NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

PROGRAM OF POSTGRADUATE STUDIES
"Computing Systems: Software and Hardware"

Master Thesis

Management of Scientific Analysis and Simulation Workflows over High Performance Computing Systems

Sofia N. Karvounari

Supervisor **Yannis Ioannidis,** Professor

Athens

MAY 2022

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
"ΥΠΟΛΟΓΙΣΤΙΚΑ ΣΥΣΤΗΜΑΤΑ: ΛΟΓΙΣΜΙΚΟ ΚΑΙ ΥΛΙΚΟ"

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Διαχείριση Επιστημονικών Ροών Εργασίας Ανάλυσης και Εξομοίωσης Υπερυπολογιστικών Υποδομών

Σοφία Ν. Καρβουνάρη

Επιβλέπων **Ιωάννης Ιωαννίδης,** Καθηγητής

ΑΘΗΝΑ

ΜΑΙΟΣ 2022

Master Thesis

Management of Scientific Analysis and Simulation Workflows over High Performance Computing Systems

Sofia N. Karvounari

A.M.: M1531

Supervisor **Yannis Ioannidis,** Professor

May 2022

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Διαχείριση Επιστημονικών Ροών Εργασίας Ανάλυσης και Εξομοίωσης Υπερυπολογιστικών Υποδομών

Σοφία Ν. Καρβουνάρη

Α.Μ.: Μ1531

Επιβλέπων **Ιωάννης Ιωαννίδης,** Καθηγητής

Μάιος 2022

ABSTRACT

During the last decade, research communities had been seeking ways for addressing complex scientific problems by harnessing the power of high-performant and scalable computing infrastructures. Many of these communities belong in the area of Life Sciences, from bioinformatics and medicine, up to neuroscience and brain research. Most modern scientific problems require powerful underlying computing infrastructures that can handle the load, the complexity, the execution of time consuming tasks and the big data that are produced. These powerful infrastructures consist of High Performance Computing systems with scalable computing, storage and network resources for analysis, simulation and execution of high-intense and complex computing tasks. European Research Infrastructures offer such High Performance Computing systems to European scientists and researchers in order to adhere to Open Science, research and innovation.

More and more scientific communities create, manage and share complex scientific workflows in an automated and user friendly way while using High Performance Computing systems in order to execute them. Moving towards an Open Science, automation of scientific workflows is not enough. In order to have reproducible scientific workflows, it is mandatory for scientists to describe workflows in a standard and common format. This offers characteristics such as portability, scalability and accessibility. In addition, it allows the immediate use of methods and data *as-is* or as inputs to other scientific workflows, creating important scientific value chains.

In the current thesis, we focus on EBRAINS, a digital European Research Infrastructure, which specializes in brain neuroscience, one of the most ambitious, promising, and important fields of research that has been a priority for Europe for over a decade, offering tools, services and data to its users. FENIX ICEI, which is the EBRAINS underlying infrastructure, provides a plethora of High Performance Computing services, ranging from scalable and interactive computing, up to virtual machines and containerization services, as well as highly performant data storage services.

After an extensive research, we recorded the state of the art High Performance Computing systems while focusing on their architecture. Also, we collected different ways of describing scientific workflows which are coupled with the different software in which they are executed. Last but not least, we collected different ways in which research communities describe scientific workflows focusing on Life Science. We also analysed and assessed the EBRAINS users' needs while understanding what is the current state, what are EBRAINS users' requirements and how to full fill those.

The current thesis' goal is the establishment of a pilot workflow management system for EBRAINS. Our approach based in four pillars:

- First, we chose Common Workflow Language (CWL), an open, common and emerging standard format, as a way for describing both workflows and EBRAINS tools. CWL offers portability and decouples the description of scientific workflows from the execution of them. EBRAINS users will use CWL for authoring workflows and EBRAINS tools.
- Second, we packaged EBRAINS tools with their required libraries, dependencies and binaries via containerisation method. In that way these bundled EBRAINS tools can be used as workflow steps, significantly enhancing their interoperability and portability. EBRAINS users will also package their tools in the same way, in order to be used as workflow steps by others.
- Third, we deployed a variety of workflow engines for workflow submission and monitoring on top of the EBRAINS underlying infrastructure and chose those that

best fit the scientific and interoperability requirements of EBRAINS. EBRAINS users will use Command Line Interfaces (CLI) for workflow submission and monitoring.

- We proposed an EBRAINS Hub for scientists to easily find, access and store their scientific workflows. In the current thesis, we designed how this Hub will look like taking into consideration the already available EBRAINS Knowledge Graph (KG) service. As a future step, integrating such a feature with EBRAINS KG will take place.
- Finally, we proposed an EBRAINS central place, a graphical user interface (GUI) in which scientists can submit, monitor and parametrize workflows. The proposed GUI has been fully designed but not implemented in the scope of the current thesis. The design process is based on high fidelity mock-ups. EBRAINS users will use this GUI instead of a CLI for a more user friendly experience. The real execution takes place in the FENIX ICEI underlying infrastructure in an opaque way, in the same way as using the CLI.

SUBJECT AREA: Workflow management

KEYWORDS: Workflows, Big data, High Performance Computing systems

ΠΕΡΙΛΗΨΗ

Την τελευταία δεκαετία, οι ερευνητικές κοινότητες αναζητούν τρόπους αντιμετώπισης πολύπλοκων επιστημονικών προβλημάτων αξιοποιώντας τη δύναμη των Υπολογιστικών υποδομών Υψηλής Απόδοσης (ΥΥΑ). Πολλές από αυτές τις κοινότητες ανήκουν στον τομέα των Επιστημών Ζωής, από τη βιοπληροφορική και την ιατρική, μέχρι τη νευροεπιστήμη και την έρευνα του εγκεφάλου. Τα περισσότερα σύγχρονα επιστημονικά προβλήματα απαιτούν ισχυρές υπολογιστικές υποδομές που μπορούν να χειριστούν το φορτίο, την πολυπλοκότητα, τη χρονοβόρα εκτέλεση υπολογιστικών εργασιών και τα μεγάλα δεδομένα που παράγονται. Αυτές οι ισχυρές υποδομές αποτελούνται από συστήματα ΥΥΑ με κλιμακωτούς υπολογιστικούς, δικτυακούς πόρους και αποθήκευση που επιτρέπουν την ανάλυση, προσομοίωση και εκτέλεση πολύπλοκων εργασιών. Οι ΥΥΑ προσφέρονται από Ευρωπαϊκές Ερευνητικές Υποδομές σε Ευρωπαίους επιστήμονες και ερευνητές ώστε να συνδράμουν στην Ανοιχτή έρευνα, επιστήμη και καινοτομία.

Όλο και περισσότερες ερευνητικές κοινότητες δημιουργούν, διαχειρίζονται και μοιράζονται πολύπλοκες επιστημονικές ροές εργασίας με αυτοματοποιημένο, φιλικό προς το χρήστη τρόπο, ενώ χρησιμοποιούν συστήματα ΥΥΑ για την εκτέλεσή τους. Προχωρώντας προς την Ανοικτή Επιστήμη, η αυτοματοποίηση των επιστημονικών ροών εργασίας δεν είναι αρκετή. Προκειμένου να υπάρχουν αναπαράξιμες επιστημονικές ροές, οι επιστήμονες πρέπει να περιγράφουν ροές εργασίας με μια κοινή και τυπική μορφή. Αυτό προσφέρει χαρακτηριστικά όπως φορητότητα, επεκτασιμότητα και προσβασιμότητα. Επιπλέον, επιτρέπεται η άμεση χρήση μεθόδων και δεδομένων ως έχουν ή ως είσοδος σε άλλες επιστημονικές ροές εργασίας.

Στην παρούσα διπλωματική εργασία, εστιάζουμε στην ψηφιακή Ευρωπαϊκή Ερευνητική Υποδομή EBRAINS, που ειδικεύεται στη νευροεπιστήμη του εγκεφάλου, ένα από τα πιο φιλόδοξα και σημαντικά πεδία έρευνας που αποτελεί προτεραιότητα για την Ευρώπη πάνω από μια δεκαετία, προσφέροντας εργαλεία, υπηρεσίες και δεδομένα στους χρήστες της. Η υποκείμενη υπολογιστική υποδομή, το FENIX ICEI, παρέχει μια πληθώρα υπηρεσιών ΥΥΑ, που κυμαίνονται από κλιμακώσιμη και διαδραστική υπολογιστική, έως εικονικές μηχανές και υπηρεσίες αποθήκευσης δεδομένων υψηλής απόδοσης.

Μέσα από μια εκτενή έρευνα, καταγράψαμε τεχνολογίες αιχμής συστημάτων ΥΥΑ, επικεντρώνοντας την προσοχή μας στην παρούσα αρχιτεκτονική τους. Επίσης, συγκεντρώσαμε διαφορετικούς τρόπους αναπαράστασης επιστημονικών ροών εργασίας που είναι άρρηκτα συνδεδεμένοι με τα λογισμικά στα οποία εκτελούνται. Τέλος, παρουσιάσαμε τρόπους προτυποποιημένης μοντελοποίησης και εκτέλεσης συγγραφής επιστημονικών ροών εργασιών επικεντρώνοντας το ενδιαφέρον μας σε ερευνητικές κοινότητες γύρω από τις Επιστήμη Ζωής. Στη συνέχεια, αναλύσαμε και αξιολογήσαμε τις ειδικές ανάγκες των χρηστών της υποδομής EBRAINS. Συγκεντρώσαμε τι μπορούν να κάνουν οι χρήστες στην παρούσα χρονική στιγμή, ποιες είναι οι απαιτήσεις τους και πως αυτές μπορούν να ικανοποιηθούν.

Η παρούσα διπλωματική εργασία έχει στόχο την δημιουργία ενός πιλοτικού συστήματος διαχείρισης ροής εργασιών για την υποδομή EBRAINS. Η προσέγγισή μας στην διπλωματική εργασία αυτή βασίστηκε σε τέσσερις πυλώνες:

- Αρχικά, επιλέξαμε την Common Workflow Language (CWL), ένα ανοικτό πρότυπο για τον ορισμό τόσο των ροών εργασίας, όσο και των EBRAINS εργαλείων. Η CWL προσφέρει φορητότητα και αποσυνδέει την περιγραφή ροών εργασίας από τα περιβάλλοντα εκτέλεσης. Οι χρήστες της EBRAINS υποδομής θα χρησιμοποιούν την CWL για την συγγραφή των επιστημονικών ροών εργασιών και των EBRAINS εργαλείων.

- Στη συνέχεια, πακετάραμε (containarised) EBRAINS εργαλεία με τις απαιτούμενες βιβλιοθήκες, εξαρτήσεις και δυαδικά αρχεία. Με τον τρόπο αυτό τα EBRAINS εργαλεία μπορούν να χρησιμοποιηθούν ως βήματα σε ροές εργασιών, ενισχύοντας σημαντικά τη διαλειτουργικότητα και τη φορητότητα. Οι χρήστες της EBRAINS υποδομής θα πακετάρουν με τον ίδιο τρόπο τα δικά τους εργαλεία, ώστε αυτά να μπορούν να χρησιμοποιηθούν από άλλους στη συγγραφή δικών τους επιστημονικών ροών.
- Επιπλέον, εγκαταστήσαμε ένα πλήθος διαφορετικών μηχανών για υποβολή και παρακολούθηση ροών δεδομένων στις ΥΥΑ του EBRAINS και επιλέξαμε αυτές που ικανοποιούν με το βέλτιστο τρόπο τις επιστημονικές και τεχνικές απαιτήσεις του EBRAINS. Οι χρήστες της EBRAINS υποδομής θα χρησιμοποιούν τη Διεπαφή Γραμμής Εντολών (ΔΓΕ) για την υποβολή και παρακολούθηση των επιστημονικών ροών εργασίας τους.
- Επίσης, προτείναμε να υπάρξει ένα αποθετήριο στην EBRAINS υποδομή ώστε οι επιστήμονες της υποδομής να βρίσκουν και αποθηκεύουν εύκολα τις επιστημονικές ροές δεδομένων. Στα πλαίσια της παρούσας διπλωματικής εργασία σχεδιάσαμε τον τρόπο απεικόνισης ενός τέτοιου αποθετηρίου έχοντας ως βάση την ήδη υπάρχουσα EBRAINS υπηρεσία (Knowledge Graph). Επόμενο βήμα και εκτός της παρούσας διπλωματικής εργασίας θα είναι η ενσωμάτωση του αποθετηρίου με το EBRAINS Knowledge Graph σύμφωνα με τις τεχνικές απαιτήσεις που θα υπάρχουν για ένα ολοκληρωμένο σύστημα διαχείρισης ροών εργασίας.
- Τέλος, προτείναμε να δημιουργηθεί ένα γραφικό περιβάλλον διεπαφής χρήστη (ΠΔΧ) στο οποίο οι χρήστες της υποδομής να μπορούν να υποβάλλουν, παρακολουθούν και παραμετροποιούν τις επιστημονικές ροές δεδομένων. Σε αυτή τη διπλωματική σχεδιάσαμε πως θα απεικονίζεται η διεπαφή με τεχνικές υψηλής πιστότητας. Στο μέλλον και εκτός του πλαισίου της παρούσας διπλωματικής, θα πραγματοποιηθεί η υλοποίηση μιας τέτοιας διεπαφής. Οι χρήστες της EBRAINS υποδομής θα χρησιμοποιούν την διεπαφή αυτή αντί της ΔΓΕ για ένα πιο εύχρηστο και φιλικό προς εκείνους περιβάλλον. Η πραγματική εκτέλεση των επιστημονικών ροών γίνεται στην υποκείμενη ΥΥΑ υποδομή FENIX ICEI με εναν διάφανο για τους EBRAINS χρήστες όπως ακριβώς και με τη χρήση της ΔΓΕ.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Διαχείριση Επιστημονικών Ροών Εργασίας

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Ροές εργασίας, Μεγάλα Δεδομένα, Υπολογιστές Υψηλής Απόδοσης

To my loving family

Maria & Nektarios, Konstantinos & Antonis.

ACKNOWLEDGEMENTS

I would first like to acknowledge my supervisor, Professor Ioannis Ioannidis for his support, and trust to me to complete the current thesis. His inspiration all those years, from undergraduate to graduate tutor as well as leader in projects I was participating, is precious.

I would also like to express my gratitude to Spiros Athanasiou for his feedback, corrections and for making this work possible. I am glad you guided and assisted me in the correct direction.

I wish to thank all my colleagues at Athena Research Center participating in the Human Brain Project (HBP) for their patience, advice and generous help given at any time. Specifically, I would like to thank Eleni Mathioulaki for her valuable work and insights which helped concluding the current thesis.

I would also like to thank Roleplay[1] for the available mock ups used under the current thesis.

My sincere appreciation to my dear friend, Aristotelis Kasomoulis, for his help in designing a number of figures included in the current thesis. Thank you for your devoted time and support to me.

Last but not least, my warmest regards to my beloved family for simply believing in me.

1 https://www.roleplay.gr/

CONTENTS

LIST OF FIGURES

Figure 39 Preview of the standardized scientific workflow as a directed acyclic graphs provided by CWLViewer, where EBRAINS tools are the nodes of the graph, and data flowing between steps are the edges.
Figure 40: Additional information for standardized scientific workflows, such as the Digital Object Identifier (DOI), as well as the CWL definition. Users will be able to execute workflows in a dedicated endpoint (graphical user interface) by pressing the Play button
Figure 41: Graphic User Interface, Rabix, for visually describing workflows. Users simply drag and drop already defined tools and connects the inputs as well as outputs of one tool to the other. In that way a graph is created where with tools as nodes and data flowing between them as edges.
Figure 42:Users provides the input parameters and the input data for the scientific workflow recipe to be submitted.
Figure 43:EBRAINS users can get notified and check the status, logs and outputs of the submitted scientific workflows.
Figure 44:Logs and output provided to the user in an opaque way via the dedicated endpoint.

