Social Mining & Big Data Analytics

SoBigData

RESEARCH INFRASTRUCTURE ++

Deliverable D9.5

SoBigData e- Infrastructure Common Facilities 2

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871042

DOCUMENT INFORMATION

PROJECT	
PROJECT ACRONYM	SoBigData-PlusPlus
PROJECT TITLE	SoBigData++: European Integrated Infrastructure for Social Mining and Big Data Analytics
STARTING DATE	01/01/2020 (60 months)
ENDING DATE	31/12/2024
PROJECT WEBSITE	http://www.sobigdata.eu
TOPIC	INFRAIA-01-2018-2019 Integrating Activities for Advanced Communities
GRANT AGREEMENT N.	871042

DELIVERABLE INFORMATION	
WORK PACKAGE	WP9 JRA2 - E-Infrastructure and Supercomputing Network
WORK PACKAGE LEADER	CNR
WORK PACKAGE PARTICIPANTS	BSC, EGI, Nubisware, OpenAIRE, USFD, UNIPI, FRH, UT, LUH, AALTO, ETH Zürich, TUDelft
DELIVERABLE NUMBER	D9.5
DELIVERABLE TITLE	SoBigData e- Infrastructure Common Facilities 2
AUTHOR(S)	Massimiliano Assante (CNR), Alessia Bardi (OpenAIRE), Enol Fernandez (EGI), Ignacio Lamata Martinez (EGI), Marco Lettere (NUBISWARE), Andrea Manzi (EGI), Pasquale Pagano (CNR)
CONTRIBUTOR(S)	
EDITOR(S)	Massimiliano Assante (CNR), Ignacio Lamata Martinez (EGI), Valerio Grossi (CNR)
REVIEWER(S)	Jurek Leonhardt (LUH), Valerio Grossi (CNR), Ilaria Barsanti (CNR)
CONTRACTUAL DELIVERY DATE	31/12/2022
ACTUAL DELIVERY DATE	09/01/2023
VERSION	1.0
TYPE	Report
DISSEMINATION LEVEL	Public
TOTAL N. PAGES	39
KEYWORDS	Data Analytics, Online Coding, Science Monitoring, Continuous Integration

EXECUTIVE SUMMARY

Deliverable D9.5 “e-Infrastructure Common Facilities 2” is the revised version of Deliverable D9.4 “e-Infrastructure Common Facilities” intended to report the design principles and software architectures characterising the release and development of the SoBigData e-Infrastructure common facilities, namely the social mining computational engine, the online coding and workflow design frameworks, and the online science monitoring dashboard.

This revised version of the document covers the first 36 months of the project, including up to date information on the progress for the existing common facilities documented in D9.4 Deliverable at M12, and information on common facilities developed between M12 and M36. Specifically, (i) the Social Mining Analytics Engine section has been enriched with a new service for performing collaborative data processing and mining on information sets (Cloud Computing Platform) and (ii) the Online Coding and Workflow section has been enriched with the report on the integration of Galaxy open-source platform for FAIR data analysis.

The deliverable consists of six sections. Section 1 briefly introduces the role of this deliverable for the development and delivery of the SoBigData e-Infrastructure common facilities. Section 2 describes the SoBigData e-infrastructure logical architecture contextualising the common facilities and how they relate with the rest. Section 3, section 4 and section 5 document the first release of the e-Infrastructure common facilities included in this report and available at M36, reporting the design principles and reference architectures of the released solutions. Specifically, section 3 describes the social mining computational engine, Section 4 presents the online coding and workflow design frameworks - which includes the RStudio, the Jupyter Notebooks via JupyterHub, and Galaxy, Section 5 reports the online science monitoring dashboard.

Finally, section 6 concludes the report illustrating the whole Release Management process and its components for continuous integration.

DISCLAIMER

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871042.

SoBigData++ strives to deliver a distributed, Pan-European, multi-disciplinary research infrastructure for big social data analytics, coupled with the consolidation of a cross-disciplinary European research community, aimed at using social mining and big data to understand the complexity of our contemporary, globally-interconnected society. SoBigData++ is set to advance on such ambitious tasks thanks to SoBigData, the predecessor project that started this construction in 2015. Becoming an advanced community, SoBigData++ will strengthen its tools and services to empower researchers and innovators through a platform for the design and execution of large-scale social mining experiments.

This document contains information on SoBigData++ core activities, findings and outcomes and it may also contain contributions from distinguished experts who contribute as SoBigData++ Board members. Any reference to content in this document should clearly indicate the authors, source, organisation and publication date.

The content of this publication is the sole responsibility of the SoBigData++ Consortium and its experts, and it cannot be considered to reflect the views of the European Commission. The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated the creation and publication of this document hold any sort of responsibility that might occur as a result of using its content.

GLOSSARY

EU	European Union
EC	European Commission
H2020	Horizon 2020 EU Framework Programme for Research and Innovation
AAI	Authentication and Authorisation Infrastructure
API	Application Programming Interface
CCP	Cloud Computing Platform
CDN	Content Delivery Network
DM	DataMiner
ICT	Information and Communications Technology
OIDC	OpenID Connect
JWT	JSON Web Tokens
REST	Representational State Transfer
SAI	Software and Algorithms Importer
SMAE	Social Mining Analytics Engine
VRE	Virtual Research Environment
WP	Work Package
WPS	Web Processing Service
XML	Extensible Markup Language

TABLE OF CONTENTS

1 Relevance to SoBigData++

1.1 Purpose of this document

This document reports the releases of the social mining computational engine, the online coding and workflow design frameworks, and the online science monitoring dashboard available in the e-infrastructure at M36. It collects the design principles and reference architectures of the released solutions.

1.2 Relevance to project objectives

One of the main goals of the SoBigData++ project is to support cross-disciplinary research and innovation on the multiple aspects of social complexity from combined data-driven and model-driven perspectives, possibly implementing Open Science practices by means of exploiting an integrated platform where executions can be repeated, compared, discussed and logged. The common facilities listed in this deliverable, that have been developed and made available in the e-infrastructure during the first year of the project, facilitate and support the above-mentioned activities.

1.3 Relation to other work packages

The e-infrastructure common facilities pave the way for the creation of a platform where interdisciplinary tools, methods, and services can be contributed by Work Package 8 and Work Package 10, towards a view where these tools, methods, and services can be shared according to tailored policies, and easily combined.

1.4 Structure of the document

The remainder of the document is as follows: Section 2 briefly introduces the e-infrastructure logical architecture as a whole, describing how the common facilities relate to the rest. Sections 3, section 4 and section 5 reports on the e-infrastructure common facilities available to date. Specifically, section 3 describes the Social Mining Analytics Engine design principles and reference architecture. Section 4 illustrates the work performed during the period for what concerns the Online Coding and Workflow integrated, namely RStudio, JupyterHub and Galaxy. Section 5 reports on the Online Science Monitoring Dashboard design principles and architecture released at Month 36. Finally, Section 6 describes the Release Management procedure that has been put in place for the software continuous integration and delivery in the e-infrastructure.

2 The SoBigData e-infrastructure logical architecture

The primary objective of the SoBigData Infrastructure is not only to support social mining practitioners with seamless access to datasets and methods of interest, but rather to provide them with a platform for the design and execution of large-scale social mining experiments, accessible seamlessly on computational resources from the project cloud.

SoBigData++ WP9 is called to support the development of the SoBigData e-infrastructure in close collaboration with other work packages that are respectively called to (a) operate the infrastructure to provide virtual access to the integrated resources (WP7), (b) integrate existing and newly collected datasets and methods in the infrastructure (WP8), and (c) integrate existing tools and methods for mining social data in the infrastructure (WP10). Within this context, WP9 puts in place actions comprising: (i) studies and definition of best practices/policies for the harmonisation of federated resources available at the local infrastructure sites; (ii) support for adaptation of existing resources to the identified best practices; and (iii) realisation of VREs supporting scientists in benefitting from the integration of the federated resources and infrastructures.

The e-infrastructure common facilities provide the tools and the access to the computational resources that are necessary to enact such practices, by relying on the e-infrastructure service whose logical architecture is outlined in the following.

2.1 Logical architecture and common facilities

Figure 1 shows the e-infrastructure logical architecture, with the common facilities for the *Science Monitoring* and the *Online coding and workflow design* on top.

The e-infrastructure is built by exploiting the web-accessible virtual machines operated and provisioned by D4Science [4], together with services for their management and administration (federated resources). This layer also includes the enabling services that are required to support the operation of all services and VREs. Specifically, it includes a resource registry service, to which all e-infrastructure resources (data sources, services, computational nodes, etc.) can be dynamically (de)registered and discovered by users and other services; an authentication and authorisation service, as well as auditing services, capable of granting and tracking access and usage actions from users; a VRE management approach for deploying specialised VREs/VLabs, based on a selected subset of applications.

On top of this layer, two engines, namely the *Storage Engine* and the *Social Mining Analytics Engine* are built to deliver common facilities to the users. The former operates transparently to the users and virtualises the available storage technologies and their actual physical hardware. The latter, to date, is realised by exploiting and enhancing the gCube DataMiner engine operated by D4Science, in order to federate and integrate existing software frameworks provided within the SoBigData++ consortium members.

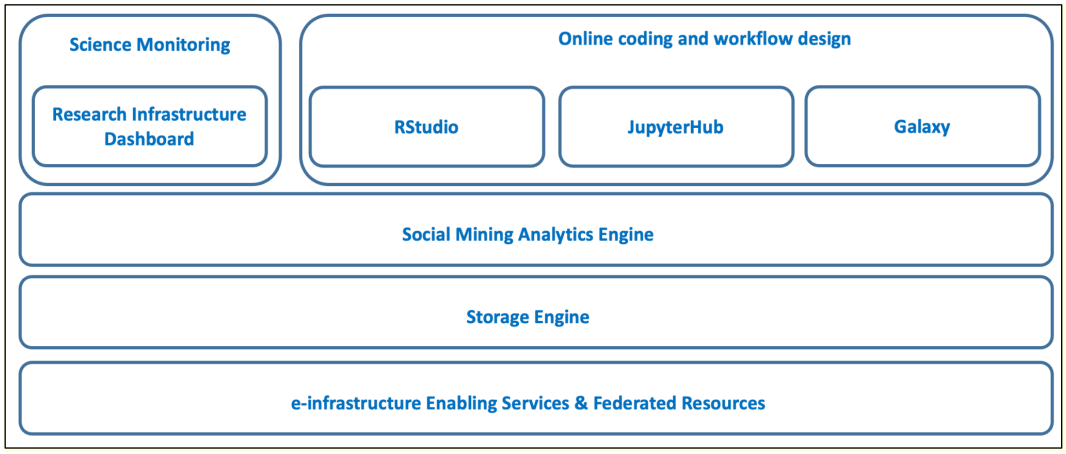

Figure 1: The e-infrastructure common facilities logical architecture

The *Online coding and workflow design* framework and the *Science Monitoring dashboard* sit on top of these facilities and exploit them to access or store data (Storage Engine) and to execute methods over this data (Social Mining Analytics Engine).

The *Online coding and workflow design* frameworks enable users to create live documents with code, text and visualisations that capture the whole research process: developing, documenting, and executing code, as well as communicating the results. The SoBigData coding common facilities available and integrated into the e-infrastructure to date are manifested in two of the three technologies depicted in Figure 1: *(i)* RStudio, for performing online coding with R language, aiming at performing statistical analyses, *(ii)* JupyterHub, to create and execute Jupyter notebooks [1] that are capable of seamlessly accessing the computational environment and resources available in the e-infrastructure, and *(iii)* Galaxy as a framework for data integration and scientific workflow design and execution.

