated Property.
F4	(meta)data are registered or indexed in a searchable resource	A dedicated resource discovery component will allow for searching data resources through the metadata and keywords (assigned by the user or automatically generated by the platform). The discover component will support simple and advanced queries that allow users to use free-text search, combine filters and drill down to the individual components of each metadata property.
Accessible		

[54] 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 26 August 2022].

A1	(meta)data are retrievable by their identifier using a standardised communications protocol	Metadata are retrievable using at least one metadata harvesting protocol (e.g., Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH)) to streamline data dissemination. Support for REST APIs with XML, JSON and YAML capabilities for metadata, as well as being extendible to support other formats in the future.
A1.1	the protocol is open, free and universally implementable	OAI-PMH and REST are open, free and universally implementable, as well as supported by the vast majority of applications.
A1.2	the protocol allows for an authentication and authorisation procedure, where necessary	Authentication and authorisation procedures will be implemented for data that are not openly available. Metadata will be open and will not require authentication and authorisation procedures.
A2	metadata are accessible, even when the data are no longer available	Metadata will be retained for the duration of the project and the lifetime of the infrastructure. In the case where a research data management platform is used, then the lifetime will be connected with the lifetime of the host repository. Metadata will be stored in a dedicated data store.
Interoperable		
I1	(meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation	Metadata will be provided in a consistent format with appropriate properties based on an enhanced version of the well-established format Dublin Core. Dublin Core has been formally standardised as ISO 15836, ANSI/NISO Z39.85 and IETF RFC 5013. The metadata will be accessible through the REST APIs, which will export XML, JSON and YAML) and also will have the ability to translate the metadata format to others based on a mapping engine.
I2	(meta)data use vocabularies that follow FAIR principles	The following vocabularies/standards will be utilised: •ISO3166: The set of codes for the representation of names of countries. •ISO639-3: The three-letter alphabetic codes for the representation of names of languages. •ISO 8601-1: Representations for information interchange for date and time •DCMI-Period: DCMI Period Encoding Scheme •DCMI-Point: DCMI Point Encoding Scheme We also anticipate more standards to be adopted when the final SLICES design is available.
I3	(meta)data include qualified references to other (meta)data	This information will be provided by the defined metadata property (“Relation”) and its sub-properties.
R1	meta(data) are richly described with a plurality of accurate and relevant attributes	Metadata will be provided in a consistent format with appropriate properties based on an enhanced version of the well-established format DublinCore. Furthermore, automatic keyword generation, e.g.

		using Machine Learning techniques, will be used to enhance discovery.
R1.1	(meta)data are released with a clear and accessible data usage license	Rights-License is part of the DublinCore properties. A list of licenses and their definitions will be provided to the user to select from when constructing their metadata. In the cases where a license does not exist, then Public Domain should be provided as a default option, but the option to select Other should also be available. Data downloaded by other users will be subject to the specified license.
R1.2	(meta)data are associated with detailed provenance	Provenance is part of the DublinCore properties and requires the user to provide a statement of any changes in ownership and custody of the resource since its creation that are significant for its authenticity, integrity and interpretation.
R1.3	(meta)data meet domain-relevant community standards	Dublin Core is one of the most widely used standards (ranks 1st in re3data) and has been formally standardised as ISO 15836, ANSI/NISO Z39.85 and IETF RFC 5013. It is cross-domain and not domain-specific.


slicesDS