1. INTRODUCTION

Nowadays more than ever, more people understand the scientific impact in humans' life. Now more than ever, many people understand how scientists, genetics, epidemiologists, can benefit from science and how they are able to solve problems that humanity suffers from. People today are more capable of understanding how epidemiologists, genetics, and biologist's scientific work can lead to objectives with valuable impacts in human life, like offering vaccines for viruses to end a pandemic. But how many of these people understand what are the means that these scientists have so they can succeed in their goals? How many of these know what is the computational power that is needed in order to run simulations, models and tests to achieve their scientific objectives? How many of them know the amount of data that is produced, how much time simulations run and how much power is consumed?

For many people in the last decades, personal computers are a critical part of their everyday life routine, their education, their work, even their entertainment. Even though personal computers play an integral role in human life, it is also true that they do not have enough power or capacity for data intensive and time-consuming simulations, models and testing scenarios like the ones scientists would like to execute. In Europe, research infrastructures with cutting edge technologies are provided to scientists from scientific fields, like neuroscience, medical, and health to accomplish scientific objectives. These research infrastructures often consist of High Performance Computing systems that allow scientists to execute scalable and interactive jobs while providing reliable storages for archiving produced big data. In the current thesis, we delve into High Performance Computing systems to better understand in which cases they are used by the neuroscientists and the state-of-the-art capabilities that they offer. Architecture diagrams are also included to better understand the construction and organization of these systems.

From the early beginning, scientists from different scientific fields created chains of processes with data flowing from one process to another for the sake of scientific impact and research innovation. Typically these processes were computational steps that manipulated data by analysing, simulating or visualizing it. These chains of computational steps are often known as workflows. Up until now, the description of the workflows was strictly associated with the underlying software that handled the execution. This coupling of defining scientific work with executing it, made standardisation an important asset that needed to be explored. Nowadays, there are technologies and means for scientists to describe their scientific work in a common, standard, open and widely acceptable way for other scientific communities to understand as well as for compatible underlying software to execute. In the current thesis, we introduce these ways and technologies of defining computational workflows in a standard way.

In this thesis, we focus on EBRAINS, which is a European research infrastructure for neuroscientists dedicated to brain related research. EBRAINS provides a powerful underlying infrastructure to its users consisting of High Performance Computing systems for large scale jobs that need to run for a long time, for interactive jobs that need enormous memory and large computing machines, as well as for enormous produced data to be safely stored after their creation. Neuroscientists at EBRAINS have a large set of tools and services at their disposal, ranging from analysing to simulating big data and brain atlases that can broadcast 2-dimensional and 3-dimensional fractions of a brain, up to mapping neural networks of different levels. Furthermore, EBRAINS users have access to highly valuable data related to the brain, in order to associate it with their scientific work.

After we assessed and summarized the current limitations that EBRAINS users face, we propose the establishment of an EBRAINS pilot workflow management system for neuroscientists to combine services, tools and data found under EBRAINS in order to create standardised scientific computational workflows that will be easily reproducible, reusable, findable and accessible by other scientific communities. These standardised workflows will be submitted in the EBRAINS' underlying powerful infrastructure on top of compatible software (workflow engines) installed and deployed for execution and monitoring purposes for the sake of the current thesis. With this pilot workflow management system approach, neuroscientists at EBRAINS can easily associate their practical work with their theoretical work in different open repositories, publications and papers and can have all the needed components of their work (data, tools and services) in place under the same format. As a last step, we proposed ways of having an EBRAINS Hub and a Graphical User Interface for EBRAINS users to store, access find and for submitting, monitoring and parametrizing workflows respectively. We used high fidelity interactive mock-ups for designing these two new features. The real integration and implementation of these, are out of the scope of the thesis.

2. STATE OF THE ART

In this section we present some of the cutting-edge and state of the art technologies with respect to workflows, as well as High Performance Computing systems. We firstly try to separate terms that have to do with workflows terminology, alleviating any misconceptions that may arise. We briefly explain the differences between business and scientific workflows as well as some of the similarities that both share.

The focus of the current thesis is **scientific computational workflows**, which are associated with computational steps accomplishing data analysis and data simulation, or in general data manipulation tasks, for achieving scientific objectives [2]. After the clear separation between different types of workflows, we present the different ways of executing them used by different scientific fields in order to offer automation, handling errors, failures, without the need of manual interventions. Even though there are many workflow engines that are capable of these specific attributes, we mostly focus on a small number of them which can run on top of High Performance Computing systems. We do not intend to make an extensive list of features and capabilities related to different engines, but to familiarize the readers with the different technologies that already exist. Our intentions lay in the fact that every workflow engine has its own ways of describing workflows, thus making the learning curve long enough for scientists who want to dive into more than one workflow engines.

2.1 Workflows

In this subsection we provide aspects on defining and executing workflows. We will try to properly and strictly define different terminologies related to workflows, since they are broadly used terms that have different meanings under different concepts. Briefly, under this thesis we are mostly interested in scientific computational workflows where steps are computing tasks that manipulate data and are executed on top of HPC systems. Further, we present different management systems that take care of the execution and monitoring of workflows. Workflow management systems provide automation to the scientists using them; prior to them, users would need to do a lot of manual preparation tasks, such as data manipulation, invoke different steps in the appropriate order, as well as deal with errors and failures. This obviously is time consuming and can entail in a lot of errors that again users would need to handle manually.

2.1.1 Workflow terminology

In general, a workflow "*is a sequence of industrial, administrative, or other processes through which a piece of work passes from initiation to completion*"[2]. With respect to the Cambridge Dictionary[3], workflow "*is the way that a particular type of work is organised, or the order of the stages in a particular work process*". Processes in a workflow can be linked together to create directed acyclic graphs (DAGs), loops or branches [21]. If there are no dependencies between them, processes can run simultaneously. This is the main difference with the synonym term "pipeline", which is a linear sequence of processes to be

2 https://www.lexico.com/definition/workflow

3 https://dictionary.cambridge.org/dictionary/english/workflow

executed in a precise order, such as from left to right, once the previous process has ended. Understandably, pipelines can be referenced as workflows where processes are executed in an exact order. Workflows are not one-time tasks or a list of tasks to be addressed in order to have a one-time output. Workflows have to do with *repetition*, so in principle, if someone follows the same sequences of processes with the same rules, dependencies, cycles, then they would get the same output. In order to make workflows reusable and reproducible, this sequence of processes, as well as all additional information, must be able to be repeated. Concepts, such as reusability and reproducibility are inextricably linked to workflows.

There is a great variety of workflows in different fields, from business to scientific ones. In business, workflow terminology is widely understood and applied in practice. The idea of business workflows was firstly introduced a hundred years back in the United States by two engineers, Frederick Winslow Taylor and Henry Gannt [43]. Frederick Winslow Taylor was an American mechanical engineer who was interested in industrial efficiency. He was influenced by how the workplace changed after the Industrial Revolution, introducing large factories, with a lot of employees working there. He had ideas about how productivity would have been increased if jobs were optimized and simplified. He proposed matching a worker to a specific type of job that suited the person's skill level and train them to have the maximum efficiency. Taylor authored a book, The Principles of Scientific Management, in which presented his ideas about efficiency. The *Taylorism Principles* that were thoroughly described in the same book [23], still have validity in today's management environment. Henry Gantt on the other hand, was an American mechanical engineer and management consultant who had contributed to the classical management theory by introducing the Gantt chart, the task and bonus system. He created the Gantt Chart in the 1910's which was a bar chart used to visually track tasks. Gantt charts are used until this time, not only for tracking individual tasks, but also scheduled milestones of a project. Taylor's and Gantt's work led to time and motion studies, which measure the time it takes for employees to complete a task or a series of tasks in order to find ways to eliminate redundant or wasteful motion. In that sense, processes were defined in a way where employees should follow to perform their jobs as efficiently as possible. In those times, tasks were executed by humans or machinery that again needed humans to interact with. Therefore, it was a necessity to have tasks to be completed in a specific order and in an efficient way. Concepts from earlier in the days, such as efficiency, mechanics and automation were major features, introduced business workflows as we know them today. As the industry and manufacturing grew, so was the necessity of managers and owners to have ways to figure out the best way of streamlining the work that needed to be done in an efficient way while at the same time ensuring the right people oversaw the right tasks. Nowadays, a business workflow is the definition, execution, and automation of business processes where tasks, information and documents are passed from one person to another for action according to a set of procedural rules[4]. Industries today want to save time and make reproducibility easily in sequences of tasks that need repetition. There are great benefits for using workflows in projects for managing tasks. Currently, organizations from large to smaller ones are familiar with the terminology of workflows and workflow management systems and do business using that approach.

In the current thesis, we are interested in defining terms associated with *scientific computational workflows*. Specifically, a scientific computational workflow is the description of processes for accomplishing a scientific objective, usually expressed in terms of tasks

4 https://www.projectmanager.com/training/define-workflow-process

and their dependencies[5]. Scientific workflows are linked to scientific processes such as modelling, automation of computational experiments, data analysis and data management. Common stages in scientific workflows are acquisition, integration, reduction, visualization, and publication of scientific data. Scientific workflows allow users to describe directed acyclic graphs where the nodes form the computational tasks, and the edges form the dependencies between those nodes [21]. Since we are talking about computational workflows, dependencies are usually data produced and flown by the different tasks. The computational tasks can be anything related to data manipulation, from analysis to simulation [42]. The simplest computational tasks in scientific workflows are scripts that have some input data and produce outputs that might involve visualization and analytical results. In that sense, scientific workflows manage flows of data, whilst running tasks can vary from very large ones to smaller individual parallel or sequential tasks [5]. Scientific computational workflows, in contrast to business workflows, are dedicated to supporting data and compute-intensive scientific experiments. Workflow tasks are organized during design time by the graphical or textual representation of a workflow and orchestrated during runtime according to dataflow and dependencies [28] as specified by the workflow designer. Tasks of a scientific computational workflow could be atomic, or a composition of more, creating a *sub-workflow*. As in [2], a step in a scientific workflow specifies a process or computation to be executed, for example a software program to be executed, a web service to be invoked, an analysis of data, or a simulation of a whole brain. The steps are linked according to the data flow and the dependencies among them. The representation of these computational workflows contains many details required to carry out each analysis step, including the use of specific execution and storage resources in distributed environments.

Scientific computational workflows are also associated with *provenance*[6]. Provenance has traditionally been used to denote the record of ownership of a work of art or an antique, used as a guide to authenticity or quality [20]. More recently, the term has been used in new ways, mostly related to the origin, context, and history of data. Data provenance is defined differently based on the context where it is applied. In data centric areas such as databases, data provenance is defined as the description of the origins of a piece of data and the process by which it arrives at the database. In workflow-centric areas, data provenance is largely regarded as the automatically and systematically captured and recorded information that helps users or computing systems to determine the derivation history of a data product, starting from its original sources and ending at a given repository [3]. Other than data provenance, workflow provenance is also used as a term. Workflow provenance refers to recording of changes in the description of a workflow as well as information during the execution of one. All provenance types are relevant in the reproducibility of scientific workflows and their output.

One of the main similarities of business and scientific workflows is that both intend to create repeatable, reproduced, and replicated procedures. This means that in addition to provenance information, whilst following the same steps, the same output will be provided. Specifically, business workflows are intended to facilitate project management by creating a repeatable procedure in a way where it is highly efficient and reproducible. Scientific workflows introduce ways where scientists can create their workflows, execute them with minimum interactions when errors and restarts of tasks must occur, and have a reproducible way of expected output.

5 https://link.springer.com/referenceworkentry/10.1007/978-0-387-39940-9_1471

6 https://www.lexico.com/definition/provenance

These flows of data can be defined as ***workflows*** with data manipulation steps combined to produce graphs, in which steps are considered as nodes and data as edges. Such data manipulation steps comprise software packaged together with all required dependencies and libraries for different hardware targets and with no configuration needed. Such packaged software is defined in a common and structured way, facilitating reusability and reproducibility, since the workflows themselves and the individual software can be reused as is. [35]

2.1.2 Workflow engines

Workflows required specific engines in order to be executed. The term *workflow execution* refers to the record of performing a sequence of steps using an engine, a system, a human operator, or a combination of them. *Retrospective workflow* [17] is associated with provenance information [2] which entails the details of every executed process together with comprehensive information about input data, the exact workflow recipe, known also as prospective workflow, the execution environment used to derive a specific data product, the data product as well as different association with studies or evaluations. Information is observed by the workflow management system or by the computational task itself [2]. The term *semi-automated workflow execution* can be used when a step in a workflow requires human intervention. By human intervention, we refer to any manual configuration during the workflow execution. In the same context, the term *automated workflow execution* can be used when a sequence of processes in a workflow can be made entirely automated.

Over the past years, execution of scientific workflows was done using scripting languages such as Bash and Python. However, these often lack the necessary flexibility, as workflow execution must be automated and monitored carefully so that it can handle program failures and avoid the unnecessary re-execution of tasks. *Workflow management systems* or *workflow engines* are designed to alleviate these problems by allowing workflows to be expressed formally in their own syntax and are deployed over a computing infrastructure to set up, execute, and monitor them. While there are significant differences between the features of different systems, most of available solutions include components responsible for executing tasks, data management (e.g., input/output data staging), task scheduling and parallelisation, as well as capturing provenance metadata [3]. There are numerous benefits of using workflow management systems [4]. For starters, execution of complicated analysis involving many tools can be automated and executed with just a single command. Also, many workflow management systems provide a graphical interface to the users in order for them to graphically design their workflows. Workflow management systems also support the description of a process in a way that is easy to control and orchestrate. They also provide rapid design, re-design, implementation, as well as re-implementation of the processes [5]. The main feature they provide is the *automation* of procedures previously done manually by developers [20], scientists, or simple users. In addition to automation, workflows can provide the necessary information for scientific reproducibility and sharing of results. They also exploit the explicit representations of computational processes at various levels of abstraction to manage their lifecycle. By providing automation and enabling reproducibility, they can accelerate and transform the scientific analysis process [6]. Scientific workflows are routinely used in many data-driven research disciplines today, often exploiting rich and diverse data resources and parallel and distributed computing platforms [2]. Thus, workflow management systems provide a systematic way of describing the methods needed and provide the interface between domain specialists and

computing infrastructures [4]. There are many commercial and free workflow management systems that use their own format to describe such execution of workflows[7]. Workflow engines can be installed in a variety of different underlying infrastructures, providing a level of abstraction, and ensuring workflow portability. Another important benefit is scalability, meaning that workflow execution can be scaled across a local machine, many nodes, cluster or HPC systems by using the same workflow engines in all infrastructures. In that sense, workflow execution of a locally executed script is more likely to run on High Performance Computing systems or the cloud with only some modifications, as these underlying systems often have specific configuration and authentication requirements that workflow engines can take care of. Thus, workflow engines that can run in different hardware and underlying infrastructures, can provide both scalability and portability.

Although there is a large variety of different workflow systems that can execute and monitor workflows, next we briefly present a small number of them, that share an active developer and user community, frequent releases, and support for HPC systems.