The online *Science Monitoring* application manifests in the Research Infrastructure Dashboard (RID)[2]. RID monitors and quantifies the outputs of the e-infrastructure in the scholarly communication ecosystem. It also identifies every research product (publications, datasets, software, and other types) produced by the SoBigData infrastructure available in the OpenAIRE Research Graph[3]) which, as of December 2022, includes more than 170 millions research products linked to funding projects, organisations and authors.

[1] https://jupyter.org/
[2] The SoBigData++ Dashboard is available to the project coordinator at https://monitor.openaire.eu/dashboard/sobigdata
[3] https://graph.openaire.eu

3 Social Mining Analytics Engine design principles and architecture

The Social Mining Analytics Engine (SMAE) includes a set of services and components for performing data processing and mining on information sets. Users' analytical methods can be developed in almost any programming language (R, Java, Python, Fortran, Octave, etc.) and then imported via the Software and Algorithms Importer and executed via the DataMiner System. The Smart Executor System completes SMAE by adding support for the execution of scheduled and repeating tasks.

The Social Mining Analytics Engine provides support for:

- the execution of analytical methods on multi-core computational nodes;
- multi-tenancy and concurrent access;
- auditing;
- the execution of analytical methods on sets of computing nodes;
- scheduled and repeated execution;
- reproducibility and repeatability of the computed results thanks to the provenance information automatically generated by the engine;
- the standard Web Processing Service (WPS) protocol.

3.1 Software and Algorithms Importer and DataMiner systems

The Software and Algorithms Importer (SAI), depicted in Figure 2, is an interface allowing the registered user to easily import analytical methods into DataMiner which, in turn, publishes these methods *as-a-Service* and manages their execution in a multi-tenant and concurrent computational environment. Additionally, it allows scientists to update their analytical methods with just a few interactions with the interface and without following long software re-deploying procedures each time. In summary, SAI produces processes that run on the computing computational environment and are accessible via the WPS standard.

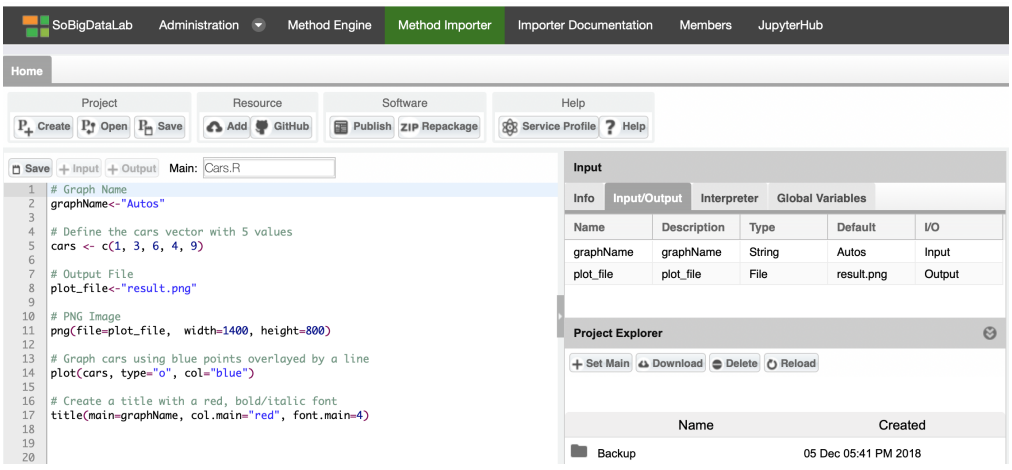

Figure 2: The Software Algorithms Importer (SAI) in the SoBigData Lab VRE

In order to import a script, three main steps are required:

1. **indicate Input, Output, and types** of the main script of the process;
2. **create the Software**: this operation packages the script and prepares it for execution in the computational environment It has to be used each time that there is a change in the interface (I/O) or in the required dependencies;
3. **publish the Software**: this operation enables the execution of the script on the computing platform in the context of the virtual laboratory where it has been imported.

Additionally, the Repackage function can be used to upgrade a published analytical method that has evolved in its code, without changing either the I/O or its dependencies.

Once imported, an analytical method becomes exploitable through the DataMiner component, which offers a Web GUI organised in three main areas: Data Space, Execution Space and Computations Space. This interface is shown in Figure 3.

The Data Space allows for accessing, reusing, and downloading the set of input and output datasets used and generated in one or more executions. Data Space items are enriched with business metadata reporting the computational method used for their generation, their execution environment (including the input parameters), the virtual laboratory where they were generated, and the date of generation.

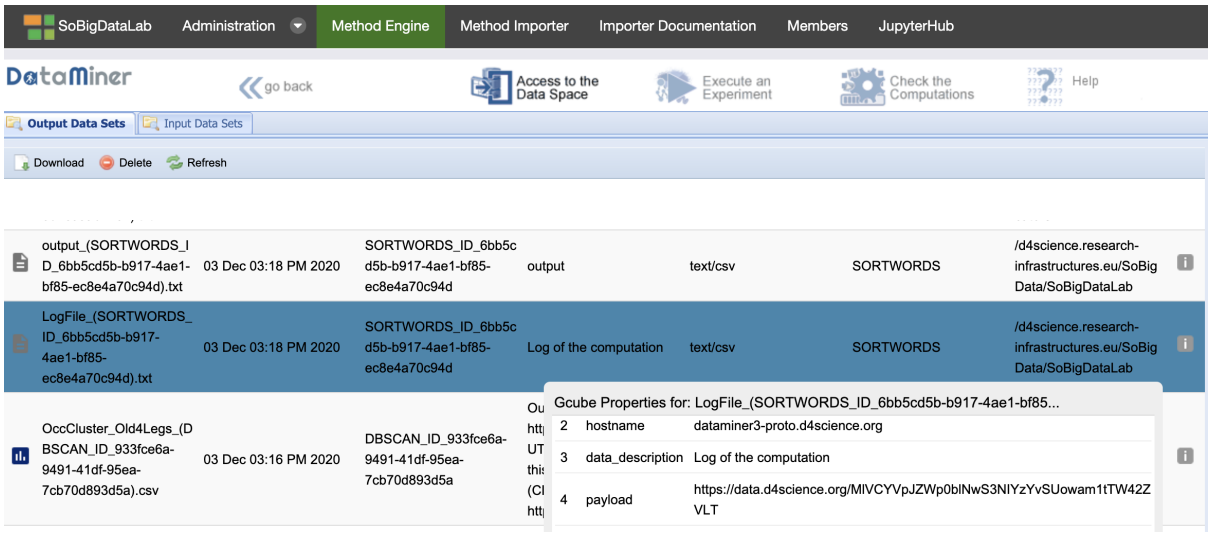

Figure 3: The DataMiner System in the SoBigData Lab VRE

The Execution Space, shown in Figure 4, presents two panels:

- The left panel presents the list of computational methods available in the virtual laboratory, which are semantically categorised (the category is indicated through SAI). For each method, the interface calls the WPS Describe Process operation to get the descriptions of the inputs and outputs.

– The right panel shows a form that allows users to specify the input parameters of the selected computational method. Input data can be clearly selected from the Workspace.

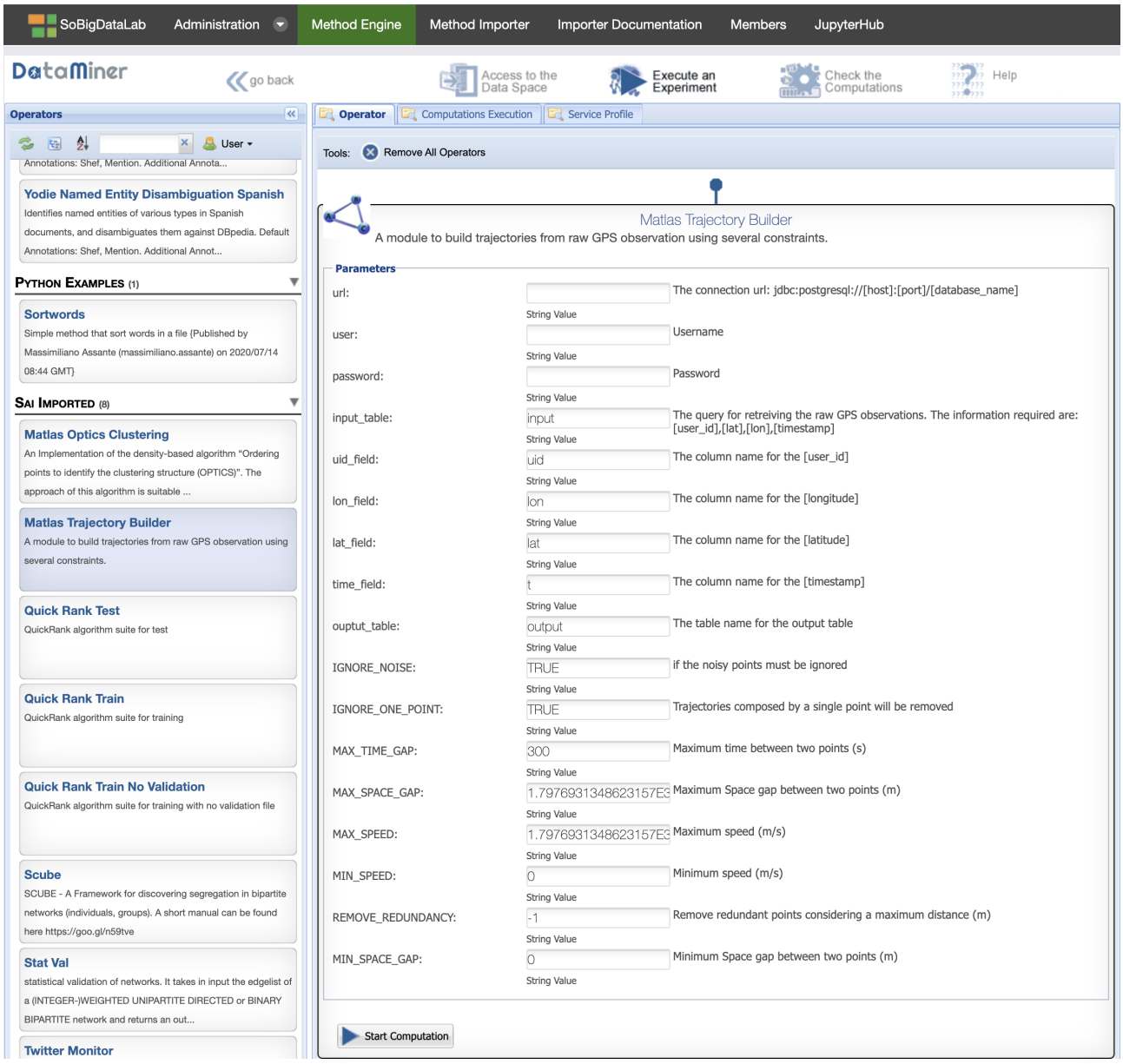

Figure 4: The M-Atlas Method interface in the DataMiner System

The Computations Space, presented in Figure 5, represents an important added-value since it reports a summary sheet of the provenance of the execution, either performed by the user or shared with him. From this same space, the computation can also be reproduced and repeated. In this last case, the Prov-O XML information associated with the computation is used to rebuild the computation request with the same parameters, and the execution can be re-submitted.