2.1.2.1 Snakemake

Snakemake [18] is a workflow management system that provides a python-like specification language together with an easy-to-use yet elaborate execution environment, to reduce the complexity of defining and executing a workflow. Snakemake follows the GNU Make paradigm, where a workflow is essentially a python script extended by declarative code to define rules that describe how to create output files from input files. Dependencies between the rules are determined automatically, creating a directed acyclic graph (DAG) of jobs that can be automatically parallelized. Commonly, rules consist of a name, input, output files and a shell command or a pure python code. Snakemake therefore offers a definition language that is an extension of Python with syntax to define rules and workflow specific properties [33]. This allows the flexibility of a plain scripting language with a pythonic workflow definition. The Python language is known to be concise yet readable and can appear almost like pseudo-code. The syntactic extensions provided by Snakemake maintain this property for the definition of the workflow. Snakemake's scheduling algorithm can be constrained by priorities, provided cores and customizable resources and it provides generic support for distributed computing for example cluster or batch systems in HPC systems. Hence, a Snakemake workflow scales without modification from single core workstations and multi-core servers to cluster or batch systems. In other words, Snakemake can manage a workflow that is executed in a standalone computer, or a clustered HPC system with the same ease from the user's perspective. Further, Snakemake only executes jobs under specific circumstances; a job is executed only if the output file is target and does not exist yet, or the output file is needed by another executed job and does not exist yet, or input file is newer than the output file, or execution is enforced by the scientist. Finally, Snakemake integrates with the package manager Conda and the container engine Singularity [46] such that defining the software stack becomes part of the workflow itself. As such, tools packaged via Conda or wrapped in a container like Singularity, can be used as workflow steps in Snakemake seamlessly, by only providing the right input to achieve the correct output.

7 https://github.com/meirwah/awesome-workflow-engines

2.1.2.2 Nextflow

Nextflow [30],[41] is a workflow engine designed for the development of data-driven computational workflows using software containers. Nextflow workflows are written using its own custom programming language Domain Specific Language (DSL), an extension of the Groovy programming language. Unlike Snakemake, Nextflow is based on the dataflow programming model that simplifies the definition of distributed parallel workflows by describing the flow of data rather than execution details. A Nextflow pipeline script is made by combining different processes, written in any scripting language. Those processes communicate via Nextflow channels, that are asynchronous FIFO queues that can be used as either process input or output, and are otherwise executed independently. The way the processes interact through their input and output channels determines the workflow execution flow. One of the most important features of Nextflow is that it provides an abstraction between the workflow definition and the underlying execution platform. There are different executors for different target systems (local, Kubernetes, AWS, HPC) that determine the execution parameters for each environment. As a result, it is possible to define a workflow and run it without any modification locally, on the cloud or on an HPC cluster simply by specifying the corresponding target execution system in the configuration file. Nextflow supports Docker [45] and Singularity [46] containerization methods for defining processes. It has also built-in support for Conda that allows the configuration of workflow dependencies using Conda recipes and environment files. Bioconda, which is a very popular tool collection for Bioinformatics, has successfully adopted Nextflow as the engine to orchestrate the execution and monitoring of workflows.

2.1.2.3 Airflow

Apache [37] is an open source platform capable of executing, scheduling, authoring, monitoring and handling events for data analysis pipelines. In Airflow, workflows are divided into one or more tasks and are represented as DAGs in which each task gets executed either parallel or one after another, depending on the dependency's the tasks have. Workflows are written in Python scripts. Scheduling of each task is the responsibility of Apache Airflow, while orchestration of workflows is the responsibility of users writing the scripts. Some of the Apache Airflow advantages are the easy monitoring of workflow orchestration while it is running. Airflow provides all necessary logs, outputs and details for each task that is executed. Apache Airflow provides a very simple and easy to use GUI for users in order for them to check logs, details, task duration and task execution time. A very important feature is that it is compatible with Google offerings, like Cloud composer, GCP, Google BigQuery, Dataproc and Dataflow. Airflow’s approach is more pipeline-oriented than data streaming-oriented. Airflow is not responsible for moving data from one task to another, but tasks exchange metadata. Finally, it offers scalability, since workflows can run locally in a computer, or in a cluster or in an HPC system by changing only the installation part of Airflow and not changing the definition of a workflow. Especially in HPC systems, Airflow is not highly recommended nor popular because there are some restrictions and absence of built-in mechanisms for scaling up and down workers on HPC clusters.

2.1.2.4 Pegasus

Pegasus [6] is funded by The National Science Foundation under Office of Advanced Cyberinfrastructure (OAC) SI2-SSI program and is used for executing data intensive workflows. It handles possible workflow errors when they occur by providing workflow-level endpoints and by retrying part of the workflow or the workflow itself in order to be correctly executed. Pegasus offers the right level of portability with respect to the underlying infrastructure, executing in an easy way workflows that run on top of Amazon, Google Cloud, many HPC clusters as well as different hardware from a single system or across a heterogeneous set of hardware resources. Pegasus can scale in the size of the workflow as well as in the available resources that are needed for the workflow to be executed. Pegasus captures data provenance at the time of execution; provenance information is stored in a database and can be queried with Pegasus-statistics, Pegasus-plot or directly via SQL. One of the most interesting and important features that Pegasus provides is the "on-the-fly" computation of data. If data is not currently available, Pegasus can produce the data on demand if a workflow describes the location of the necessary data and the number of resources that are needed for the computational steps to run. Pegasus is designed to manage workflows executed over potentially distributed data and compute resources. Workflows are represented as DAGs where nodes are the computational steps to be executed and edges are the data flowing between different computational steps. A variety of different scientific fields have already adopted Pegasus for their different scientific workflows, including astronomy, bioinformatics, earthquake science and more, proving that Pegasus is a reliable and active workflow engine.

2.1.2.5 UNICORE

Uniform Interface to Computing Resources (UNICORE) [40] is an open source project under BSD license. It can operate through various operating systems like Linux, UNIX, MacOS and different batch systems such as SLURM, Torque, LSF on top of HPC systems. UNICORE comes with a web portal, a GUI, a command line interface and an API to facilitate users. JSON is used as the workflow description language for defining workflows. From the official documentation[8]: *"The Workflow engine allows to run arbitrarily complex cross-site workflows. It offers a wide range of control constructs and other workflow features such as variables, hold points and more. Execution tasks will be submitted to UNICORE/X servers. The Workflow engine includes a per-workflow file catalogue, allowing powerful and flexible data management during workflow execution. The Workflow engine shares its security features with UNICORE/X and allows flexible user authentication. Using delegation based on tokens, the user only needs to authenticate once to run cross-site workflows."* UNICORE/X service is the central component of a UNICORE installation. It is responsible to accept client requests transmitted by the Gateway, authenticated requests, authorization, and invokes the appropriate service, like the Workflow System. Other than the Workflow System, services include tasks submission and tasks management, storage access and file transfers.

8 https://www.unicore.eu/docstore/workflow-8.0.0/workflow-manual.html

2.1.2.6 Toil

Toil [49] is an open source, portable, scalable pure Python workflow engine that supports pipeline and workflow definitions. Toil can run in a variety of different underlying infrastructures from locally to cloud like AWS and Google Compute Services. There are three different configurable pieces that let the execution and monitoring of a workflow take place. The first is the Job store, that centralizes all files used by different jobs and the details of the processes to run. In a crash or failure, all information is stored there to ensure the minimal repetition of the work during a resume or retry. There are two types of this Job store, the File Job store and a Cloud job store. The file job is for local use only and keeps the workflow information in a directory in the machine where the workflow is launched. The Cloud Job store can be either an AWS Job store which is a backend that supports AWS S3 bucket, or a Google job store that supports a Google Cloud storage. There is also compatibility with batch systems supported by different HPC workload managers like SLURM and Torque. Under these environments, a leader and workers are responsible to coordinate all tasks and files through the centralized Job store. Finally, the Provisioner provides tools set for running Toil workflows in a particular cloud platform.

2.1.3 Workflow authoring

In scientific computational workflows, *definitions* refer to workflow recipes which define and/ or describe in which order and by what dependencies the sequences of processes will be executed. Workflow recipes or else *prospective workflows* can be referred to as the right amount of information gathered that can explain the workflows' sequence in order to be reusable and replicable by others, as well as by the author of the workflows itself. In general, in order to have steps that can be repeated, so to have reproducible workflows [17], it is very important to describe every step of the workflow with clear instructions. Even in the case of semi-automated workflows, where human interaction is needed, it is crucial to describe the human intervention process in a concrete way for the workflow to be reproducible. There are different ways to describe a workflow. In some cases, the description of a workflow is strictly associated with the workflow engine or the workflow management system that will run upon. In other cases, there are some standard ways to describe workflows that are engine independent. For the latter case, we will thoroughly provide more information in the next sections.

Where definitions of workflows are strictly associated with the workflow engines that will run upon, there are two ways of describing workflows. First, scientists can either have GUIs provided by the workflow engines in which they can visualize their scientific workflows, or they can define them textually by composing recipes using standard formats that workflow engines can understand [4]. In GUIs, scientific computational workflows can be designed visually, using block diagrams, links, cycles, branches by simple dragging, dropping and drawing lines and different schemes. Further, workflows can be represented in charts using the conventional symbolism, such as cylinders for storage, rectangular boxes for processes, rhomboids for decisions, parallelograms for data records [35].

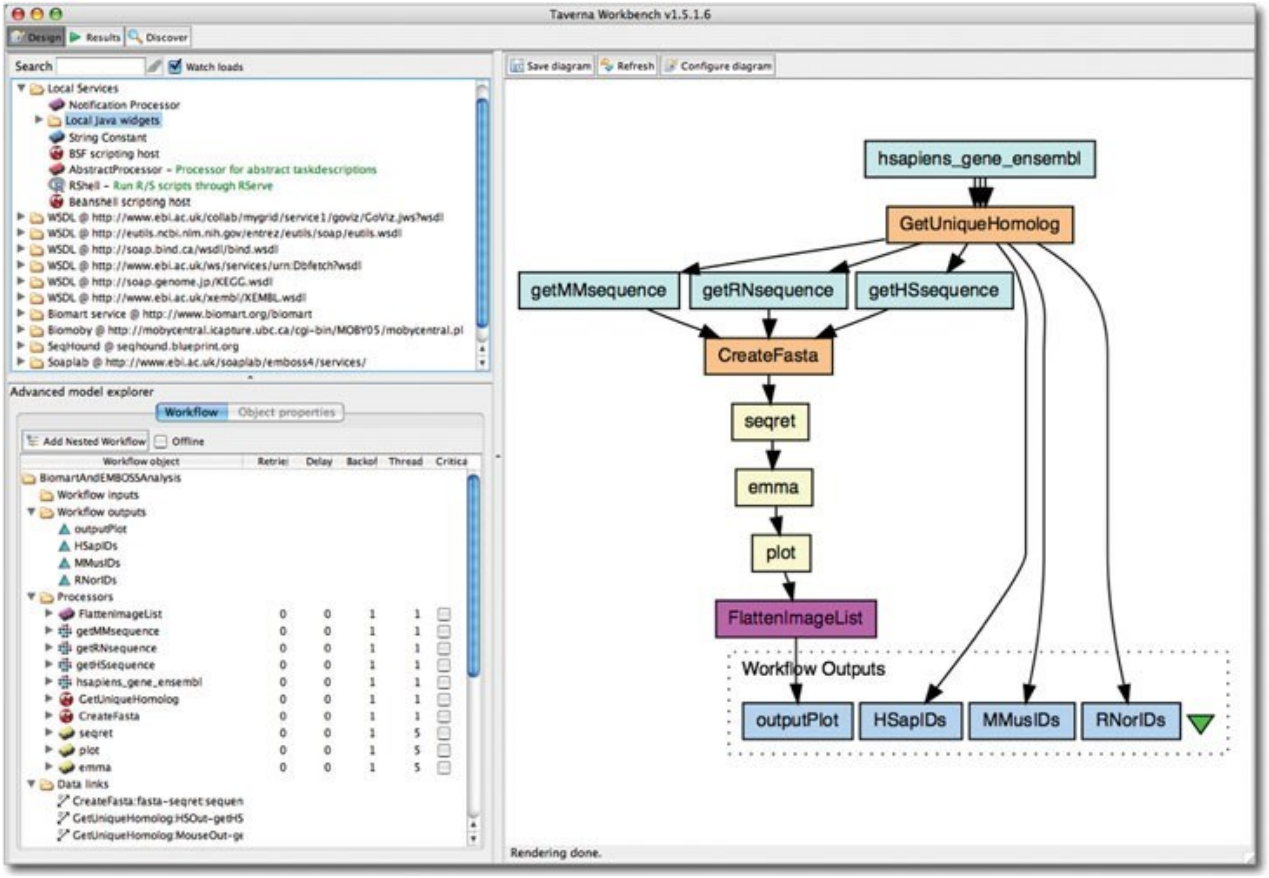


Figure 1: Graphical User Interface (GUI) provided by Taverna workflow management system. In this GUI, users are able to visually describe scientific workflows. The execution is strictly associated to Taverna. [https://launchpad.net/taverna]

As for textual definitions of workflows associated with the workflow engines that run upon, there are different domain specific languages that exist. From the aforementioned list of different workflow engines, Snakemake follows the GNU Make paradigm, where a workflow is essentially a python script extended by declarative code to define rules that describe how to create output files from input files. As a simple example Figure [Figure], we can see a workflow defined via Snakemake. In that same way, definitions of different workflows are associated by different workflow engines previously presented.

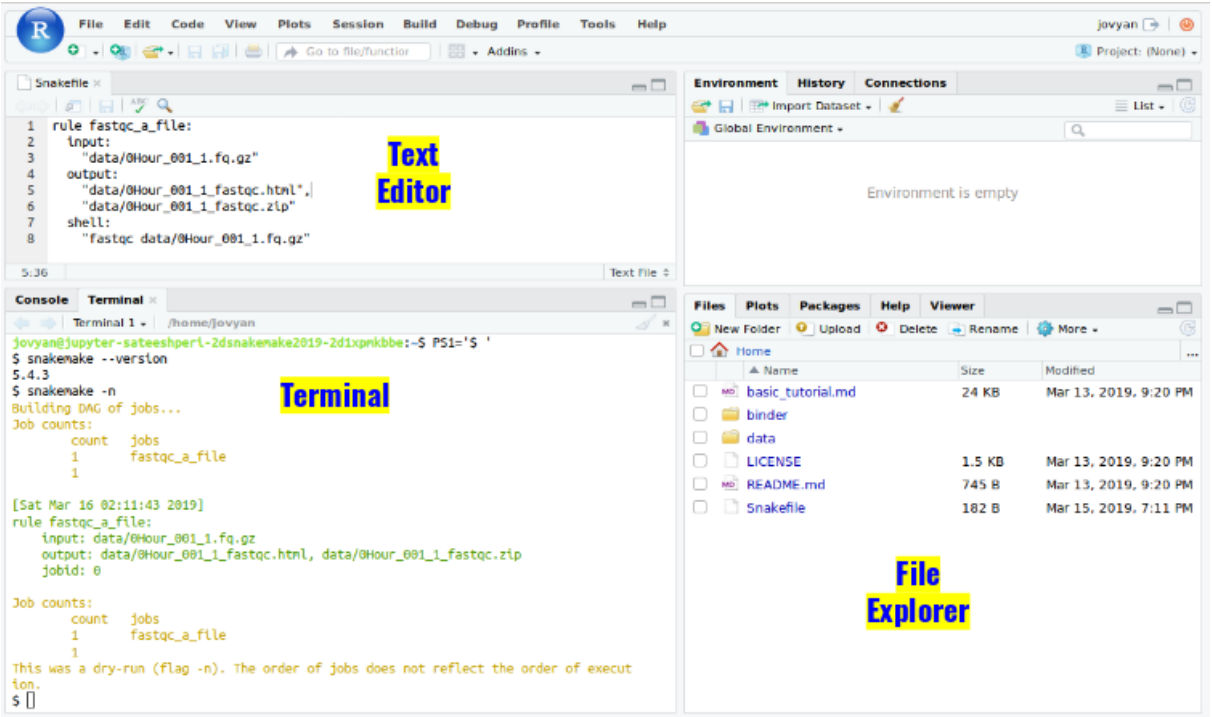


Figure 2: Simple text editor for Snakefile descriptions. Snakefiles are strictly associative with Snakemake executing the different rules. [https://gwu-omics2019.readthedocs.io/en/latest/snakemake_basic.html]

As another example, as we see in Figure [Figure] when users use Nextflow for workflow execution, workflows must be written by its own custom programming language Domain Specific Language (DSL), which is an extension of the Groovy programming language. This means that user will have to experience themselves each time with a specific way of describing workflows depending on the workflow engine that is used in the underlying infrastructure. This often takes quite some time and the learning curve can be exponentially high.

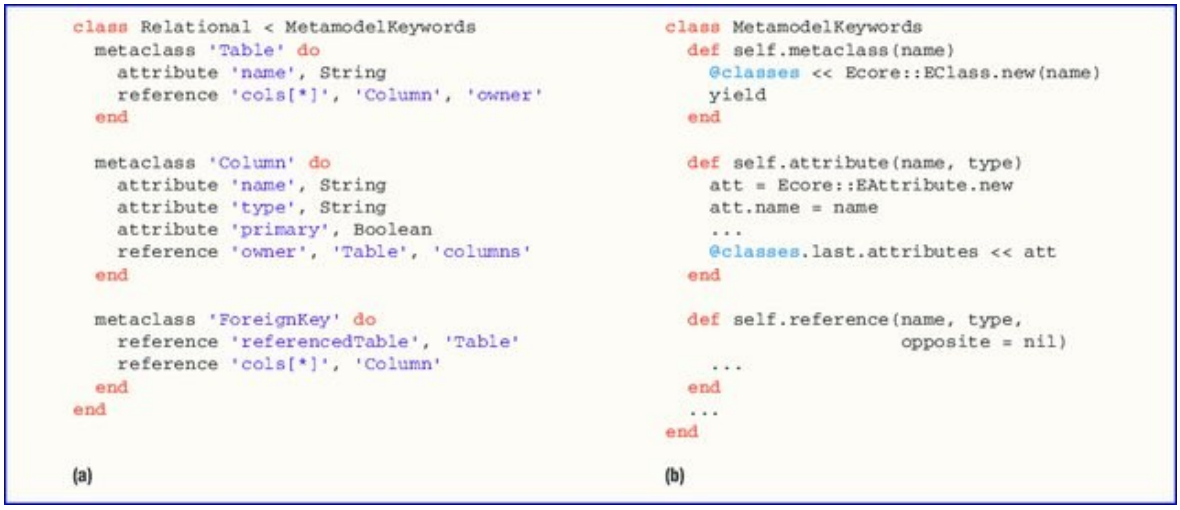


Figure 3: Workflows written in DSL that Nextflow workflow engine is able to understand and execute. [https://www.researchgate.net/publication/266661846_A_Domain-Specific_Language_for_Building_Self-Optimizing_AST_Interpreters/figures?lo=1]

2.2 High Performance Computing

The last couple of decades, PCs have taken an enormous technological leap making their use an essential part of our everyday lives, from education to entertainment. However, they have a limit on what they can offer to their users; the data that can be manipulated, the parallel execution that they can provide while solving a problem, and the speed in which they can generate some valid output. Other than data, the nature of problems that scientists, deal with, have shifted to more complex ones, needing powerful and strong underlying infrastructures [44]. Users often need access to dedicated hardware, such as GPUs to visualize results. There are also cases where software can be executed faster in parallel using OpenMP or MPI technologies. Therefore, in the last decades HPC systems are used by more and more by scientific communities to take advantage of the great capabilities the offer. HPC systems consist of hundreds of thousands of computer servers connected via a network, called nodes, which can work in parallel, boosting processing speed.

HPC refers to extremely high computation power, large storage and powerful infrastructure for solving complex and demanding problems that may need an exceptional amount of power or large capacity for enormous data. HPC systems are associated with domains like climate change, astrophysics, financials, medicine, genome sequencing, and many more. Recently, HPC systems were involved in different ways in providing therapies for COVID-19, supporting the development of vaccines to end the pandemic, or even simulating the virus itself to better understand its genome.

Next, we present some of the key aspects of High Performance Computing systems.

2.2.1 Brief History

In this subsection we take a step back and understand how the concept of High Performance Computing systems came to be. HPC systems are strictly associated with supercomputers [32], but they are not just that. ‘Computer’ back in the early 17th century, was a term used to mean “one who computes". In the 1800’s[9,10] specialists (which were mostly women), performed repetitive calculations to compute navigational tables, tide charts and planetary positions before electronic computers were available. Back in 1922, an estimated 64.000 human computers could forecast the weather for the whole globe by solving differential primitive equations numerically [7]. Also, during the two World Wars, human computing became a profession. Since men joined the army, it was again women who took over these responsibilities. During World War I, women computers were involved in calculating ballistic tables [8], producing map grids and navigation tables [9]. During World War II, women computers examined the nuclear and particle tracks left on photographic emulsions and played an integral role in the Manhattan Project where they were working with different mechanical aids to assist in numerical studies of complex formulas related to nuclear fission [10]. In 1945, the Electronic Numerical IntegrAtor and Computer (ENIAC) was created, a programmable computer designed to compute ballistic tables. ENIAC has a significant role in the end of World War II by deciphering the Nazi’s

9 https://blog.quantinsti.com/journey-computing/

10 https://www.livescience.com/20718-computer-history.html

code. It was women whose job transitioned from analogue calculations by hand, to digital computing at ENIAC [32].

Other honourable mentions of first-generation computers[11] are Standards Eastern Automatic Computer (SEAC) built for the United States National Bureau of Standards and ERA 101 that later renamed to UNIVAC 1101, built for the predecessor to the National Security Agency [32]. It is considered that the terminology supercomputer was firstly introduced in the 1960 where Seymour Cray co-founded the Control Data Corporation [12], [11], which designed computers like the CDC 1604, the first-generation computer with transistors in a period where vacuum tubes [13] were found. They also designed CDC 6600, which was the fastest computer at that time with performance up to three megaFLOPS [14]. From that point in time, a new era of computing systems was introduced, shaking the foundation of the development of large computers with high performance. Seymour Cray after some more experimentation started the Cray Research company. Until today, CRAY is strictly associated with HPC and supercomputers, with several CRAY supercomputers listed in the top 500 ranking.

Some honourable supercomputers of the year 2020, which are accessible to EBRAINS scientists are JUWELS (rank #7) and PIZ DAINT (rank #12).

11 https://www.computerhistory.org/timeline/1953/

- JUWELS (rank #7)

Figure 4: JUWELS Booster Module - located in Forschungszentrum Juelich (FZJ)

- PIZ DAINT (rank #7)

Figure 5: PIZ DAINT CRAY XC50, XEON E5-2690V3 12C 2.6GHZ, ARIES INTERCONNECT, NVIDIA TESLA P100 located in Swiss National Supercomputing Center

2.2.2 Architecture

In this subsection we will provide a more in-depth analysis of High Performance Computing systems. In Europe, HPC systems are divided into Tiers. Tier-0 consists of facilities like European centres with enormous capacity (Petaflops), Tier-1 consists of National Centers and Tier-2 Regional and University centres.

To better understand the difference of magnitude between personal computers and HPC systems, consider the following. PCs can have up to 64 CPU cores, while Fugaku, the top supercomputer[12] currently has 7,630,848 CPUs. Further, a typical PC has a processing power of 3 GigaFLOPS, while a supercomputer like the one mentioned above has a processing power of 537,212 TeraFLOPS.

HPC systems no longer consist of one massive machine. An HPC system can have up to hundreds of cabinets in which the hardware exists, taking up considerable amounts of space for safely them. HPC systems consist of a large number of clusters of nodes [26]. A node has a CPU, memory, and networking to communicate with other nodes. Nodes can vary in focus (e.g., computing nodes, login nodes). Regarding computing nodes, nodes of the same and different clusters inside an HPC system are connected through a network that offers high throughput (up to 100GB/s) as well as high performance. With respect to the login nodes, these are just meant for users to login and submit jobs to a batch system. Users are not supposed to do any other action in the login nodes other than submitting jobs in the batch system. Usually, a workload manager like Slurm and Torque will take care of the submitted jobs to the batch system, will queue the jobs, and will be responsible to provide the output.

A user can access login nodes through Secure SHell (ssh) to submit their tasks. Further, users must for allocate resources on compute nodes. Different supercomputing centres provide a different user experience when intensive jobs must run on computing nodes. Some HPC systems let the users interactively work on computing resources, or directly work on computing nodes by creating yet another terminal.

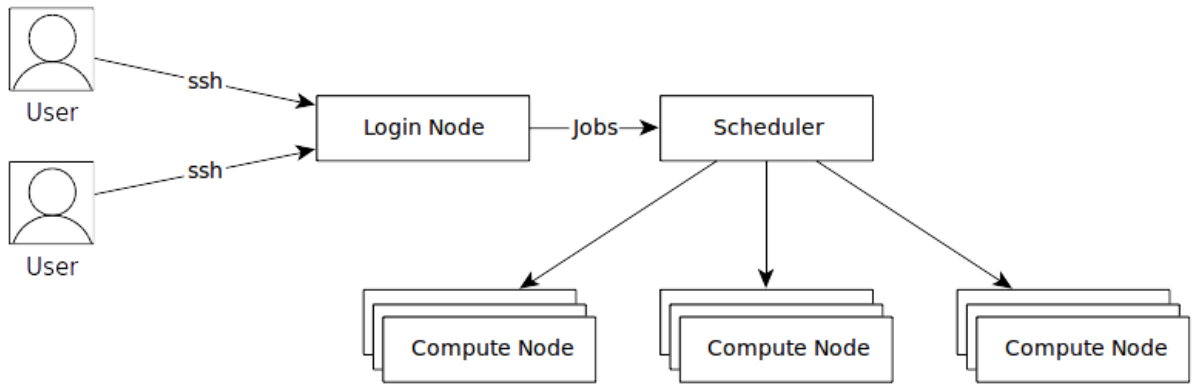


Figure 7: Users entering login nodes of an HPC system via secure shells. The login nodes are only meant to submit jobs in the workload manager / scheduler. Scheduler is responsible to start jobs when resources are available in the different compute nodes. [https://docs.hpc.qmul.ac.uk/intro/]

12 https://www.top500.org/lists/top500/2021/11/

2.2.3 Batch Systems

High Performance Computing systems mostly use batch operating systems for users to interact with the computing nodes, as previously mentioned. In most HPC systems, users will not interact directly with the compute nodes, but they will write their commands off-line in a file, called batch file, which will be submitted by the scheduler into the batch system and will run as soon as the resources are available by the resource manager. Overall, batch systems receive job scripts that users provided, schedules them and manages the execution. Users will only need to specify the minimum number of resources that the job will need in order to run properly, and they will not need to know anything about the batch system and its components overall. With this abstraction of information from the users, batch systems can run thousands of jobs simultaneously.

Batch systems consist of 3 types of nodes, the master nodes, the submit / interactive nodes and the compute nodes. The master nodes are usually responsible for overseeing the creation, submission, execution and finalization of a job. The submit / interactive nodes are the ones in which users manage their workload, meaning that users can monitor their jobs execution by receiving the status. Finally, compute nodes are responsible for executing the jobs. These nodes either communicate with the master nodes or also play the role of the master node themselves.

From the user perspective, they should always use the workload manager system, or else batch system, to submit jobs in an HPC system. They must not use the login nodes to run their batch jobs because there is a risk of exhausting computing resources and this will slow down every other user that is logged in the login nodes, since resources there are shared. Also, users may not have direct access to the compute nodes, which often are not connected to the internet at all. Users cannot interact with the batch system by themselves for scheduling, queuing, or resource management. Therefore it is important that users submit their jobs to a workload manager for fair scheduling and resource management.

2.2.3.1 Schedulers / Resource managers

Overall, cluster batch control systems are used by users to schedule and manage jobs on the system. This is because there are multiple users, each of them can neither interact directly with the compute nodes nor know which compute nodes exist and how much resources (like CPU) overall are currently available for them to use. Two important components exist under a batch system, the scheduler, and the resource manager.

Schedulers' capabilities are sequentially queuing jobs, assigning priorities to them, parallelizing whatever jobs can be parallelized, and overall controlling jobs. When the job is scheduled to run, thus the resources are available, resources are allocated for this job only and no other job can use them in order to also run. Briefly, the scheduler of a batch system is responsible for queuing jobs submitted in a batch system. Different algorithms for scheduling the execution of jobs in a queue exist. On the other hand, resource managers in a batch system must exist in order to control and monitor the execution of a job itself once resources are available and the job is scheduled to be executed. It is possible that multiple tasks in a job are launched across multiple cores, GPUs and different nodes and must be taken care of by the resource manager. A scheduler cannot provide these capabilities, so both schedulers and workload managers need to exist in an HPC system, either under the same software or in different integrated software.

As for the schedulers, different types exist in the batch systems. With respect to the algorithms, First come first served (FCFS), makes the scheduler execute the first job that arrives in the queue. The last job submitted will be added to the bottom of the queue. Once resources for the first job of the queue are available, the job will be executed. In that approach, no other jobs that are of lower priority from the one job on top of the queue will run out of order. This can give many delays in the execution of the queue and sometimes make resources become idle. Also, this is not a fair scheduling approach, since jobs with lower priority but also needing less resources, will have to wait in the queue in order to be executed. Another type of scheduling algorithm which tries to solve the problem described above, is the back-fill algorithm. It comes to solve the problem of executing jobs with lower resources when resources are idle and available but not enough in order for the job with higher priority to be executed. What that means is that a job with lower priorities may start first only if it does not delay any other job with higher priority. In that sense, when resources are not available for larger jobs with higher priorities, they can be allocated for jobs with fewer resource needs and a smaller runtime limit.

Modern schedulers overall provide in even more ways of fairness, when it comes to serving execution of jobs, because already provided solutions like FCFS or back-fill are still not always fair. Some components for fair prioritization are the wall clock, the importance and the size of the job. Fair share also comes with respect to the different resources consumed by whole projects overall. For example, projects that over consume resources for a long period of time, they will get notified for getting lower priorities to their jobs in order to not constantly over consume resources. On the other hand, projects that consume less resources for a long period of time, will get higher priorities for their tasks to be executed

As for the resource managers or else workload managers, they are responsible for providing the low-level functionality to control (start, cancel) and monitor the jobs once they are scheduled for execution, as well as for collecting statistics of all processes running tasks of a job. All statistics of a certain batch job and batch job's steps are aggregated and saved in a database for future reference by individual users or groups. Different types of workload managers run in HPC systems, like Slurm and Torque. Sometimes, schedulers also are also workload managers and vice versa.