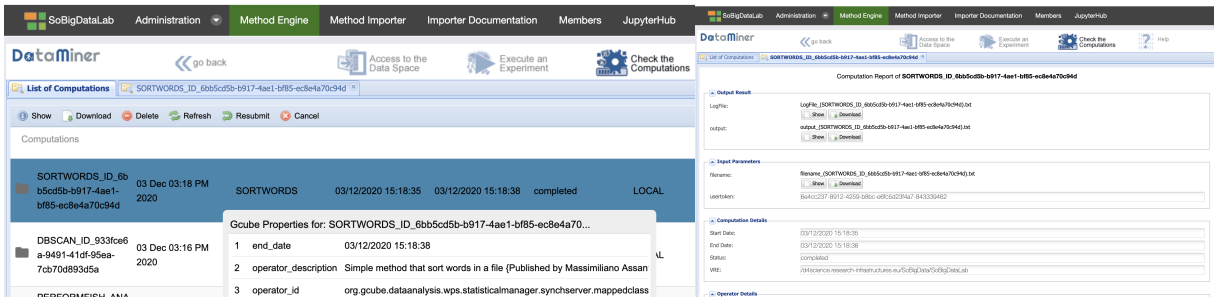

Figure 5: The DataMiner System computations space

3.1.1 Design principles and implementation

DataMiner (DM) is a service based on 52°North's WPS implementation[4]. DM is developed in Java as a Web Service running on Apache Tomcat and endowed with gCube libraries.

The complete DM architecture is made up of two sets of servers (clusters) that operate in a VRE acting as the Master Cluster and the Workers Cluster, respectively, as depicted in Figure 6. In a typical deployment scenario, these clusters are made up each of 16 servers managed by a load balancer (HaProxy), which distributes the requests uniformly to these servers. Each machine executes a DM service that communicates with Resource Registry to notify its presence and capabilities. The load balancer is indexed in the Resource Registry and is the main access point to interact with the different DMs. The Workers Cluster serves cloud computations.

The Master and the Workers clusters are dynamically provisioned by D4Science through an orchestration engine that uses Ansible scripts to configure multi-tier applications in a reliable, repeatable and consistent manner.

When a WPS request reaches the Master Cluster balancer, it is distributed to one of the cluster services (master DM). The different DMs host processes provided by several developers, specifically *local* and *cloud* algorithms.

Local algorithms run directly on the master DMs and possibly use parallel processing on several cores along with a large amount of memory. In contrast, cloud algorithms use distributed computing with a MapReduce approach and rely on the DMs of the Workers Cluster (cloud nodes).

With respect to the standard 52°North's WPS implementation, DM adds a number of features that enables collaboration between users and the repeatability and reusability of results. This is achieved by exploiting the common services available in the infrastructure, in particular:

[4] https://52north.org/software/software-projects/wps/

- DataMiner accepts inputs from workspace folders shared among several users, which enables the execution of challenging experiments by a group of people. This complements the execution of computations from files provided via HTTP links or embedded in WPS execution requests.
- The outputs of the computations are written to the distributed cloud storage and immediately returned to the client at the end of the computation. Afterwards, a separate thread makes this information accessible via the user's workspace. Indeed, after each successfully completed computation, a workspace folder is created containing the input, output, parameters of the computation, and a provenance document summarising this information. This folder can be shared with other users, who can re-execute the process again. Thus, the complete information about the execution can be shared and reused, fostering collaborative experimentation.

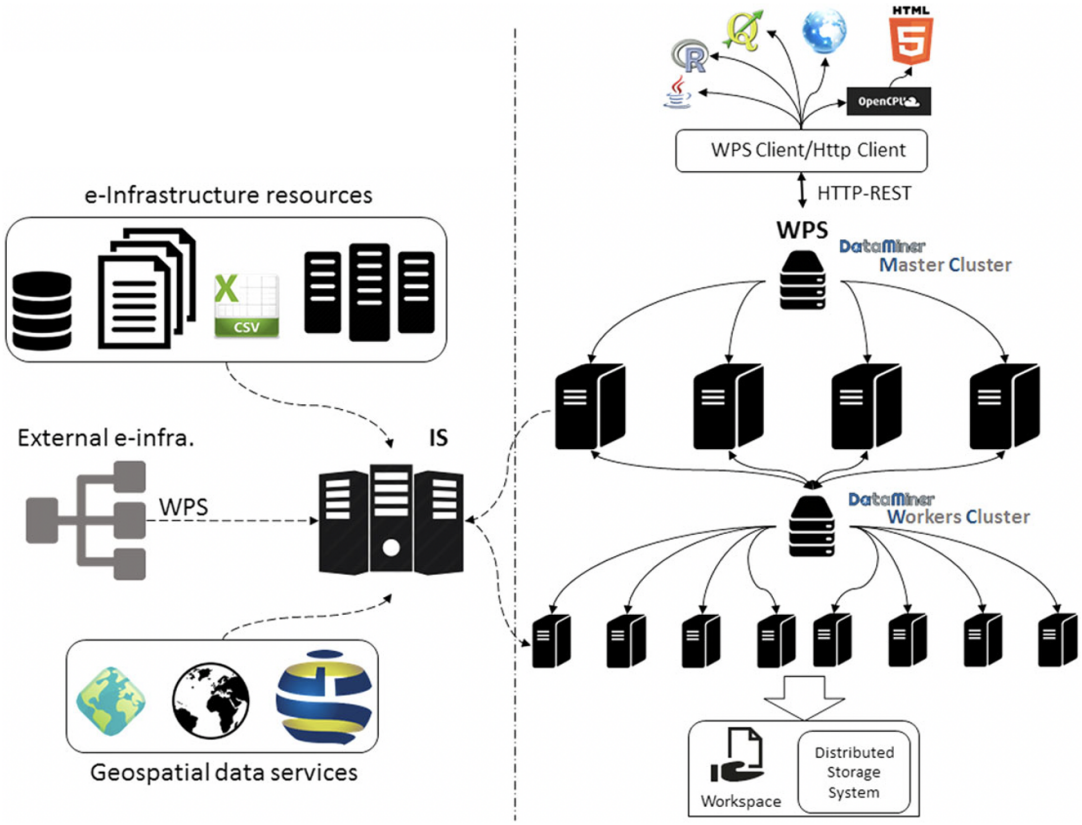

Figure 6: The DataMiner System architecture

The *Software and Algorithms Importer* and the *Data Miner* components are available in the following Git repositories:

- Software and Algorithms Importer (enabling the import of the analytical method): https://code-repo.d4science.org/gCubeSystem/statistical-algorithms-importer
- Data Miner (engine): https://code-repo.d4science.org/gCubeSystem/dataminer
- Data Miner Client (client for Java applications): https://code-repo.d4science.org/gCubeSystem/data-miner-manager-cl

- Data Miner Pool Manager (enabling the deployment of analytical methods): https://code-repo.d4science.org/gCubeSystem/dataminer-pool-manager
- Data Miner Manager User interface (the main user interface component): https://code-repo.d4science.org/gCubeSystem/data-miner-manager
- Data Miner Widget (a component of the main user interface): https://code-repo.d4science.org/gCubeSystem/data-miner-manager-widget
- Data Miner Executor (a packed user interface enabling just the execution of methods): https://code-repo.d4science.org/gCubeSystem/data-miner-executor

3.2 SmartExecutor

The SmartExecutor service, whose architecture is illustrated in Figure 7, allows users to execute tasks and monitor their status. Any task can be scheduled, repeated periodically, or activated upon request. A task has to be implemented as a plugin of the service, while its usage and exploitation can be performed using the SmartExecutor REST API.

The service enables the use of patterns to manage repetitive tasks that can be scheduled with a known frequency. It is typically used to periodically invoke a computational method imported into DataMiner. A common case is the monthly aggregation of raw data that is collected on a daily basis. In this example, the data aggregation is implemented via a computational method imported into DataMiner, whereas the monthly execution of the aggregation is performed through a task of the SmartExecutor service. The combination of the two services allows both single users and all users of a virtual laboratory to access the collection of aggregated data through the Workspace.

3.2.1 Design principles and implementation

The SmartExecutor service allows the execution and monitoring of *Tasks*. Each instance of the SmartExecutor service can run the Tasks related to the plugins available on that instance, for which they publish descriptive information about the co-deployed plugins in the Resource Registry.

Clients may interact with the SmartExecutor service through a library (the SmartExecutor Client) of high-level facilities to simplify the discovery of available plugins in those instances. Each client can request the execution of any Task or request information about the state of an existing execution. Moreover, the SmartExecutor service allows task execution through the use of co-deployed plugins, collecting input parameters needed for the execution of the requested plugin.

The execution is invoked every time it matches the scheduling parameters. The way to schedule the plugin execution is indicated by the scheduling parameter. There are two different ways to schedule an execution:

- **run and die**, in which the plugin is launched only a single time, without repetition.
- **scheduled**, in which the plugin is executed over time according to a specified delay interval or to a *cron* expression (a pattern that describes time frequencies).

SmartExecutor instances could take care of a scheduled execution when the node where it was previously allocated crashes or is overloaded. To achieve this goal, a scheduled task description is registered in the Resource Registry through the Resource Manager.

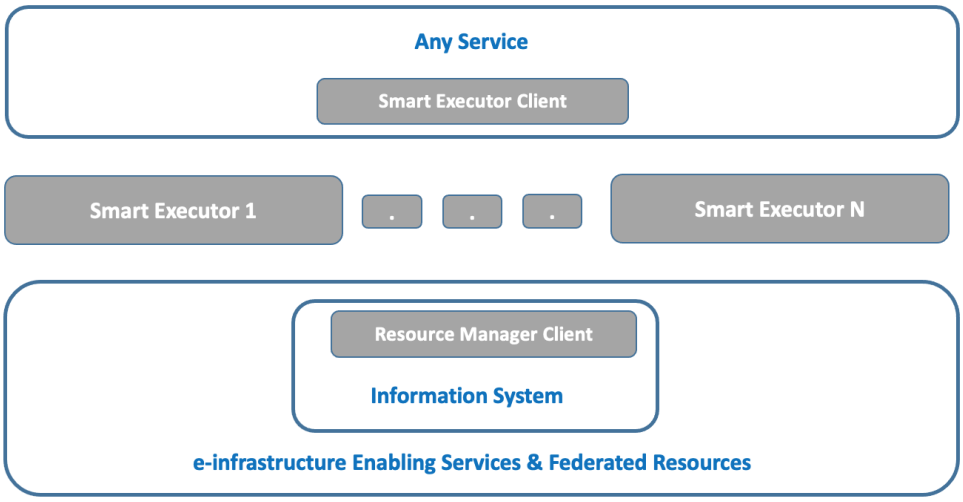

Figure 7: The Smart Executor System architecture

The Smart Executor components are available in the following Git repositories:

- Smart Executor service: https://code-repo.d4science.org/gCubeSystem/smart-executor
- Smart Executor Java client: https://code-repo.d4science.org/gCubeSystem/smart-executor-client

3.3 Cloud Computing Platform

The Cloud Computing Platform (CCP) represents an evolution of the current Social Mining Analytics Engine (SMAE). Several fields of ICT have experienced a major evolution during the last decade and many new advances, such as the widespread adoption of microservice development patterns. This resulted in substantial improvements in terms of interoperability and composability of software artefacts.

REST APIs have become a fundamental technology to build Web Services, while Webcomponents and Microfrontends aim at porting the advantages of service orientation downstream straight into the GUIs. New programming languages with smaller footprint, better focus on particular domains and simplified syntax have been designed, and the increased freedom that microservices grant has boosted their adoption.

As a result of microservice adoption and cloud-native approaches, new requirements and novel patterns have emerged also at operational level. The so-called “containerisation”, as a lightweight alternative to the virtualisation of computing resources, has become the de facto standard for packaging and provisioning software. Tools and languages have been created for the automation of provisioning, microservice orchestration and inspection and monitoring of software and infrastructure components.