2.2.3.1.1 SLURM

Simple Linux Utility for Resource Management (SLURM) [50] is "*an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters*" as stated in the official site[13]. Slurm is responsible for allocating exclusive or no-exclusive resources to HPC systems for some period of time so users can run intense, time and memory consuming jobs. Slurm schedules jobs with respect to their different priorities taking into consideration different job parameters. The Slurm scheduler distributes jobs that are gathered from the login nodes into computing nodes of the HPC system in which the computational tasks run. It also acts as a resource manager since it is responsible for starting, executing and monitoring jobs on allocated resources, which were submitted in the batch system queue by different users. Finally, SLURM is responsible to resolve contentions for resources by managing the queue of jobs that are to be executed by running specific algorithms that take into consideration the wall clock, the size of each job, the resources available and more.

13 https://slurm.schedmd.com/documentation.html

Slurm workload manager consists of one primary centralized manager (slurmctld) which is responsible to monitor resources and work. For high availability and fault tolerance reasons, there is also a backup manager (secondary slurmctld) to take over in case the primar manager fails or it is unable to manage monitoring of works and resources. Every compute node in the HPC system has a Slurm demon (slurmd) that waits for work, executes it, returns status and waits to re-do the same procedure.

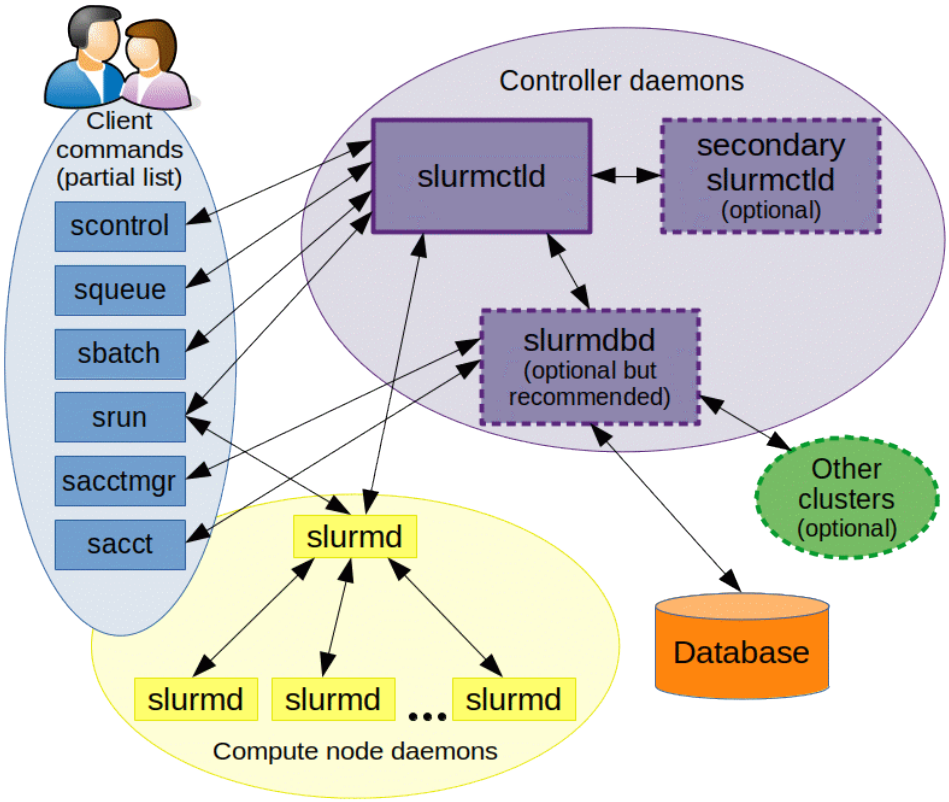


Figure 8: Slurm consists of two components. A centralized manager called controller deamon to monitor resources. And daemon compute nodes that wait execute work, return status and become idle. Although users can interact with either the controller or the compute nodes, it is better to interact with the controller for Slurm to have a better control overall. [https://slurm.schedmd.com/quickstart.html]

From the user perspective, they launch jobs by submitting batch scripts to the scheduler via SLURM. The job enters the queue after the scheduler gives a certain amount of priority to the job. When resources are available and after a fair share, the job is launched to the allocated nodes. As seen in [Figure 8] , users' commands like squeue for reporting status of jobs as well as sbatch for submitting a job script are controlled by the primare Slurm manager. User commands like srun that is used to submit jobs for execution in real time, can either communicate directly with the slurm demon in one of the compute nodes or the primar slurm manager itself. In many HPC systems it is not advisable for users to use srun in order to submit jobs in real time. It is better to use sbatch in order for jobs to be queued for all the reasons we have presented in the previous sections. Optionally, Slurm's plugins can be used for accounting, back-fill scheduler, job prioritization algorithms and more.

2.2.3.1.2 Torque

Tera-scale Open-source Resource and QUEue manager (Torque) [51] is a resource management system that is responsible for providing control over batch jobs and distributed compute nodes. Torque is based in openPBS and freely available to be downloaded and used. It manages batch jobs that users submit in HPC systems and by using a very basic default scheduler, schedules the batch jobs to be executed. This default scheduler will not provide very good utilization for the HPC resources since it is based on a simple scheduling First In First Out (FIFO) algorithm. Overall, this gives no fair share to the computational resources. This is why a specific scheduler needs to be integrated with Torque. Most Torque users choose to use a packaged, advanced scheduler such as Maui or Moab. Torque provides accounting records for batch jobs in a predefined directory. Some of its advantages is the remote submission, launching and managing parallel and serial batch jobs. It also scales up to very large clusters and is currently in use in systems with tens of thousands of nodes.

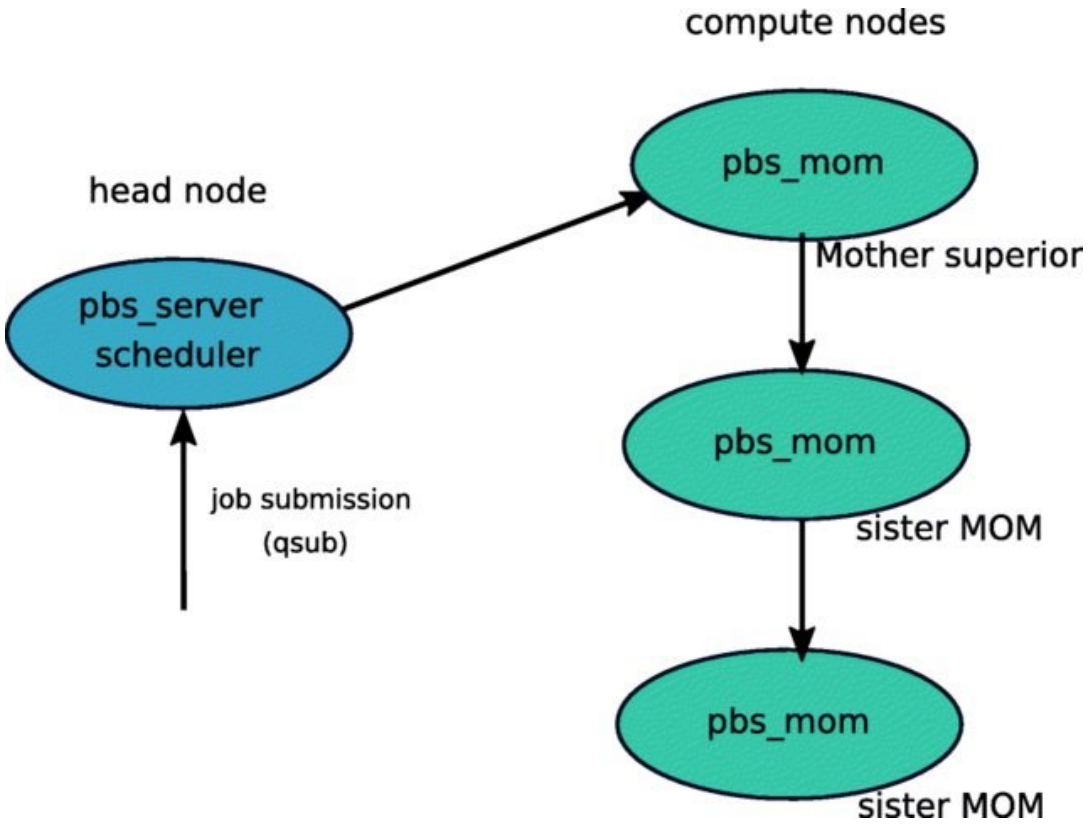


Figure 9: TORQUE consists of a head node and compute nodes. pbs_server scheduler is running in the head node while pbs_mom are daemons running on the compute nodes[19]

Diving into the architecture of Torque, a Torque cluster consists of one head node and many compute nodes. The head node runs the server daemon and the compute nodes run the client daemon. The head node also runs a scheduler daemon. The scheduler interacts with the server to make policy decisions for resource usage and allocate nodes to jobs. Users' commands for submitting and managing jobs can be installed on any host. Users

submit jobs to the host by using the qsub command. When the server receives a new job, it informs the scheduler. When the scheduler finds available nodes and thus resources for the job, it sends instructions to run the job with the node list to the server in the head node. Then, the server sends the new job to the first node in the node list and instructs it to launch the job [19].

2.2.4 Message Performance Interface

As previously described, from an architectural point of view, HPC consist of a large number of clusters that each consist of nodes with CPUs and memory. These computing nodes take care of the simulations running that need an enormous amount of memory that only one node alone could not handle. They also take care of how faster the simulations will run by combining more processors of different nodes in order to get the output. For those reasons, there is a need to access more processors and more memory than is feasible to find in one node alone. Message passing has proven to be very useful for arranging computations on multiple nodes. Architecturally, each node has its own data that other nodes do not have access to. Only when there is a need to exchange data one node can inform the rest of the nodes by sending or receiving messages to and from the others. Message Performance Interface is a standard library for message passing between different nodes in the same cluster, as well as in different clusters. It is based on open standards and is the most widely used interface for that matter.

With respect to HPC systems, MPI is a standard specification for parallel computing architectures allowing message passing between various nodes and clusters. In other words, MPI standard is an API for processes that need to send, wait or receive messages. It is already installed and ready to use in large HPC systems, but it can also be installed and run in local machines like desktop or personal computers. Every small or larger cluster of nodes that deal with data analysis and data simulation that uses many nodes simultaneously, uses MPI. MPI method is designed for high performance, portability and scalability and thus is the most common protocol used in HPC systems. MPI allows writing portable parallel programs for all types of parallel systems from small, shared memory nodes to large ones. There is a vast variety of MPI implementations globally known and used by different HPC systems like Open MPI and MPICH.

Some of the advantages of MPI comes with the fact that the MPI interface was designed and implemented with good performance as a primary goal. In its core development, MPI standard will take advantage of the fastest network transport currently available, without developers implementing the different interfaces and protocols that exist in different cases. MPI overall gives a high-level way of programming since all network alleviations and configurations are taken care of by itself.

2.2.5 Modules

HPC systems typically have a large number of software already installed along with their different version numbers. There are times when users would like to package software with different versions into one specific environment, for debugging, testing, or deployment reasons. Combinations of different software versions simultaneously installed in the user's HPC environment could lead to misconceptions, conflicts and misconfigurations. For users of the HPC systems, it would be important and critical if they could save that environment configured with all the software versions packaged and easily switch at any point in time.

The Environment Modules project implemented Modules for the purposes of dynamically managing environments inside HPC systems. Prior to that, editing the bashrc file was the only way of managing software in a system. This of course was an error prone procedure leading not only to errors but also to misconfigurations and misalignments between different systems that needed the same software, but also for different versions that might not work with each other.

Module is a user interface for dynamic modification of the HPC environment which allows easy changes between shell HPC environments by initializing, modifying and unsetting environment variables. In that way, taking care of software and different versions of it can be simplified in a precise and controlled manner. Finally, the user can save the created environment for later use by saving the current state of the environment with all the software and versions in a modulefile collection. In that way, modulefiles make easier transactions between production, development, and debugging environments.

2.2.6 Storage

There are various kinds of storage associated with HPC systems. Near the compute nodes usually exist file systems capable of high throughput and low latency, called hot storages. These hot storages are capable of good performance when it comes to Input/Output operations. They overall give high performance but have a low amount of capacity and they will not supply backing up policies for data and with low amount of capacity. All data spaces under this hot storage are reachable from both HPC login nodes as well as computing nodes. When simulations, analysis and data manipulation tasks in general, are executed inside an HPC system, they will provide some output in one of the existing hot storages. Since, hot storages are not meant for actually storing data, users will have to move data to another location for archiving it, like the warm or cold storages.

Other storages like the warm ones, are usually located outside of the HPC systems and are capable of archiving and backing up data that will not change very often. These storages, like the Swift Object storage, often store objects, have enormous capacity with high latency but minimum throughput when there is time to move data back to the HPC systems. Last but not least, cold storages exist which are usually used for archiving cold data in a magnetic tape. Exchanges of data between cold storages and every other storage that exist is time consuming since they do not provide high throughput and thus making them not the best primary storages when it comes to running data intensive jobs in HPC systems.

With respect to transferring data from different hot storages to other hot storages inside an HPC system, there are known ways for users to transfer it. Such ways are scp with authentication, rsync for synchronizing data across the internet where only differences are sent between source and destination as well as sftp connections on remote hosts that can transfer files in both directions. With respect to warm storages, usually these are graphically distributed storages providing high availability, that offer REST API methods of getting or storing data there. Finally, for the cold magnetic storages backup methods and specific libraries exist in order for data to be archived periodically in those storages.

2.2.6.1 Hot storages near HPC systems

The hot storages refer to POSIX - like file systems existing near compute nodes in an HPC system. Extreme bandwidths, high semantics (POSIX), high concurrency as well as high throughput are some of the features that these hot storages have. Term data locality refers

to the fact where computation is shifted to where data is stored, rather than transferring chunks of enormous data to where the computation will take place. Data locality arises from the usage of these hot storages which are geographically located near the computing nodes. Jobs and processes run into the same computing node use the hot storages as shared memory for inter communication too. The most common hot storages are file systems like Lustre, General Parallel File System (GPFS) and Network File System (NFS).

Lustre file system[14] is an open source parallel file system that supports a lot of requirements of HPC environments and is commonly used for large scale cluster computing. It is designed to offer scalability, high-performance and high-availability. Some of the specifications are being POSIX compliant and supporting hundreds of petabytes of data storage and hundreds of gigabytes per second in simultaneous aggregate throughput. POSIX file systems have POSIX-type libraries for access that also give good performance. Two major components are Metadata Server (MDS) which stores metadata such as filenames, access permissions and directories and the Object Storage Servers (OSS) which are the locations where data is actually stored. One single file can be stored on several OSSs making high availability and fault tolerance two of the benefits offered by Lustre.

GPFS is a high performance clustered file system developed by IBM. Many HPC systems are using GPFS type for managing files. As specifications we can identify that GPFS provides high-speed file access that can be done simultaneously by applications executing on multiple nodes of HPC clusters.

Many HPC systems also use Network File System (NFS) to have storage systems accessible to all computing and head nodes. But some HPC systems also have dedicated local storage for each compute node. This is to help offload I/O from major storage drives when users are running jobs with intensive I/O. [31]

These hot storages consist of file systems with different types, quota, expiration days, backup policies, access speed and capacity. Typical examples of those file systems are $SCRATCH, $HOME and $PROJECT which are based on either the LUSTRE type file system or the NFS. Some of them do not back up data automatically but instead purge the file system like the $SCRATCH which is a very fast, low latency, high throughput file system. Data transfers can be up to 300GB/s in that file system. In general, only copies of data must exist there, and the original ones must be placed in a long-term repository because typically this filesystem automatically deletes chunks of data after some period of time. These long-term repositories can be either the warm or cold storages that will be presented later on. Data must live in a more concrete environment where backup policies exist, like the $PROJECT filesystem which is usually laying on the GPFS file system, which provides backup of data and easy recovery from problems.

It is important that users make use of these hot storages that exist near the computing nodes, as well as clean up their temporary files from file systems that get purged, when running jobs in an HPC system.

14 https://www.lustre.org/getting-started-with-lustre/

2.2.6.2 Cold storages

Cold storages usually refer to magnetic tape storage software. In this kind of storage, magnetic tape is used as a recording media to store data. This kind of storage is mainly used for archiving cold data that are immutable and secure but at the same time very difficult to retrieve. Tapes are used for periodically backing up data as well as archiving long-term data. High Performance Computing systems have procedures for periodically storing data in tapes from different repositories for backing up policies. Tapes are not used as the primary storage for everyday use because of the low performance they give, since having data to / from magnetic tapes from / to other storages means high costs in terms of money as well as in size.

2.2.6.3 Warm storages

In order to minimize the gap between hot and cold storages, warm tier storages come to the frontline. These object storages cost lower than the cold (tape) storages and often provide great capacity, scalability and reliability. Storing long-term cold immutable data in a cost-effective and efficient way is much cheaper than having data stored in magnetic tapes. Object storage is a data storage architecture in which data is managed like objects inside containers, in contrast with file systems where data is managed as file hierarchies as well as block storages in which data is managed as blocks within sectors and tracks. Each object in a container typically includes the data itself, a variety of metadata and a unique identifier. Usually, objects in containers can be retrieved by REST API requests.

As far as industry is concerned, Amazon Simple Storage Service (S3) is an object storage service that offers scalability, data availability and performance to its customers. S3 is used for storing structure as well as unstructured data, making it a viable choice for certain data lakes.

As far as HPC systems are concerned, there is a vast variety of object storages like Ceph, Swift, OpenIO and MinIO that are used. Ceph [53] is an open source distributed object storage that has an associated file system used in distributing computing. Ceph has a worldwide large and active community for support and development efforts. Under this community members of major HPC systems and storage vendors are also participating.

OpenIO[15] is a software defined object storage that supports S3 and it is also optimized for HPC systems. In an architectural point of view, OpenIO stored objects in a flat structure within a distributed directory with indirections, making query paths to be independent of the number of nodes and also leaving performance unaffected by the growth of capacity. This will make archived data visible giving maximum parallelism on requests for immutable data.

Finally, OpenStack Object storage, Swift, offers cloud storage software for storing and retrieving data. From the official documentation[16], "*Swift is used for redundant, scalable data storage using clusters of standardized servers to store petabytes of accessible data. It is a long-term storage system for large amounts of static data which can be retrieved and updated. Object Storage uses a distributed architecture with no central point of control, providing greater scalability, redundancy, and permanence.*" It is ideal for cost

15 https://www.openio.io/

16 https://docs.openstack.org/swift/pike/admin/objectstorage-intro.html

effective scale out storages with the concept of containers and objects and provides a fully distributed API-accessible platform. Swift can be integrated to applications directly or indirectly as backup, archiving and data retention.

2.2.7 Containerization

Executable code can be deployed to the HPC systems, either as self-contained applications to be launched by service or user-workloads, or as libraries for providing functionality to other HPC applications. They can be packaged in the form of natively compiled modules appropriate for the HPC system's runtime environment or containerised in one of several HPC appropriate container technologies. Encapsulating an application into a container image can be easier than providing all dependencies for the application via operating system packages.

In general, there is a great variety of containerization methods that can be used in all HPC systems. Some of them are Sarus [47], Singularity [46] and Shifter which a developer can use in High Performance Computing (HPC) systems.

The most frequently used tool is Sarus, due to security-oriented methods on HPC systems, the compatibility with Open Container Initiative (OCI) standards and the compatibility with Slurm workload manager, that is usually used in HPC systems. Even though Docker is a most known containerization tool that is globally used in a lot of cases, it is not the preferred tool to be used under an HPC system because of some security concerns. The main concerns have to do with the fact that Docker in order to run containers requires root privileges, but HPC systems are by default multi-user environments where users have restricted access to their own data. Another problem that makes Docker not the primal candidate is the lack of support with the scheduling workload manager like Slurm. Finally, Docker in an HPC system offers a high level of isolation in terms of namespace and filesystem that makes it unnecessary. In HPC systems usually users want easy access to the host file system and most of the work needs to run as the current user.

Singularity is also a well-known containerization method under HPC systems developed by Lawrence Berkeley National Lab. Singularity as well as Sarus provides the correct amount of security on HPC systems and are both compatible with the Open Container Initiative (OCI) standards. This means that both can pull images from registries that are OCI compatible. They both can import and convert images adopting the OCI Image Format. With Sarus and Singularity, developers can run images in HPC systems that include libraries or services that will run directly to HPC environments. Specifically, Singularity images are files (SIF) saved in the filesystem, that the user can simply run. Singularity containers run in user space, which makes the permissions of the user identical inside and outside the container and eliminates the security concerns that arise when using Docker. Additionally, the containers automatically have access to the host filesystem, since $HOME, $PWD and $TEMP are always mounted. Singularity is ideal for HPC systems [19], as it favors integration rather than isolation, while still preserving security restrictions on the container, and providing reproducible images. While Docker is not supported by different HPC systems, Singularity is compatible with all Docker images and allows running Docker containers natively on those HPC environments. Singularity can pull images from DockerHub as well as Singularity Hub[17]. This is the containerization way that

17 https://singularity-hub.org//

most HPC systems use to run Docker containers that do not need root privileges in HPC systems.

3. EBRAINS RESEARCH INFRASTRUCTURE

3.1 European Research Infrastructure

As stated in [52] “Research Infrastructures (RIs) are facilities, resources or services that constitute large sets of research equipment or instruments and represent or complement knowledge resources such as collections, archives and databases.” They are used by a large majority of research communities to conduct research and foster innovation. Research Infrastructures supply the necessary data, services and tools to scientists to explore and lead to state-of-the-art inventions, innovations, cutting edge technologies in a variety of scientific fields, from health, medical to neuroscience, from bioinformatics to marine research and more18.

Research Infrastructures can be single-sited, distributed, or virtual by enabling services electronically. They may consist of instruments, archives, scientific data, computing systems and communication networks. They often require a structured information system for data management and for enabling information and communication [52]. They support top-level research and can be organised at the national and regional level, at European Union Member State, European and global level.

Across Europe, there is a great number of research infrastructures that can be used by scientists to make their scientific innovations happen. To avoid duplication of different research infrastructures in the European area, European Strategy Forum on Research Infrastructure (ESFRI) was established in early 2002. Currently, the policy on Research Infrastructures mainly involves the activities of ESFRI, including ESFRI Roadmap, projects and landmarks. Different working groups coordinate the different projects that exist, monitoring and assessing the implementation of existing research infrastructures related to the different categories.

Specifically, in the context of Health and Food, there are thirteen (13) research infrastructures associated with Life Science currently providing cutting edge technology for Healthcare. Namely, Biobanking and BioMolecular Resources Research Infrastructure (BBMRI-ERIC)19 is a gateway for access to biobanks and biomolecular resources for health research. European Advanced Translational Research Infrastructure in Medicine (EATRIS)20 is a new development pathway for translating novel biological insights into effective solutions. European Clinical Research Infrastructure Network (ECRIN)21 is a network for multinational, high-quality, clinical trials for top-level medical research. ELIXIR22 is a sustainable infrastructure for interoperability of public biological and biomedical data resources. European Marine Biological Resource Centre (EMBRC)23 is a world-class platform for fundamental and applied research on marine bioresources and marine ecosystems. European Infrastructure for Multi-scale Plant Phenomics and Simulation (EMPHASIS)24 is a multi-scale phenotyping platform for food security in

18 https://www.s4d4c.eu/topic/4-3-3-european-scientific-infrastructures-and-organisations/

19 https://www.bbmri-eric.eu/

20 https://eatris.eu/

21 https://ecrin.org/

22 elixir-europe.org

23 https://www.embrc.eu/

24 https://emphasis.plant-phenotyping.eu/

different agro-climatic scenarios. European Research Infrastructure on Highly Pathogenic Agents (ERIHNA)25 is the a pan-European distributed Infrastructure dedicated to study high-consequence emerging and re-emerging pathogens. European Infrastructure of Open Screening Platforms for Chemical Biology (EU-OPENSCREEN)26 is the high-throughput screening platforms and chemistry resources for Life Sciences. European Research Infrastructure for Imaging Technologies in Biological and Biomedical Sciences (Euro-Bioimaging)27 is the large-scale open physical user access to state-of-the-art biological and biomedical imaging technologies. INFRAFRONTIER28 is a European Research Infrastructure for the generation, phenotyping, archiving and distribution of mouse disease models to unravel the role of gene function in human health and disease. Integrated Structural Biology Infrastructure (INSTRUCT ERIC)29 is a peer-reviewed access to a broad range of technology, expertise and training in structural biology. Infrastructure for System Biology Europe (ISBE)30 is a coordination effort to interconnect the best experimental and modelling facilities for Systems Biology. Microbial Resource Research Infrastructure (MIRRI)31 is a coordinated platform to manage microbial resources to support research in biotechnology. In the context of Data and Computing research infrastructure, Partnership for Advanced Computing in Europe (PRACE)32 is the top level of the European High Performance Computing ecosystem.

In the context of brain research in Healthcare, European Brain ReseArch InfrastructureS (EBRAINS)33 was accepted in early 2021 as an ESFRI European research infrastructure. EBRAINS is a digital research infrastructure built by Human Brain Project (HBP) and is filling a gap in the landscape of different research infrastructures in Health and Food by providing research related to Brain. EBRAINS as an open research infrastructure will supply European scientists with FAIR data, services, tools and technologies in order to conduct their research and at the same time will connect EBRAINS with other research infrastructures to achieve Open and Fair science overall.

3.2 Introducing EBRAINS RI

European Brain ReseArch INfrastructureS (EBRAINS) is a digital research infrastructure built by Human Brain Project (HBP). The Human Brain Project (HBP)34 is one of the three Future and Emerging Technology (FET) Flagship projects. EBRAINS started in 2013 and it is one of the largest brain research projects in the world. A lot of individual scientists, engineers, and groups like research centres and teaching hospitals throughout Europe have joined forces for better understanding the human brain, with the help of the EBRAINS

25 https://www.erinha.eu/

26 https://www.eu-openscreen.eu/

27 https://www.eurobioimaging.eu/

28 https://www.infrafrontier.eu/

29 https://instruct-eric.eu/

30 https://cordis.europa.eu/project/id/312455

31 https://www.mirri.org/

32 https://prace-ri.eu/

33 https://ebrains.eu/

34 https://www.humanbrainproject.eu/en/

Research Infrastructure. EBRAINS aims to serve brain research, brain medicine and development in Artificial Intelligence (AI), computing and data science. EBRAINS is powering a new era in Brain research. It gathers an extensive range of data and tools for brain related research. Its ambition is to provide the scientific community with an open, state of the art, environment that fosters collaborative brain science, while opening the way to ground breaking discovery, and securing Europe's leading position in the dynamically growing field of multidisciplinary brain research and its exploitation.

The EBRAINS infrastructure is shaped by the principle of co-design, which means that (a) the needs of the scientists serve as the basis for the development of tools and services, and (b) the insight and expertise of scientists flow into the conception and realisation of the infrastructure. EBRAINS takes great care to ensure that data, models, and software available through EBRAINS conform to the European Union's high ethical standards and respects the FAIR[35] principles, thus providing Openness to the EBRAINS researchers. EBRAINS shares data, models as well as tools that adhere to the FAIR [54],[27] data principles for scientific data management. Thus data, models and tools under the EBRAINS RI meet the four foundational principles of Findability, Accessibility, Interoperability and Reproducibility. In that way FAIR-ness is provided to scientific communities and individual scientists and researchers of the EBRAINS. Currently, no research infrastructure for brain activities is available in Europe, which makes EBRAINS unique and important. EBRAINS is a research platform that can be used by European countries and it is extending also to countries outside Europe. It provides tools and services that each user can use while composing complex workflows.

3.3 EBRAINS Architecture

In this subsection we will depict what EBRAINS looks like from an architectural point of view, after we divide it into three concrete layers. First in the bottom layer, which is focused on the powerful underlying infrastructure, FENIX ICEI supplies EBRAINS with High Performance Computing services like scalable and interactive systems, virtual machines and different types of data repositories, in order for scientists to use them for executing their analysis and simulation experiments providing valuable outputs. The middle layer presents some of the complementary EBRAINS services which can be used by the scientists of EBRAINS for easing their everyday work. In the upper layer, we will depict show some tools and services that can be combined in order for scientists to facilitate their scientific work. Other than the tools and services, scientists have a plethora of FAIR data which can be used in the different experiments. Also, a collaborative platform is available and can be used for collaboration between different teams, for sharing code via Jupyter notebooks and for sharing data between your team members as well as for knowledge exchange via dedicated wiki pages.

35 Findable, Accessible, Interoperable, Reproducible

3.3.1 Bottom Layer: Fenix Research Infrastructure

In this subsection we will present the bottom layer of the EBRAINS architecture. Partnership for Advanced Computing in Europe (PRACE), teamed up with ICEI Project and Fenix Research Infrastructure to deliver advanced computing services to Europe. The Interactive Computing E-Infrastructure (ICEI)[36] is funded by the European commission under the framework partnership agreement of the Human Brain Project (HPB). In the project, five (5) leading European supercomputing centres are working together to develop a set of e-infrastructure services that will be federated to form the FENIX Infrastructure. The abovementioned centres consist of:

- BSC (Barcelona Supercomputing Centre - Spain)
- CEA (Commissariat à l'Energie Atomique - Franche)
- CINECA (Consorzio Interuniversitario del Nord Est italiano per il Calcolo Automatico - Italy)
- CSCS (Centro Svizzero di Calcolo Scientifico/Swiss National Supercomputing Centre - Switzerland)
- JSC (Jülich Supercomputing Centre - Germany)

All 5 sites agreed to align their services to facilitate the creation of the FENIX Infrastructure which supplies EBRAINS with very powerful underlying infrastructure services. Researchers from or associated with the Human Brain Project are the prime users of the e-Infrastructure.

Under this subsection we will briefly list the underlying services that are provided for research communities and users of FENIX research infrastructure, extended also to EBRAINS.

In the next subsections, a visual representation sketching the current status of the powerful underlying infrastructure will be presented as well as a more in depth analysis of the different FENIX services, namely Scalable Computing, Interactive Computing, Active Data Repository, Archival Data Repository, Virtual Machines.