3.3.1 Design principles and implementation

The Cloud Computing Platform (CCP) represents an evolution of the current Social Mining Analytics Engine (SMAE). Several fields of ICT have experienced a major evolution during the last decade and many new advances, such as the widespread adoption of microservice development patterns. This resulted in substantial improvements in terms of interoperability and composability of software artefacts.

REST APIs have become a fundamental technology to build Web Services, while Webcomponents and Microfrontends aim at porting the advantages of service orientation downstream straight into the GUIs. New programming languages with smaller footprint, better focus on particular domains and simplified syntax have been designed, and the increased freedom that microservices grant has boosted their adoption.

As a result of microservice adoption and cloud-native approaches, new requirements and novel patterns have emerged also at operational level. The so-called "containerisation", as a lightweight alternative to the virtualisation of computing resources, has become the *de facto* standard for packaging and provisioning software. Tools and languages have been created for the automation of provisioning, microservice orchestration and inspection and monitoring of software and infrastructure components.

1.1.1 Design principles and implementation

The vast landscape of new opportunities described above, in addition to the greatly increased requirements and expectations, have been the drivers for the design and development of a new Cloud Computing Platform that represents the result of a global rethink of the SMAE. After a critical overview of the current solution based on DataMiner and its Methods Importer (SAI), a use case driven analysis of the requirements for CCP has been performed, identifying the following ten use cases:

- **Use case 0 - Workspace abstractions:** CCP should be able to access file resources stored on different distributed workspace abstractions.
- **Use case 1 - Access Data as a Service:** CCP should be able to access data exposed as services.
- **Use case 2 - Low-level polyglot:** CCP should natively support as many programming and scripting languages as possible.
- **Use case 3 - High-level polyglot:** CCP should support different execution environments for scripts and applications.
- **Use case 4 - Support complex tools:** CCP should support complex tools that could have been used in pre-existing workflows.
- **Use case 5 - Support coarse grained parallelism:** CCP should support the possibility of subdividing scripts into parallel tasks.
- **Use case 6 - Support fine grained parallelism:** CCP should support the exploitation of parallel execution environments.
- **Use case 7 - Link to structured code repositories:** CCP should enable access to code stored on different code and artefact repositories.
- **Use case 8 - Develop with non headless workflows:** CCP should support environments with their own visual or interactive frontend.

- **Use case 9 - Discovery, consolidation and community driven growth:** It should be possible to discover consolidated data sources, methods and algorithms, both interactively and programmatically.

In addition to the use case analysis, a common architectural vision has been shared. The vision is logically represented in Figure 8.

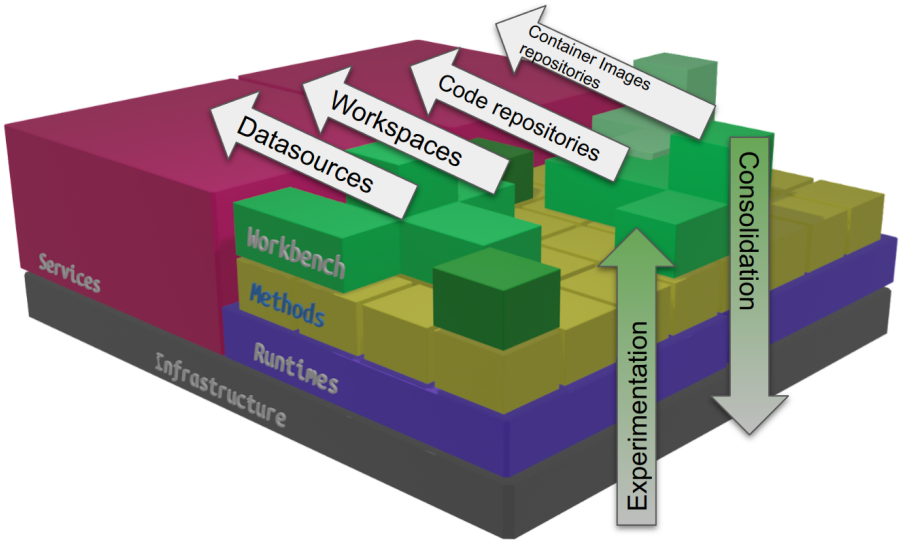

Figure 8: Common architectural Vision

In this vision, CCP is a layered set of components starting at the bottom with the **Infrastructure** layer, encompassing components such as hardware, Virtual Machines, container base clusters, storage facilities and networks.

The **Runtimes** layer offers a set of prebuilt, preconfigured execution environments such as container or Virtual Machine images. Platform cloud engineers are in charge of adding, adapting, or removing runtimes according to the requirements of the users, which express their needs in terms of compatibility with the methods they develop.

The **Method** layer contains specification of computational methods that can be anything, from social mining algorithms to AI classifiers and data harvesters. Data scientists with development skills are encouraged to develop new Methods or cloning existing ones, being their responsibility to choose compatible runtimes or propose new ones to be integrated. Tools for sharing the Methods with communities such as Virtual Research Environments are made available at this layer.

The overall user community works at the **Workbench** layer, which is the abstraction of overarching tools that are able to directly use the available Methods, compose them into workflows and integrate them into visual tools, such as Jupyter Notebook.

Typically, an e-infrastructure such as SoBigData offers a set of common facilities in the form of **Services**. These include, for instance, databases and data stores, workspaces, code repositories and repositories for container images. Methods executed in the CCP are able to exploit those services together with external ones.

Experimentation is the term that defines the activity of configuring new Runtimes, defining new Methods and using them in the Workbench.

In the opposite direction, **Consolidation** represents the possibility to transform dynamic objects into more static ones in order to improve reusability, portability and overall performance. For instance, workflows or combinations of Methods could be transformed into Methods themselves or even Methods into Runtimes.

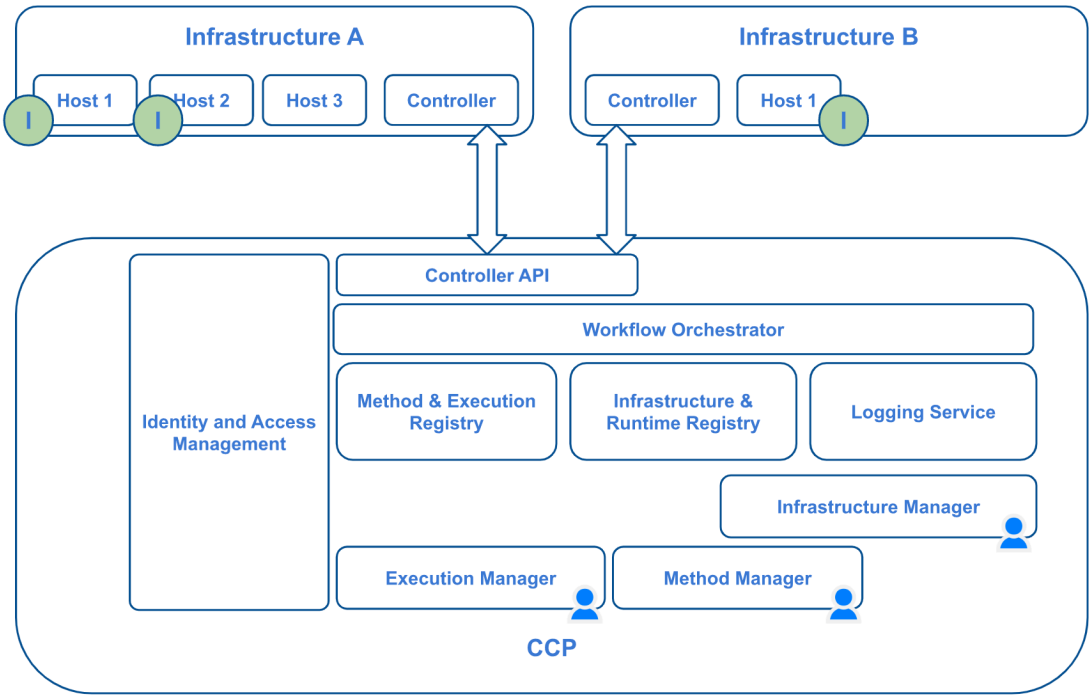

Figure 9: Logical architecture design

The logical architecture presented in Figure 9 shows the natively distributed nature of CCP. Starting from the top, **Infrastructures** (as computing resources that will host CCP executions, i.e. anything from simple laptops up to clusters of server *Hosts*) can be connected as runtime execution environments by installing a **Controller** component. Within an infrastructure, *Hosts* are computational nodes like servers delegated to run actual execution of methods, mostly in the form of running containers or Virtual Machines.
Controllers are processes that communicate through a specific API with the CCP in order to poll for *Tasks* to perform on the Infrastructure they control. *Tasks* may include building or deploying new Runtimes, reporting on the overall status of the Infrastructure and deploying and executing Methods.

In order to keep the current state for CCP, a couple of registries are involved. Specifically, the *Method & Execution* Registry and the *Infrastructure and Runtime* registry.

User driven visual components are available to manage Infrastructures, Runtimes, Methods and Executions at the frontend. Those components are identified by the user icon in Figure 9. Because many of the operations involved are lengthy and asynchronous, CCP includes a *Logging Service* that is used to send back realtime notifications to user components about the state of a particular process. These notifications include advancement of Executions, advertisement of Infrastructures status updates, and error conditions.

All complex processes involved in CCP are implemented as workflows inside a *Workflow Orchestrator* which, in addition to granting a high level of flexibility and customisation, allows for a centralised endpoint to monitor progress and check for errors that may occur. At the basis of all interactions among external actors, such as users and Controllers, a strong authentication and authorisation mechanism is enforced by *Identity and Access Management*. This enables it to address security requirements as well as to implement ownership attribution and auditing.

3.3.2 Release candidate version

At the time of writing, a release candidate version accessible since September 2022 is in operation for a selected audience of testers. CCP is packaged as a Docker based stack that can be deployed automatically. Figure 10 depicts the technological architecture of the current implementation.

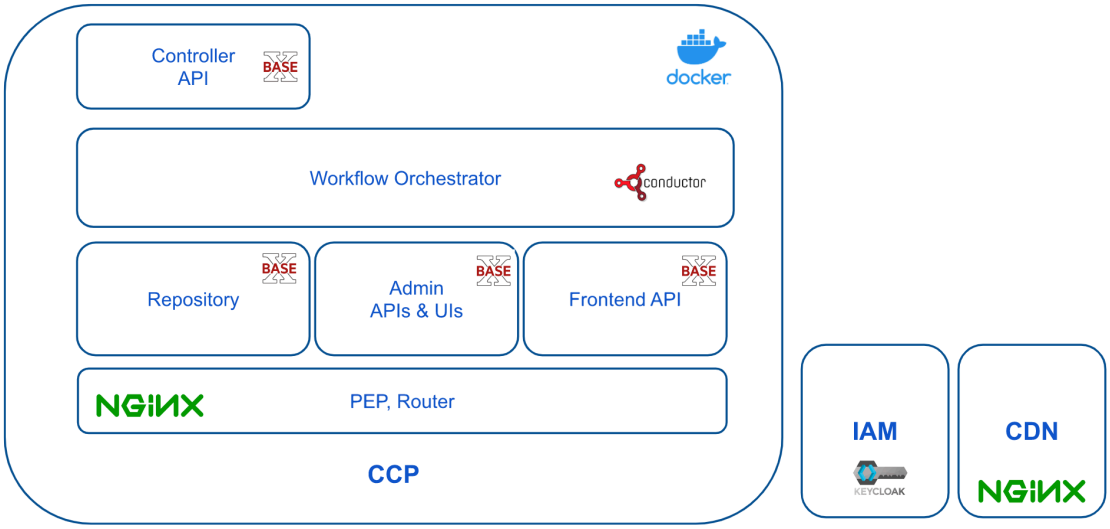

Figure 10: Technological architecture implementation

The functionality packaged internally to the stack comprises:

- An NGINX[5] proxy that acts as API Gateway, authorization Policy Enforcer Point (PEP).
- A set of REST APIs implemented with BaseX[6]:

[5] nginx: https://www.nginx.com/
[6] BaseX: https://basex.org/

  - A set of Admin APIs and UIs for accessing internal functionality and managing the databases and the workflow engine. These interfaces are only available to Admin users.
  - The Worker API, which is the API for infrastructure Controllers. It contains operations for advertising Infrastructures, reporting Infrastructure and Runtimes status and polling for Methods to execute. This API can only be accessed by the workers.
  - The Frontend API, which is the API that allows external applications to access CCP functionality, such as requesting a Method editing, creation or execution, monitoring Infrastructure status and retrieving Execution data. This API can be accessed by any users with *DataMiner* privileges in the corresponding CCP VRE.
- A Repository implemented in BaseX that stores data about Infrastructures, Runtimes, Methods and Executions, representing the state of CCP.
- A workflow engine based on Netflix OSS Conductor[7], which executes the CCP internal workflows.

The external components that are involved are:

- The SoBigData Identity and Access Management service implemented in Keycloak[8].
- The SoBigData CDN (Content Delivery Network) service provides HTML, CSS and JS-based Web components for accessing CCP functionality in a visual way.

Controllers are deployed outside the CCP stack "in front of" the Infrastructure they control. Communication towards CCP is performed via authenticated HTTPs calls, in order to grant access from any point of the Web. This makes it possible to integrate all types of infrastructure, including PCs.

In order to accomplish the Use Case 8 "Develop with non headless workflows", it was decided to follow a different approach with respect to SMAE for the production of visual interfaces. Instead of providing complete applications, the development team has written a set of Web components that can easily be integrated in external applications and frameworks, which made possible to develop a set of visual widgets that can be integrated in standalone Web pages, such as Wordpress or Web Portals, Jupyterhub and even mobile applications.

The Visual features currently developed include:

- **InfrastructureView -** For admin users shows what runtimes are available on what Infrastructures and allows for interactive provisioning;
- **MethodEditor** - Specify a Method according to the OGC Process specification[9] data model. Provide naming, keywords, inputs, outputs and provisioning information;
- **MethodList** - List of available methods. Searchable by name, description and keywords. Item is draggable to an ExecutionForm.

[7] Conductor: https://conductor.netflix.com/
[8] Keycloak: https://www.keycloak.org/
[9] OGC: OGC Process specification

Figure 11 shows the InfrastructureView, MethodEditor and MethodList visual interfaces.

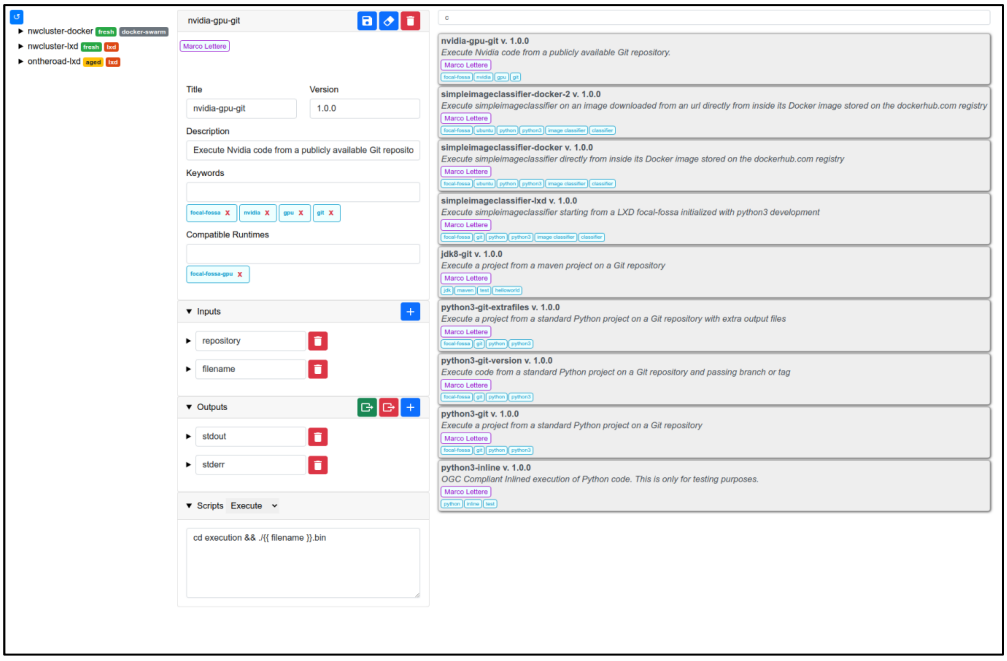

Figure 11: Standalone web page including InfrastructureView, MethodEditor and MethodList.

Figure 12 shows the **ExecutionForm** Visual, which provides the compilation of an OGC Process request according to the Method specification (Inputs can be specified with appropriate widgets and outputs can be selected). The **ExecutionMonitor**, depicted in Figure 13, is used to visually get real time updates about execution requests, including access to all meta information (Infrastructure and runtime descriptors, method descriptors in OGC format, Request inputs and output files).

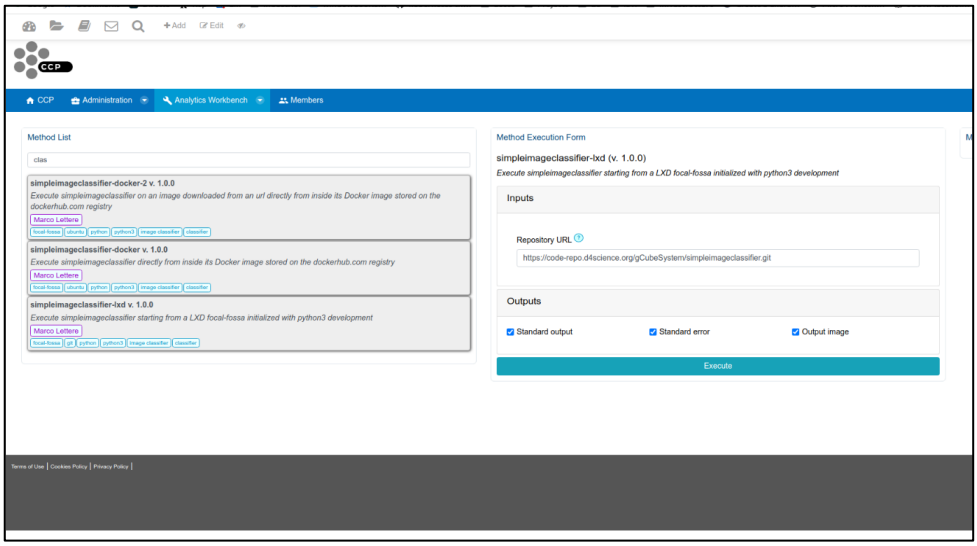

Figure 12: MethodList and ExecutionForm displayed inside a D4Science Gateway built with Liferay.

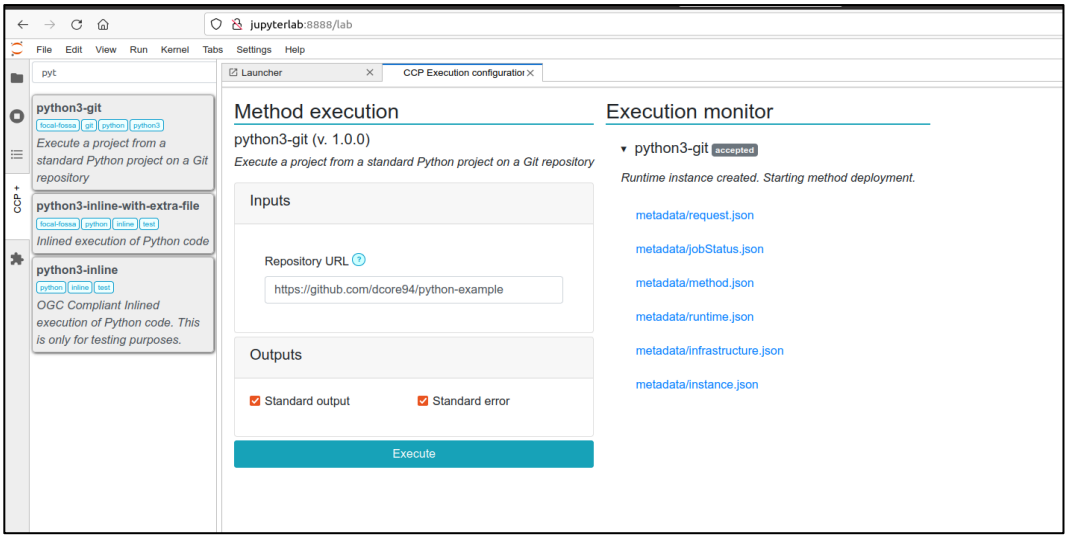

Figure 13: MethodList, ExecutionForm and ExecutionMonitor integrated in JupyterHub.

4 Online Coding and Workflow design principles and architecture

The online coding and workflow system enables users to create live documents with code, text and visualisations that capture the whole research process: development, documentation, code execution, and results communication. SoBigData coding common facilities available and integrated into the e-infrastructure are represented by three applications (see Figure 1): (i) **RStudio**, for performing online coding with R language to conduct statistical analyses, (ii) **JupyterHub**, which allows executing Jupyter notebooks providing users with access to computational environments and resources of the e-infrastructure, and (iii) **Galaxy**, which is an open-source platform to enable the use of tools as part of a workflow.

The design and logical architecture for each of these applications is presented in the following Sections 4.1, 4.2 and 4.3.

4.1 RStudio

RStudio allows performing online statistical analyses with R (a programming language for statistical computing and graphics). RStudio is provided on SoBigData e-infrastructure, ensuring its operation and orchestration in a cluster. The cluster is composed of multiple hosts, each of which is assigned exclusively to a single user during an online session. At the end of the session, all the content stored in that host may be removed by the e-Infrastructure, so that the host is available for a new user. The user can persist the R-session into their private Workspace (using the Cloud storage via the Storage Service) that is accessible through the RStudio Application. The RStudio interface is shown in Figure 14.