3.2.1.1. Representation

There is a large number of services that FENIX offers with Scalable Computing services, Interactive Computing services, Archival Data Repositories, Active Data Repositories and Virtual Machines being the most important ones. A representation of the bottom layer in the EBRAINS architecture is presented below in the image below:

36 https://cordis.europa.eu/project/id/800858

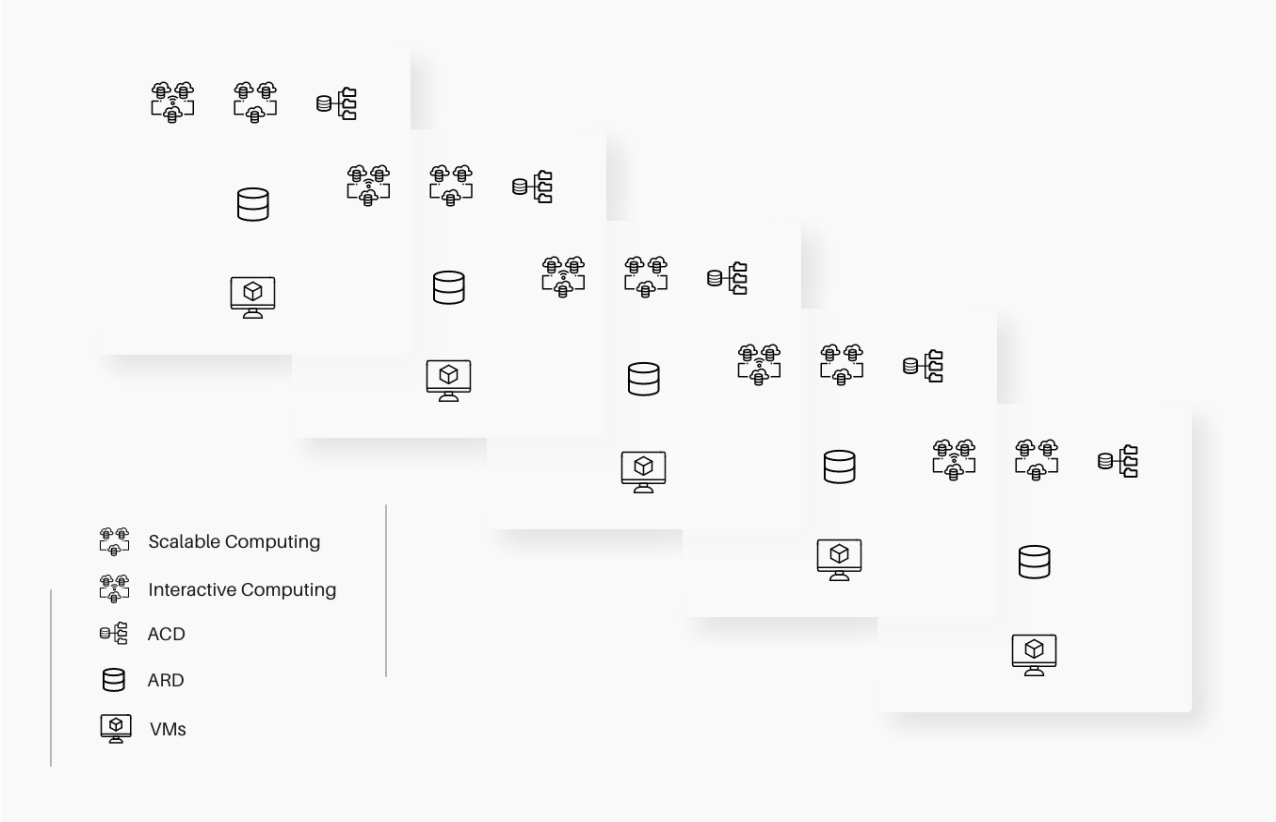


Figure 10: All 5 FENIX sites consist of services related to Scalable, Interactive computing, Active Data Repositories (ACD), Archival Data Repositories (ARD) as well as Virtual Machines (VM).

From the above representation, some more abstractions of the bottom layer can be examined to make their use easier in a scalable and fault tolerant way. Firstly, Scalable Computing Services, Interactive Computing Services and Active Data Repositories are closely combined together. This sub- layer can be associated with HPC systems where scalable and interactive services are delivered to the users for high intense and time consuming jobs. Active data repositories are close to these powerful computational resources and offer good performance and high bandwidth.

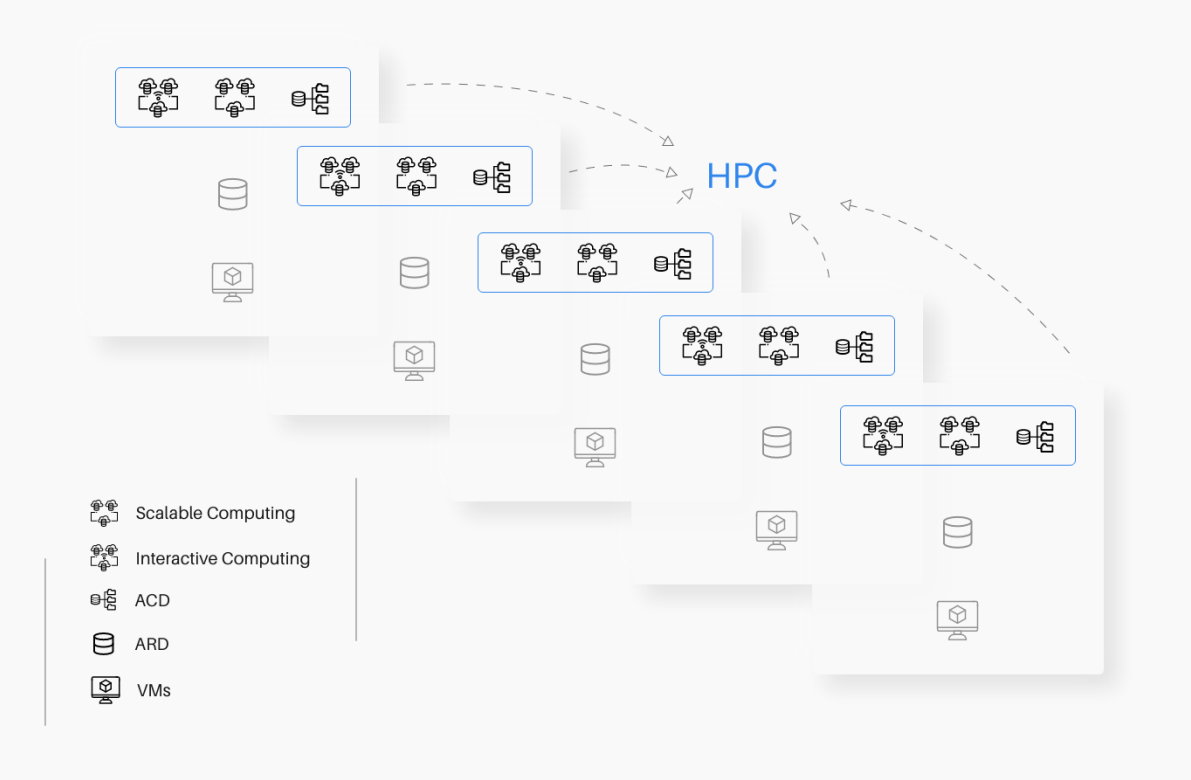


Figure 11: High Performance Computing systems consist of services related to Scalable and Interactive computing as well as Active Data Repositories.

As far as the Archival Data Repositories, an abstraction of object storages can be made possible in order to have federated object storages all combined for archiving long term, cold data providing data locality with different HPC systems as well as scalability and fault tolerance. Transfers of bulk data can be easily made by the API functionalities that the Object storages offer.

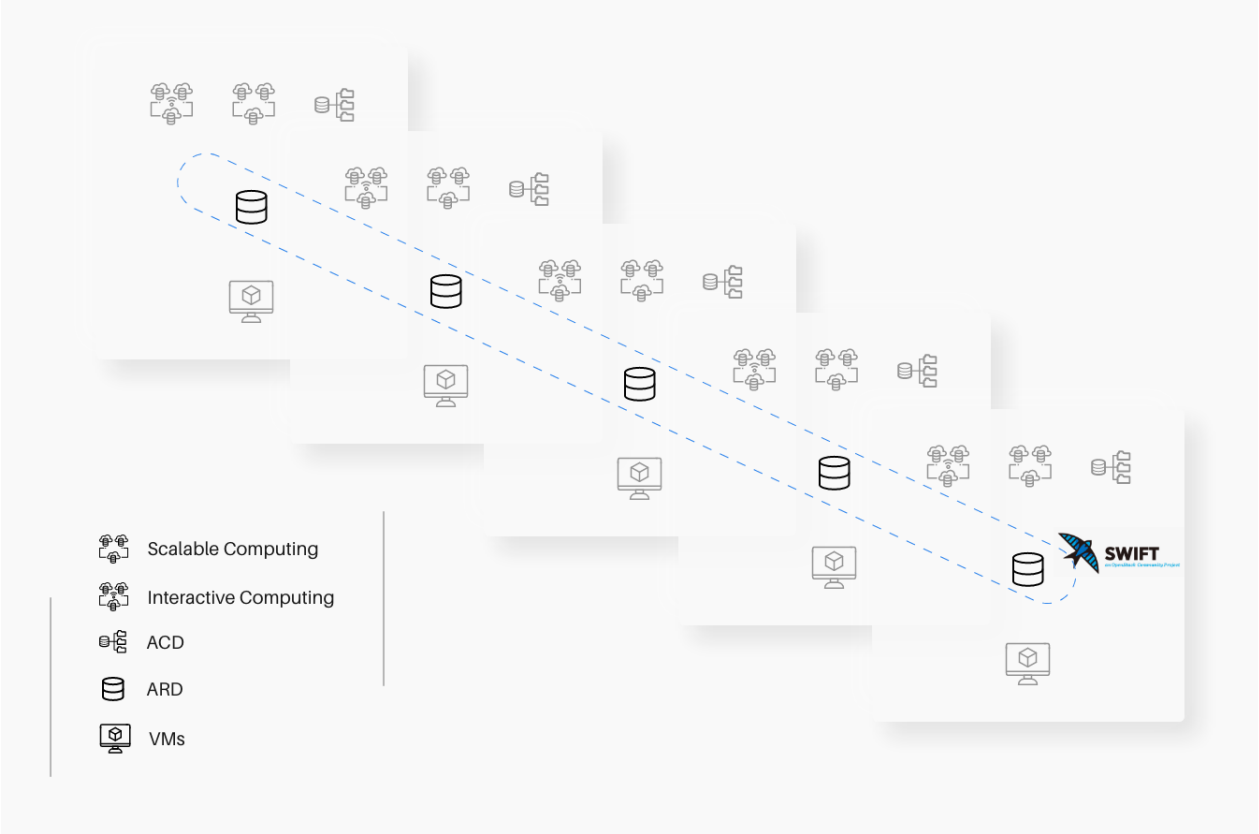


Figure 12: Swift Object Storage is a federated, scalable, reliable cloud storage for storing and archiving unstructured data.

Last but not least, we can abstract a layer of Virtual Machines services by adding OpenShift as an upper layer. OpenShift is used as a Container Orchestration Platform and is an open-source cloud development Platform as a service (PaaS), which enables the developers to develop and deploy applications on cloud infrastructure. It leverages the Kubernetes concept of pods which are one or more containers deployed in one host. OpenShift offers load balancing methods and configuration of applications in a way that under exceptional circumstances like high CPU, or high memory usage or even many requests, the application load can be distributed in more than one pod. Usually, containers hosting long running services run on top of OpenShift due to the added-values it offers.

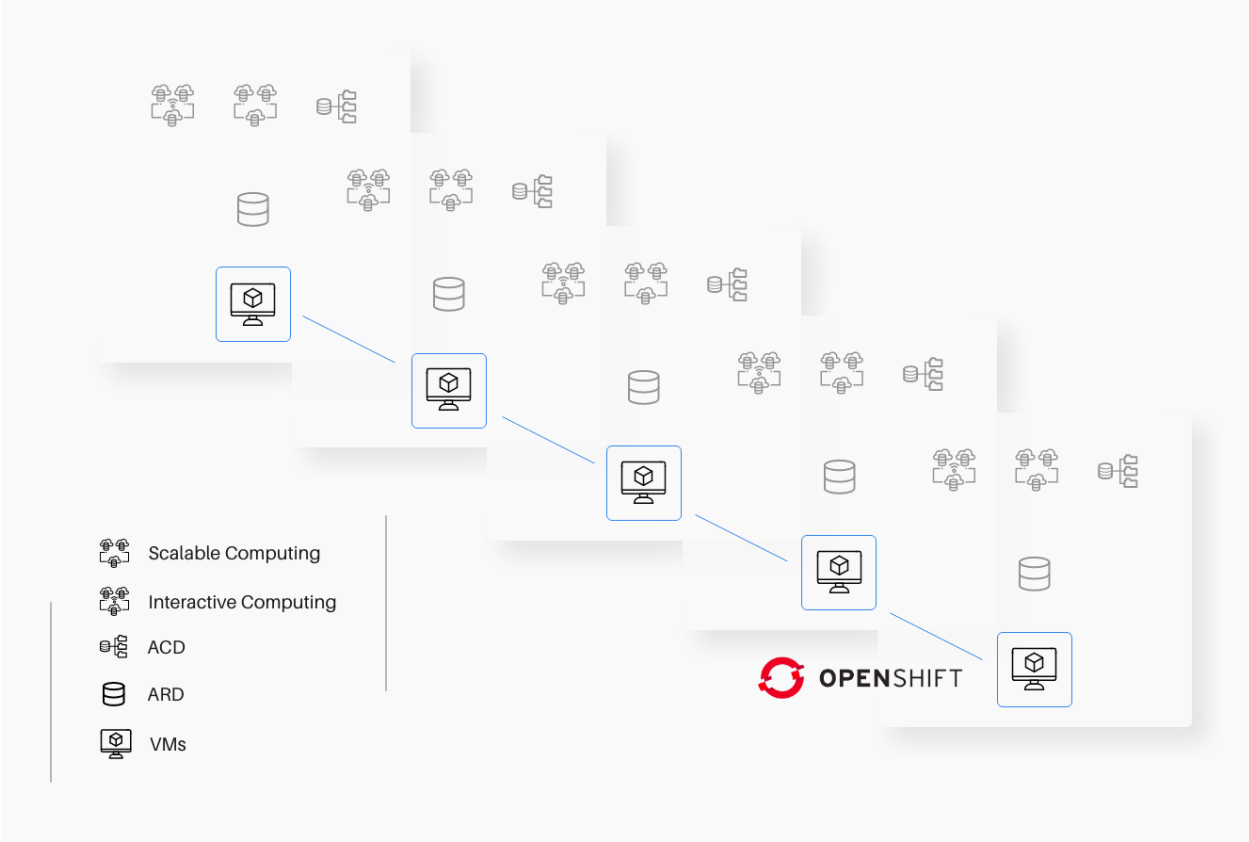


Figure 13:OpenShift is used as a Container Orchestration Platform and is an open-source cloud development Platform as a service (PaaS), which enables the developers to develop and deploy applications on cloud infrastructure.

3.2.1.2. Scalable Computing

Scalable Computing systems pare provided by FENIX ICEI and accommodate usability of massive parallel HPC systems that are suitable for highly parallel simulation or high throughput data analysis tasks. Services like these are the perfect candidates for computational data analysis and simulations that require high compute performance, high memory bandwidth and large capacity offered by a large set of high end computing servers. Since resources are not unlimited even in these enormous supercomputing systems, there is a need for scheduling and managing tasks in order for maximum utilization to be offered. Schedulers and workload managers are capable of scheduling and managing resources for batch jobs in a way where expensive computational resources are dynamically allocated periodically for accomplishing the respective outcome. Batch jobs that use scalable compute services, will use parallel file systems as data repositories allocated near to compute resources for high-bandwidth and low latency accessing to data. One key aspect that scalable computing services provide is the Message Passing Interface (MPI) communication protocol that is capable of high performance, scalability and portability.