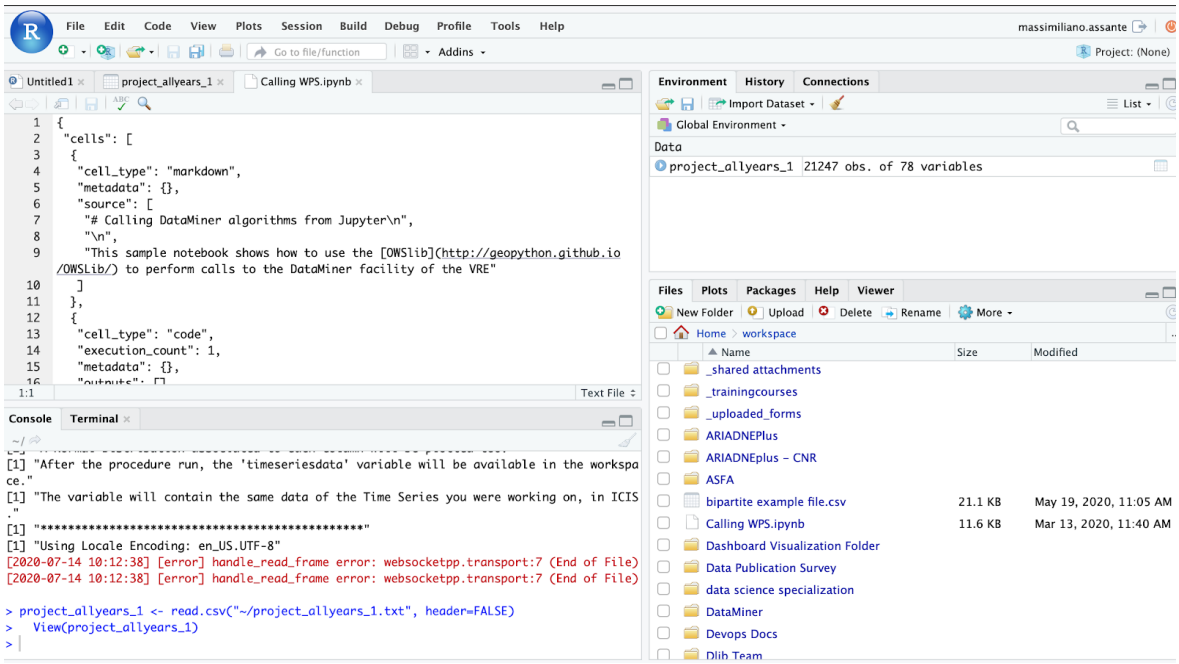

Figure 14: VRE The RStudio Interface with private Workspace mounted transparently

4.1.1 Design principles and implementation

The design and implementation reference architecture of the RStudio coding solution facility available in the e-infrastructure is presented in Figure 15. RStudio applications are deployed on a cluster of virtual servers and shared equally among users. Users starting new RStudio sessions (from the e-infrastructure Virtual Lab) are assigned automatically to one of the instances.

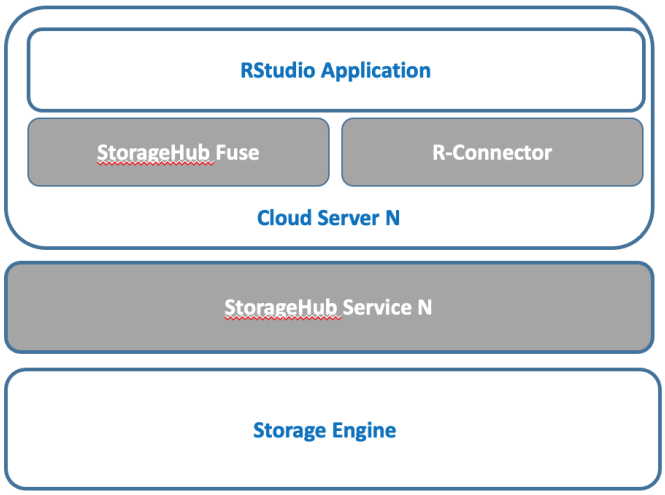

Figure 15: The RStudio solution reference architecture

As mentioned before, users can persist R-sessions in their private Workspace that is accessible through the RStudio Application. This feature is enabled by the **StorageHub Fuse** component, a custom component developed to work with the Linux Filesystem in Userspace (FUSE)[10] software interface. This connects the Linux filesystem and the SoBigData workspace interface, known as the StorageHub service. The **StorageHub** Service is replicable and a proxy (HaProxy) on top is used for proxying requests to the deployed instances of it. StorageHub relies on two different storage technologies to store item metadata: Apache Jackrabbit[11], as a metadata repository, and PostgreSQL[12], as the back-end database for Apache Jackrabbit. One other distinguishing feature of this service is that the actual payload of the items can be stored on a number of in-house and commercial storage technologies that are provided by the e-infrastructure Storage Engine, such as MongoDB Clusters and Cloud Storage solutions (e.g. Amazon S3). Finally, the **R-Connector** component is a bespoke component that connects user's SoBigData Identity with the R Application User Identity transparently, based on the OpenID Connect (OIDC) standard. OIDC is an identity layer over OAuth 2.0 that uses JSON Web Tokens (JWT), allowing third-party applications to verify the identity of the user and to obtain basic user profile information. OIDC is increasingly becoming a leading technology for Single Sign-On and identity provision on the Web.

[10] https://en.wikipedia.org/wiki/Filesystem_in_Userspace
[11] https://jackrabbit.apache.org/
[12] https://www.postgresql.org/

The *StorageHub Fuse Integration* library, the *R-Connector,* and the *StorageHub service* source code are available in the following Git repositories:

- StorageHub Fuse Integration: https://code-repo.d4science.org/gCubeSystem/sh-fuse-integration
- R-Connector: https://code-repo.d4science.org/gCubeSystem/RConnector

StorageHub Service:https://code-repo.d4science.org/gCubeSystem/storagehub

4.2 JupyterHub

Notebooks are becoming the *de facto* mechanism to provide an easy online coding system combined with interactive computing. In the context of SoBigData++, the deployment and operation of a Notebook service integrated with the SoBigData e-infrastructure has to satisfy the following requirements:

- The service should be integrated and accessible via a number of SoBigData Virtual Research Environments (VREs).
- The service should be integrated with SoBigData Authentication and Authorisation system.
- The access to SoBigData Workspace should be implemented, in order to provide access to user's files in the VRE Workspaces within the Notebook environment.
- Users should be able to access different flavours of Notebook, both in terms of pre-installed software and hardware capabilities.
- In addition to SoBigData Workspace, users need persistent storage to save a working copy of their notebooks across sessions.
- Users should be able to publish their notebooks in the SoBigData catalogue.

Considering that SoBigData e-infrastructure is powered by D4Science, and the positive experience of integration with D4Science performed during the AGINFRA+ project, JupyterHub has been deployed over a Kubernetes cluster[13] to offer a Notebook solution with seamless access from the SoBigData VREs graphical environment.

4.2.1 Design principles and implementation

The reference architecture of the JupyterHub solution in SoBigData e-infrastructure is depicted in Figure 16. Kubernetes is used for the automated provision of Notebook servers, which run as containers, offering the possibility to select the image flavour to run and to limit the CPU and RAM used by each instance. The Kubernetes environment provides an easy way to scale the deployment by adding new machines (worker nodes) to the existing cluster, expanding the capacity of the service whenever necessary.

[13] https://kubernetes.io/

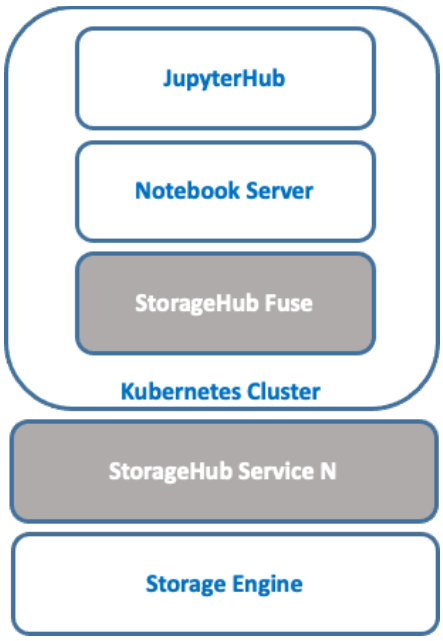

Figure. 16: The JupyterHub solution reference architecture

The project *Zero to JupyterHub*[14] is used to run JupyterHub on Kubernetes, as it offers many customisable add-ons that are needed for the integration with the SoBigData environment. One of them is the possibility to extend JupyterHub authentication system to use the SoBigData Authn/Authz framework[15]. Specifically, a new JupyterHub Authenticator class has been implemented to collect the user personal token from a VRE and use it to obtain user account details stored in the JupyterHub database. JupyterHub extensions specific for SoBigData are available[16], together with other customisations developed for EGI needs. Further integration activities were required to expose the JupyterHub GUI within the VRE portal, so that JupyterHub's GUI is displayed inside the VRE Web page.

JupyterHub allows users to run different server instances (which might differ in hardware specifications or pre-installed libraries) according to the user's need. SoBigData offers two different server options, an *Official* instance and a *Staging* instance (illustrated in Figure 17), which are based on the D4Sience Notebook distribution[17]. These instances are similar in capacity and have ready-to-use libraries that are useful to conduct the activities of SoBigData. The main difference is that the *Staging* instance is used to test the installation of new libraries and software components before publishing them for general use. Images are automatically built and published to a container registry (DockerHub) to make them available for the

[14] https://zero-to-jupyterhub.readthedocs.io/en/latest/
[15] https://dev.d4science.org/authorization
[16] https://github.com/EGI-Foundation/egi-notebooks-hub
[17] https://github.com/EGI-Foundation/egi-notebooks-images/tree/master/single-user-d4science

Kubernetes cluster. The system is highly flexible, so new servers can be easily allocated as needed in SoBigData, in order to meet the demands and needs of the users.

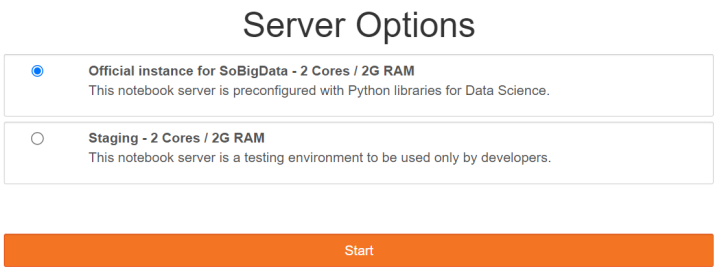

Figure 17. Notebooks server options

Apart from the regular Notebook servers, new servers are deployed to meet user needs whenever necessary. Figure 18 shows an example of the dedicated server created for the XAI Summer School during August 2022.

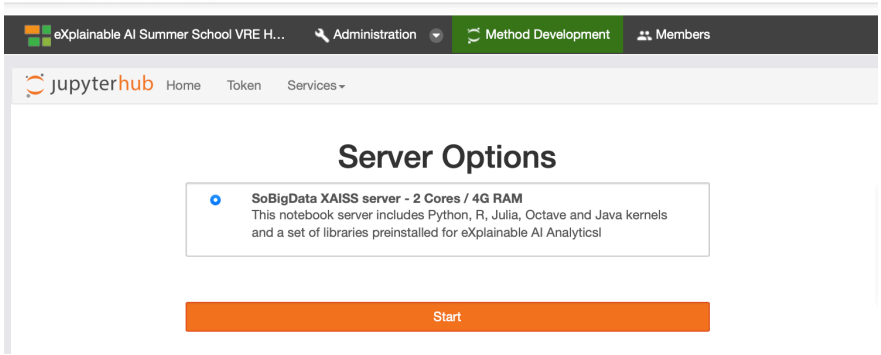

Figure 18. Notebooks dedicated server for an event

Further integration steps have been performed in order to implement access to D4Science workspaces. The access is implemented via the StorageHub Fuse library described in Section 4.1, which allows mounting the user workspace folders in the Notebook environment. The library integration[18] allows the workspace to be mounted in a sidecar container[19] for the user's main Notebook container and to be available under the user's home.

Finally, the Kubernetes cluster has been configured with an NFS server to host the users' volumes, in order to provide persistent working folders, and to host persistent shared data spaces for each of the VREs.

More details of the integration activities performed can be found in Figure 19.

[18] https://github.com/EGI-Foundation/egi-notebooks-images/tree/master/d4science-storage
[19] A container that runs alongside a main container to provide extra functionality.

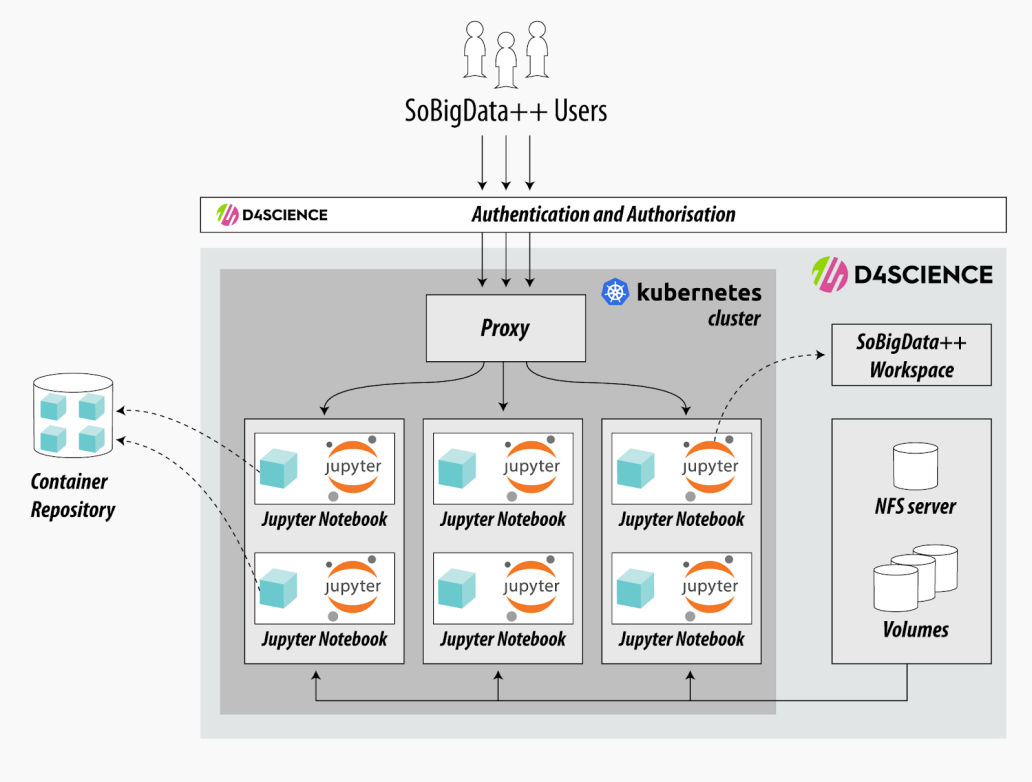

Figure 19: The JupyterHub deployment for the SoBigData e-infrastructure

The service has been operative to users of the SoBigDataLab VRE since November 2020, and is continuously enriched with new libraries and functionalities. The current service deployment has enough capacity to offer concurrent access to 80 users thanks to the 125 GB of memory of each cluster node. This hardware allocation is monitored regularly, and cluster resources are increased or decreased according to user demand.

The operation of the cluster has been enhanced with automatic backups of the NFS server volume, and cluster monitoring has been substantially improved. Additionally, a dedicated Grafana dashboard[20] has been deployed to aggregate relevant data coming from the cluster nodes and to configure notifications for administrators.

From the functional point of view, one of the requirements of the project is to allow users to publish notebooks in the catalogue of SoBigData. Sharing of notebooks is already available through the Workspace, which also enables their publishing in the catalogue. However, a future extension to publish notebooks directly from the Jupyter environment will be analysed and possibly implemented to further simplify and promote the publication of notebooks.

[20] https://grafana.com/

The current deployment of the service does not support multi-site clusters, so Kubernetes resources are provided by a single member of D4science.org: **CNR**[21]. Service provision from multiple sites will be investigated, which can guarantee user access to the service when it has to be partially stopped for an upgrade or in case of disruptive incidents that affect a single site.

4.3 Galaxy

Galaxy [3] is a framework for data integration and scientific workflow design and execution. It was originally developed to support genomics research and knowledge exchange, but has proven to be useful for the integration of arbitrary processing components in a multitude of different disciplines and domains.

The software is available under an Academic License and can be freely downloaded and deployed as a stand-alone Web platform. The basic distribution offers a generic set of widely used tools and can be extended with additional components (added via the *Galaxy Tool Shed* repository) and custom scripts (added by the administrator of a deployment). Each added component in Galaxy contains annotations that provide basic information and guidelines for its usage, and can be combined in workflows using the included design environment. An example of the Galaxy user interface can be seen in Figure 20.

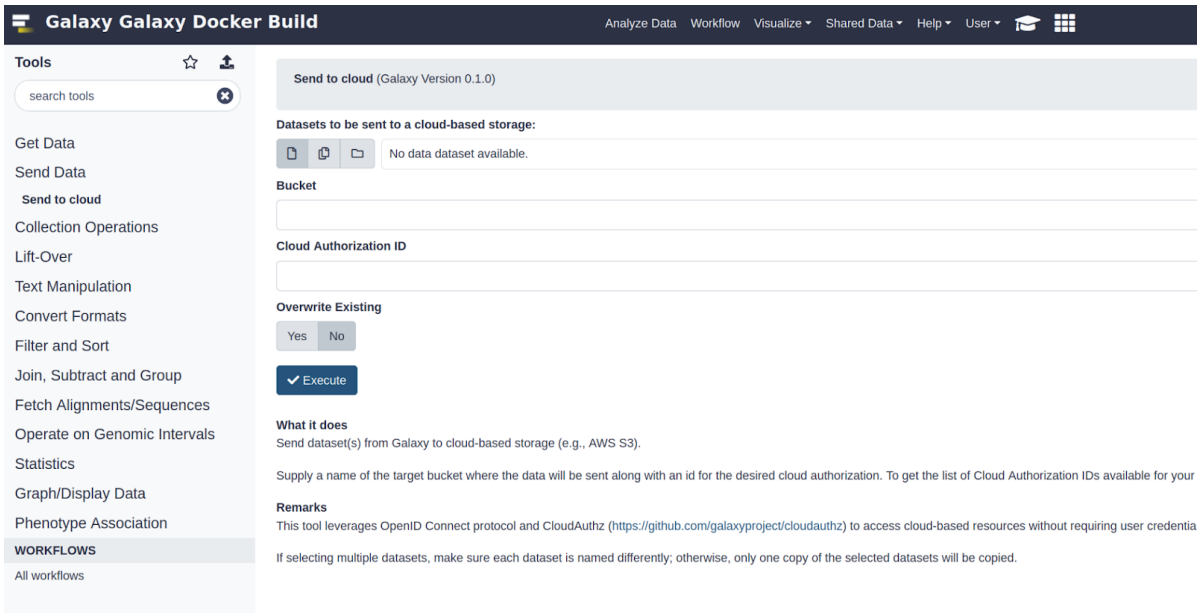

Figure 20: Example of Tools in the Galaxy GUI

[21] Consiglio Nazionale delle Ricerche (Italy) https://www.cnr.it/en

4.3.1 Design principles and implementation

The reference architecture of the Galaxy solution in SoBigData e-infrastructure is depicted in Figure 21. Galaxy is deployed in a Kubernetes cluster with three nodes (represented as vertical boxes) that run different instances of Galaxy. Each VRE uses an independent Galaxy server on the same Kubernetes cluster and is built from the upstream Docker container image of the Galaxy distribution (obtained from a Container Repository), which is configured and extended to:

- Enable the authentication of users with the SoBigData AAI framework.
- Contact the Social Mining Analytics Engine (SMAE) periodically, to populate a list of tools available in the VREs and make them available for defining workflows in Galaxy.
- Run jobs as containers in the existing Kubernetes cluster.

One of the advantages of having separate instances of Galaxy for each VRE is that they are isolated and can be easily configured to target the specific needs of the users of the different VREs.

Tools from the Social Mining Analytics Engine can be invoked as part of any workflow, and SoBigData extension for Galaxy takes care of: (i) Sending the right user credentials, (ii) Converting input and output data from Galaxy to the DataMiner formats and (iii) Registering the outputs of the tools as *data* in Galaxy, so that they can be further used in other workflow steps.

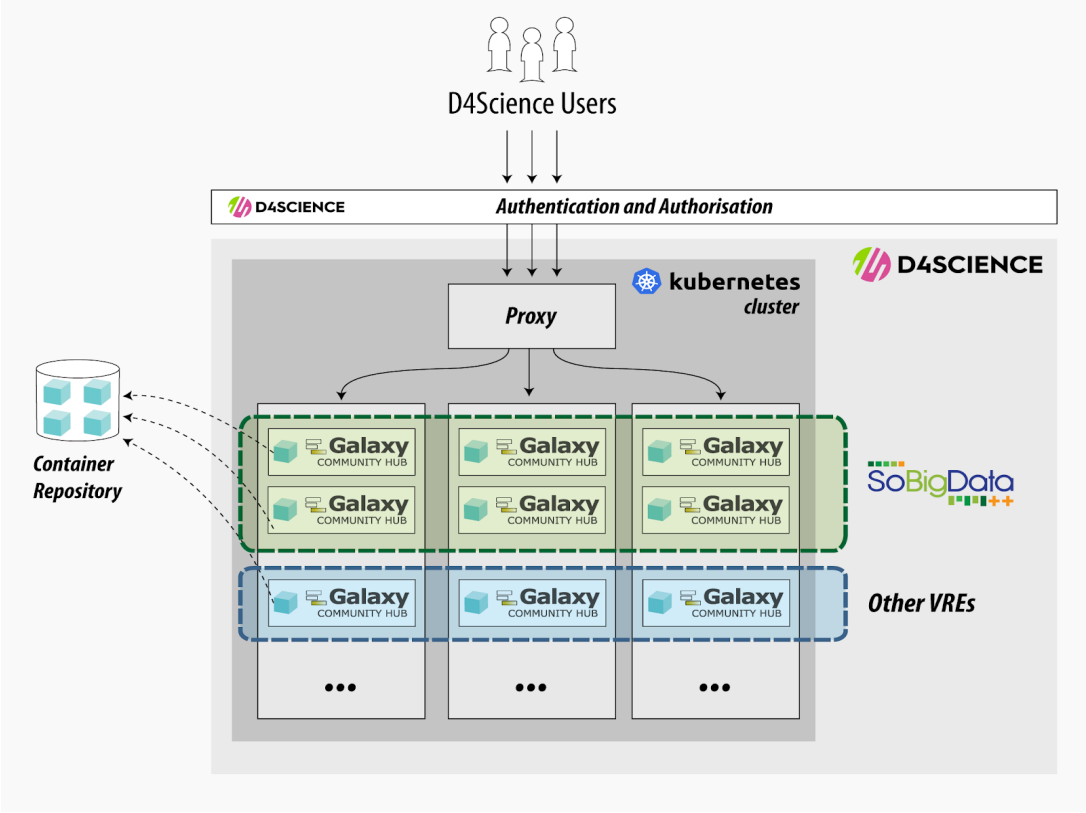

Figure 21: Galaxy reference architecture in the SoBigData e-infrastructure

The implementation of Galaxy in SoBigData is ongoing, and the platform is expected to be ready in the coming months.

5 Online Science Monitoring Dashboard: design principles and architecture

The Online Science Monitoring dashboard monitors and quantifies the outputs of the SoBigData infrastructure in the scholarly communication ecosystem. It identifies every research product (publications, datasets, software, and other types) produced with assistance of the infrastructure available in the OpenAIRE Research Graph[22]) which, as of December 2022, includes more than 170M research products linked to funding projects, organisations and authors. Once identified, the products are analysed to produce statistics and charts with indicators about different aspects of Open Science and research impact (e.g. Open Access, links between research products, data re-use, etc). The dashboard also acts as a single entry point for users to discover, search, browse, and get access to research products related to the infrastructure and hosted in several scholarly communication sources (e.g. repositories, journals, archives, etc).

The identification of SoBigData products is performed by scanning the OpenAIRE Research Graph for links to the three funding projects of the infrastructure (SoBigData, SoBigData++, and SoBigData RI PPP[23]), which can be found in the metadata records describing the research products or in the acknowledgement statements of the full texts of the publications. For the identification of funding information in the full texts, a full-text mining algorithm is run on all ~10 million Open Access full texts aggregated in the OpenAIRE infrastructure.

The dashboard is also connected to the Zenodo community dedicated to SoBigData[24], where researchers transparently publish research objects that are produced using the tools and services of the infrastructure. If a research product has not been identified automatically, users are also able to manually add it to the dashboard by using the *Link* functionality of the OpenAIRE EXPLORE portal[25] or the dashboard itself.

The monitoring dashboard is currently accessible to the project coordinator at https://monitor.openaire.eu/dashboard/sobigdata.

5.1 Design principles and implementation

The Online Science Monitoring dashboard is built with the OpenAIRE Research Community Dashboard[26] [1], a framework for the set-up and deployment of highly configurable portals for research communities and infrastructures. The capacity of the dashboards to support several configuration scenarios is the leading principle that guided the design, in order to be able to address different requirements of different stakeholders (discipline-specific research communities and mature and young research infrastructures). The configuration options can be defined using a Manager Dashboard that allows selected users - called *managers* or *curators* to:

[22] https://graph.openaire.eu
[23] SoBigData RI Preparatory Phase Project, ga. n. 101079043
[24] https://zenodo.org/communities/sobigdata
[25] https://explore.openaire.eu
[26] Accessible at: https://connect.openaire.eu

- Configure the algorithms that OpenAIRE will use to identify the relevant products by specifying the funded projects and the data sources where the products and their descriptive metadata records are available (including specific Zenodo communities). When the dashboard serves a disciplinary research community, managers can also specify a list of keywords and subject terms: all the research products whose metadata matches such terms will be included in the dashboard. This latter option is not available for dashboard serving research infrastructures. On the other hand, managers of a research infrastructure can use the Manager dashboard to define, fine-tune, and test the full-text mining rules.
- Manage end-user claims: dashboard users can use the *Link* functionality of the OpenAIRE EXPLORE portal and the dashboard to assert that a product is relevant for the infrastructure. Managers can confirm or reject these assertions.
- Configure statistics and charts: OpenAIRE makes available a number of statistics, such as the percentage of Open Access publications and data, the number of publications linked to data and software and the growth of Open Access publishing through the years. Managers can decide which of these are public or not, for internal monitoring.
- Configure the look and feel of the dashboard, by choosing colours, fonts, and logos to reflect the identity of the infrastructure it serves.
- Personalise the content of the Web pages, by enabling/disabling menu items and modifying the content of the dashboard pages, for example to include useful instructions about best practices on Open Science to be followed by the researchers of the infrastructure.

The specified configurations are used by the other components in order to provide a unique and tailored experience. In particular, the configuration of the algorithms is used by the Oozie jobs that analyse the OpenAIRE Research Graph and enriches it by tagging the products that are relevant to the infrastructure. The products and their tags are then: (i) Indexed on a Solr cluster[27] to support search and browse functionality on top of which the API[28] and the dashboard Web GUI[29] are built upon. (ii) Analysed with the Statistics Tool based on Apache Impala[30] for the calculation of statistics and the production of charts that are shown on the Web GUI. Figure 22 presents these components inside the architecture diagram of the Online Science Monitoring dashboard.

[27] https://solr.apache.org/
[28] Accessible at: https://develop.openaire.eu
[29] Accessible by managers at: https://monitor.openaire.eu/dashboard/sobigdata …
[30] https://impala.apache.org/

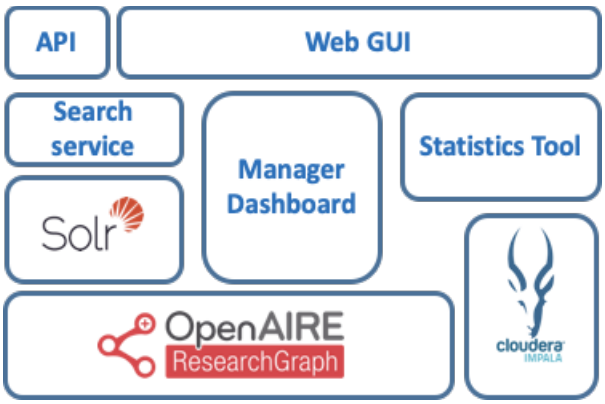

Figure 22: Architecture of the Online Science Monitoring Dashboard

6 Conclusions: Release Management - software continuous integration

The majority of the components described in this report have been contributed by the gCube software system and other open-source systems as RStudio, JupyterHub, and many others.

gCube [2] is an open-source software system designed and implemented to enable the creation and operation of a Service-Oriented Infrastructure that supports the definition of Virtual Research Environments (VREs). It is exploited by the D4Science infrastructure and, subsequently, by SoBigData, and its components are widely used in several application frameworks.

It covers a rich array of "mediators" for the integration and exploitation of services and facilities offered by other tools and systems like the ones listed above. This system has been implemented with the support of the European Commission in the context of a series of projects[31].

As depicted in Figure 23, the enabling technologies selected to properly support the continuous integration process in gCube are Gitea[32] (as Git hosting service), Jenkins[33] (as automation server) and Maven[34] (as project management and comprehension tool).

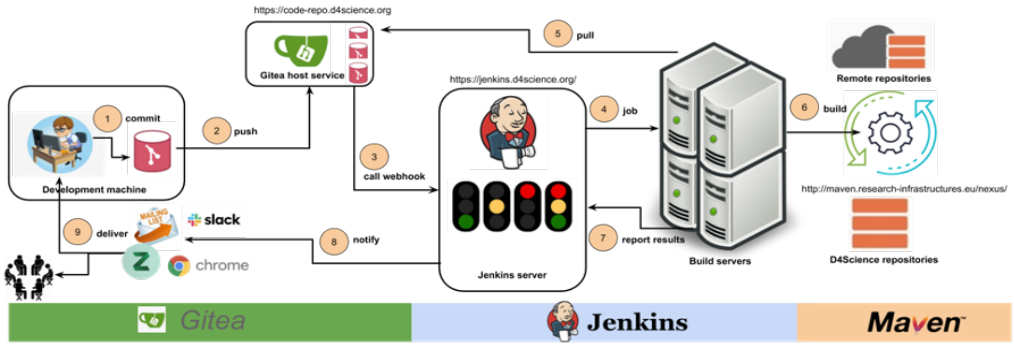

Figure 23: Continuous integration workflow

The proper configuration and interactions of these systems are the foundation for automating part of the software development process, with continuous integration and facilitating technical aspects of continuous delivery. The following Sections 6.1, 6.2 and 6.3 will discuss these three systems in more detail.

6.1 Version Control System

A Version Control System is a piece of software that keeps track of modifications to files in a repository. These systems are normally used to record source code. The gCube software system adopts Git as its code management system.

[31] See https://www.gcube-system.org/about for the complete list of projects
[32] https://gitea.io/
[33] https://www.jenkins.io/
[34] https://maven.apache.org/

SoBigData uses a complete code hosting Git solution based in Gitea, which has been deployed on the D4Science infrastructure[35]. (This supports the management of Git repositories and its settings to enable the creation of new components.

gCube currently contains 274 repositories in Gitea that are managed by 16 developers[36].

6.2 Build Tool

A build tool is a tool that automates the software creation process. One of their main advantages of automating the build process is that they minimise the risk of human error when compiling software manually. Additionally, automated build tools operate faster than humans, as they have to perform the same steps manually. gCube uses Maven as its build tool.

Maven requires a dependency repository to work properly. The D4Science infrastructure makes use of an instance of Nexus[37], which is a free artefact repository with universal support for many Maven-compatible formats. The D4Science Nexus instance[38] is configured to host the following repositories:

- gcube-snapshots: for artefacts under development.
- gcube-staging: for artefacts to be deployed on the D4Science production infrastructure.
- gcube-releases: for artefacts deployed on the D4Science production infrastructure.

6.3 Continuous Integration

Continuous integration refers to all the steps necessary to create, remove, modify and monitor a software component. This has been integrated and deployed in the D4Science infrastructure through Jenkins[39]. Jenkins enables the creation of workflows to integrate components, which are often referred to as *pipelines*.One of the biggest challenges of a Continuous Integration pipeline is to guarantee concurrent operation: poorly configured pipelines can run multiple builds of the same project simultaneously on the same slave worker can interfere with each other, creating inconsistent situations..This is the case when every repository modification (known as *commit*) is immediately applied to the *master* branch or when the Git repository is wrongly used as a backup system. To solve this problem, gCube implements the following rules and practices for any Jenkins project:

- A Jenkins project always uses the *master* branch.
- If a Jenkins project is configured to build a second branch, this branch must generate a software artefact with a different version than the one from the *master* branch.
- The development of new features and bug fixing are done in dedicated branches (task branching).

[35] https://code-repo.d4science.org/
[36] https://code-repo.d4science.org/gCubeSystem
[37] https://www.sonatype.com/nexus-repository-oss
[38] http://nexus.d4science.org/nexus/content/repositories
[39] https://jenkins.d4science.org

- Merges into the *master* branch are performed only when the feature or fix implemented by the secondary branch is stable.
- Commits are not always pushed to the remote repository, but only when they add a significant, self-contained and consistent piece of working code.
- The *master* branch code must always be in a releasable state.

By adopting this set of rules and practices, builds are triggered only when a stable feature is merged into *master*, while commits on the other branches do not involve Jenkins. If two branches are built at the same time in a Jenkins project (which should be temporary), their different versions guarantee that there are no conflicts in the published artefacts. The history of all builds conducted by Jenkins is available for consultation[40].

[40] https://jenkins.d4science.org/view/all/builds

References

[1] Baglioni M., Bardi A., Kokogiannaki A., Manghi P., Iatropoulou K., Principe P., Vieira A., Nielsen L. H., Dimitropoulos H., Foufoulas I., Manola N., Atzori C., La Bruzzo S., Lazzeri E., Artini M., De Bonis M., Dell'Amico A. (2019) ***The OpenAIRE Research Community Dashboard: On Blending Scientific Workflows and Scientific Publishing***. In: Doucet A., Isaac A., Golub K., Aalberg T., Jatowt A. (eds) Digital Libraries for Open Knowledge. TPDL 2019. Lecture Notes in Computer Science, vol 11799. Springer, Cham. https://doi.org/10.1007/978-3-030-30760-8_5

[2] Assante M., Candela L., Castelli D. , Cirillo R., Coro G., Frosini L. , Lelii L. , Mangiacrapa F., Marioli V. , Pagano P. , Panichi G., Perciante C., Sinibaldi F. (2019) ***The gCube system: Delivering Virtual Research Environments as-a-Service***. Future Gener. Comput. Syst. 95: 445-453 https://doi.org/10.1016/j.future.2018.10.035

[3] Afgan E., Baker D., Batut B., van den Beek M., Bouvier D., Čech M., Chilton J., Clements D., Coraor N., Grüning B., Guerler A., Hillman-Jackson J., Jalili B., Rasche H., Soranzo N., Goecks J., Taylor J., Nekrutenko A., Blankenberg D. (2018) ***The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update***. Nucleic Acids Research, 46, Issue W1: 537-544, https://doi.org/10.1093/nar/gky379

[4] Assante M., Candela L., D. Castelli D. , R. Cirillo R., G. Coro G., L. Frosini L. , L. Lelii L. , F. Mangiacrapa F., Pagano P. , Panichi G., Sinibaldi F. (2019) ***Enacting open science by D4Science***. Future Gener. Comput. Syst. 101: 555-563 https://doi.org/10.1016/j.future.2019.05.063