3.2.1.3. Interactive Computing

Interactive computing services, which are also provided by FENIX underlying infrastructure, give access to computational resources via interactive sessions for visualizing large data sets, for computing via Jupyter notebooks, for data manipulating and post processing. In general, there is a need for real time interaction with a program during runtime that is motivated by the need of estimating the state of the program, its future tendency, accessing intermediate results without waiting and steering the computation by modifying input parameters or boundary conditions. These services refer to the capability of a system to support distributed computing workloads while permitting on the fly interruption by a user. Components of interactive computing are front-end systems for users to interact with the program during the runtime, while in the meantime running applications consume HPC resources like CPU, memory and GPU accelerated compute nodes for heavy visualization of large outcomes. Usage scenarios of interactive computing is visualization, processing, reduction of large amounts of data especially when processing cannot be standardized or implemented in a static workflow. Scientists perform interactive processes like reduction and production of data views that may include even complex processing like convolution, filtering, clustering. All those processing steps can be parallelized in order to take advantage of HPC resources but this could become a bottleneck for users to open sessions into separate interactive steps by batch jobs as their scheduling would delay the entire execution. In general, scientific communities started to use R, Stata, Matlab/Actava and Jupyter notebooks as interactive frameworks and scripting languages to integrate the compute and data processing applications running in batch systems. Remote visualization (VirtualGL, paraview), Rstudio and Matlab are also often used. Some of the Interactive computing services' characteristics are the high end volatile memory configuration, high bandwidth access to data repositories, tight integration with scalable computing service, scale- out or connection to running scalable compute jobs, access to GPU for visualization, JupyterHUb service for interactive computing and support of different containerization methods like Singularity, Shifter, Docker. There is a great support for efficient handling of interactive sessions, maximizing resource utilization while executing different workloads. Also, there is support for staging of data quickly across multiple memory and attached storage tiers, and improving energy consumption. With respect to combining interactive computing services with scalable computing services, a balance between batch jobs and interactive sessions is taking place overall.

3.2.1.4. Active Data Repository

Active Data Repositories can be associated with high performance parallel file systems for storing data for a short period of time. These repositories are close to the scalable and interactive computing services for faster exploration and data manipulation. Good performance is achieved, since repositories accommodate high bandwidth and high Input/output Operations Per Second (IOPS) rates. The repositories are implemented as file systems based on LUSTRE, GPFS as well as IBM Spectrum Scale and are associated via a POSIX interface for fast data access. Data in Active Data Repositories are not federated in any way and it is mandatory for users to move it to other types of data repositories (like Archival Data Repositories) in order to keep them backed up, archived and safe. Data inside these repositories is typically replicas of data and objects, with master copies of them being located in other storages.

3.2.1.5. Archival Data Repository

Another type of repository that FENIX offers to its users, is called Archival data repository. Archival Data Repositories are repositories optimal for providing capacity, reliability and availability of data and objects stored in them that are usually used to share data outside of the HPC centres. Data in these repositories can be federated providing access to geographically distributed data storages. Data stored there, are called cold, since it cannot be easily regenerated and their storage mandates long-term accessibility. The most known interface for storing data in that repository type is OpenStack Swift Object storage. Swift is a highly fault-tolerant object storage service that stores and retrieves unstructured data objects living inside containers. Archival Data Repositories in contrast with Active Data Repositories do not provide high-bandwidth and low-latency access that HPC systems require. Thus, movements of data from the different kinds of repositories are mandatory.

3.2.1.6. Virtual Machines

FENIX offers services for deploying virtual machines in a stable and controlled environment to its users. Some of the features that Virtual Machines provide are virtual CPUs, virtual GPUs, SSD that are performance-optimised storage, HDD for capacity optimised storage, and large amounts of memory. Users can fully customize their Virtual machines by selecting different flavors of machines in specifications as well as in the software itself. They can use predefined images for launching the machines or use customized ones. Some Virtual Machines also provide integration with GPUs for supporting visualization and interactive sessions. Even though there are a lot of providers who support resource oversubscription, FENIX deals with a more strict plan where there is no elasticity in resources like CPU, GPU and memory.

3.2.1.7. Data movement

As previously presented, different kinds of repositories are used for different facilities, from reliable storing of master copies of long-term data to having replicas of data near the compute resources for high bandwidth, low latency and good performance while analysis and simulation experiments are taking place. Since, FENIX offers a number of Active and Archival Data Repositories, there is a need to also provide ways of moving data between one and the other when it is mandatory.

As a first reliable way for moving data between the different types of data repositories, Active and Archival, users can make use of the Swift Object storage Command Line Interface in order to upload and download objects from buckets, into POSIX file systems and vice versa. In that way data can be transferred near the computational resources when it is needed and back to the long-term, reliable, with the large amount of capacity repository once the data manipulation task is finished and the output is provided.

As previously described, Active Data Repositories offer a great variety of different POSIX file systems existing in a FENIX site. When users need to run batch jobs for scalable and compute services, replicas of data need to exist in the SCRATCH file system that is located near to scalable and interactive resources for high bandwidth and low latency. If master copies of data are stored in other POSIX file systems like the PROJECT or the HOME, transferring of data from one POSIX file system to the other needs to take place. For this transfer, users can use batch jobs that take advantage of the SLURM workload manager system which offers high availability and will overcome hardware or software

failures in case they occur. In these jobs, rsync-ing can be possible to be completed since SLURM will be responsible for taking care of any corruptions between the transferring. A job will be submitted in SLURM in order for data to be moved inside two Active Data Repositories. This feature will offer restarts in case when network misconfiguration takes place as well as assurance that all needed data is moved.

3.3.2 Middle Layer: Complementary EBRAINS services

In this subsection, we will provide some complementary EBRAINS services that can be referred to as the middle layer of EBRAINS architectural diagrams. These services exist for easing the everyday life of scientists and users of EBRAINS.

3.3.2.1 Collaboratory

One of the complementary EBRAINS services that is delivered to EBRAINS users is the Collaboratory. Inside Collaboratory, collabs exist as the entry points of collaboration between different users. A collab extends across multiple services to share its content with a team of users who have a specific set permissions. From this page, users can create, edit, and browse wiki pages. They are a convenient way of publishing content in a specific target group by choosing if the collab will be public or private and by managing permissions for people inside the EBRAINS consortium. Collab also gives the opportunity to users to interactively work on Office documents via OnlyOffice and store them inside a dedicated collab's Drive. All files inside Drive are version controlled and users can roll back to a previous one quite easily. The most critical aspect that Collaboratory offers is the JupyterHub service. EBRAINS users, thus researchers of the EBRAINS, can access Jupyter notebooks in order to work together with other users, sharing code, documentation and data. Inside collab users can access JupyterLab notebooks and spawn instances of them on a hosted JupyterHub, in a reliable and fast way. Not only are JupyterLab notebooks one of the most promising interfaces for interactive work among developers and neuroscientists but they also provide great benefits like an easy way for a user to run large jobs in an HPC system or Neuromorphic hardware.

3.3.2.2 Collaboratory

Another complementary EBRAINS service that can be used by EBRAINS users is data proxy. Data proxy is responsible to act like a middleware application where a user with an EBRAINS account can access SWIFT Object storages that the ICEI FENIX underlying infrastructure provides as the Archival Data Repository, without the need to have a dedicated FENIX user account. Data proxy authenticates users with the EBRAINS authentication and authorization service. It also provides a service account, where data is tracked by whom can access it. A Swift Object container, namely bucket, can be associated with each and every researcher workspaces'. Buckets are used for large but cold data, data that does not change that often, like brain scans and EEG files. Data inside a Bucket are called objects. As currently architectured, researchers' workspaces and buckets are strictly associated, meaning that users' permissions define what actions can do to the specific bucket. A correlation follows:

a) User that has a *Viewer* permission in a collab - can *Read* data in a collab's bucket
b) User that has an *Editor* permission in a collan - can *Create, Read, Update, Delete* data in a collab's bucket
c) User that has an *Admin* permission in a collan - can *Create, Read, Update, Delete* data in a collab's bucket
d) Users with *no permissions* in a collab - can do *no Actions* in a collab's Buckets
e) Data proxy provides an API endpoint for programmatic access as well as an interface for users to manually manage permissions of different objects in a bucket inside a workspace.

3.3.2.3 JupyterLab

In general, JupyterLab [22] is a web-based interactive development environment for Jupyter notebooks, code and data. JupyterLab configures and arranges the user interface to support a wide range of workflows in data science, scientific computing and machine learning. Jupyter notebook is an open-source web application that allows users to create and share documents that contain live code, equations, visualizations and narrative text. At EBRAINS, JupyterLab installation on top of OpenShift makes Jupyter notebooks one of the most promising tools for interactive computing between developers of EBRAINS, researchers and/or neuroscientists. Members of EBRAINS that access the Collaboratory, can share their work via Jupyter notebooks to other members, groups or units of the Consortium. Jupyter notebooks via EBRAINS Collaboratory make integration with HPC systems and Neuromorphic hardware quite easy. A load balancing functionality needs to be in place so JupyterLab instances can be spawned at different FENIX sites transparently from the viewpoint of developers.

3.3.2.4 Authentication/ Authorization

Authentication and authorisation are critical when it comes to security. In general, authentication is the procedure where the system identifies a user. Authorisation on the other hand is the act of granting or denying the right of a user to complete an action. The centrally provided Identification and Access Management (IAM) service for EBRAINS runs on Keycloak, which acts as Identity Provider Broker (IdP Broker) to independently supplied IdPs and also provides an OpenID Connect (OIDC) IdP itself for authentication and authorization (based on OAuth2.0 authorization). Practically, it authenticates EBRAINS users and authorises them for the different services. A powerful aspect of the IAM service is that it provides Single-Sign-On across all integrated services. In that sense, users can login into different tools and services by providing the same EBRAINS credentials. Finally, the IAM service provides means for services to connect to each other.

3.3.2.5 UNICORE

UNICORE (Uniform Interface to Computing Resources [40]) offers a ready-to-run system including client and server software. UNICORE makes distributed computing and data resources available in a seamless and secure way in intranets and the internet. UNICORE consists of four components. The UNICORE/X server is the central component of a UNICORE site. It hosts the services such as job submission, job management, storage access, and provides the bridge to the functionality of the target resources, e.g. batch systems or file systems. UNICORE/X is deployed in each FENIX site. UNICORE TSI is a daemon running on the frontend of the target resource (e.g. a cluster login node). It provides a remote interface to the operating system, the batch system and the file system of the target resource. It is used by the UNICORE/X server to perform tasks on the target resource, such as submitting and monitoring jobs, handling data, managing directories etc. The TSI, which is also deployed in all FENIX sites, performs the work on behalf of UNICORE users. The UNICORE Registry server provides information about available services to clients and other services. It is deployed only in one FENIX site. Last but not least, the UNICORE Workflow service provides advanced workflow processing capabilities using UNICORE resources. The Workflow service provides graphs of activities, submits and manages the execution of single UNICORE jobs. The Workflow service offers a REST API for workflow submission and management and uses an easy-to-understand workflow

description syntax in JSON format. EBRAINS users may configure a UNICORE/X server to run High Performance Computing (HPC) jobs or they can run Jupyter Notebooks that use UNICORE to submit HPC jobs. With that approach, authentication is taken care of and a mapping between users' credentials and HPC accounts is taking place, so only users or service accounts with HPC quotas can run HPC jobs in an HPC system.

3.3.2.6 Container registry

Containerization is one of the most promising ways of packaging tools with their dependencies, libraries and binaries. EBRAINS provides a dedicated docker registry in order for EBRAINS users to store, find and access container images associated with tools and services of the different service categories from the upper layer of EBRAINS. The EBRAINS Docker registry is based on Harbor, an open-source registry. EBRAINS users log into the registry and can push and pull public images as well as images with specific permissions, like ones existing under a project in which they are members.

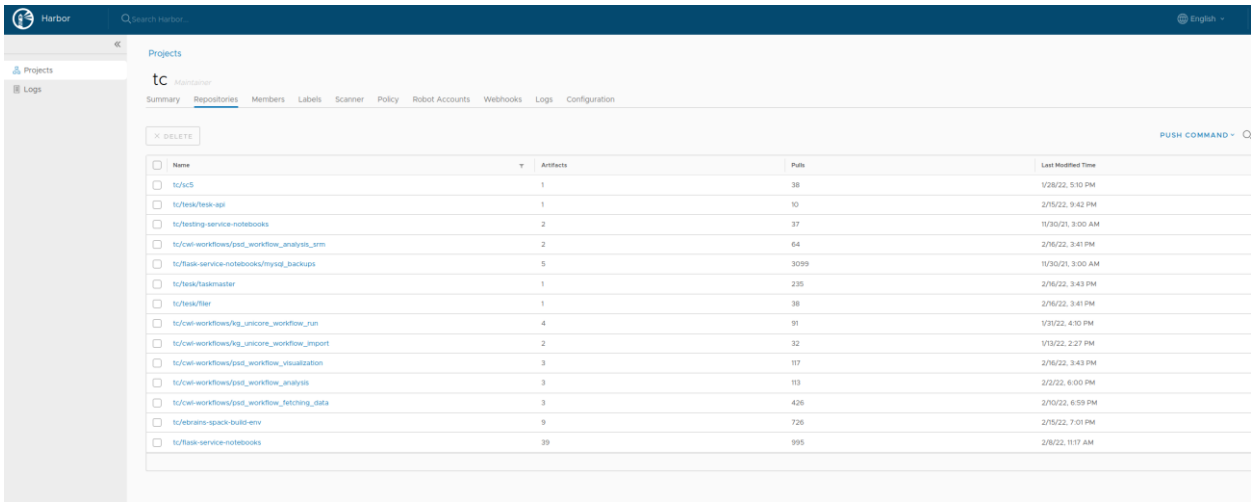


Figure 14: Harbor is a dedicated EBRAINS container registry for storing, finding and accessing containers built from EBRAINS users

3.3.2.7 Knowledge Graph

Knowledge Graph is a metadata management system built for EBRAINS. Specifically, it is used as a multi-modal metadata store that combines information from different fields on brain research, data / models, software existing in EBRAINS. Neuroscientists that wish to share their data in Findable, Accessible, Interoperable and Re-usable (FAIR) ways can apply for user support to have their data and models curated by a group of specialized curators. In that way they can make their data and models easy to be discovered and reused by other researchers. Knowledge Graph also offers rich metadata annotation, an important aspect of finding, using and re-using actual data and models. One feature of great importance is the new metadata models that are already released under the umbrella of the open Metadata Initiative for Neuroscience Data Structures (openMINDS) and are adopted by the EBRAINS Knowledge Graph. openMINDS gathers a set of metadata models describing heterogeneous neuroscience data. With respect to (re)-using data and models found in Knowledge Graph at EBRAINS a comprehensive collection of metadata is captured in full detail for provenance reasons. Data assets, methods and models, created within EBRAINS capture metadata that pertains to input/output files, software version, environment at which computation runs, started by agent or person, hardware system, configuration files. Different use cases within EBRAINS may need further information to be captured, so the overall process could be reproduced. EBRAINS by providing Knowledge Graph gives programmatic ways for users for finding and (re-)using dataset and models as well as graphical user interfaces for making the procedure more applicable to researchers with no prior programmatic knowledge. Comprehensive tools and services for publishing FAIR data and computational models. The services provide long term data storage, citable DOIs, defined conditions and licenses for use of data, and tags to make the data discoverable, interpretable, and re-usable. The actual storage of datasets is provided by ICEI Fenix. Connection between the Knowledge Graph registry with the Archival data repositories is user oriented in the sense that the user should associate data with Digital Object Identifiers existing in Archival data repository.

3.3.3 Upper Layer: Entry point for scientists

Under this subsection, the upper level of EBRAINS architectural representation will be depicted. In this upper layer, scientists and researchers at EBRAINS have plenty of service categories and tools in their disposal that can use in a collaboratively way with other communities while at the same time can access data related to brain for scientific objectives and research purposes.

3.3.3.1 Services

There are five Service categories that are the foundations of the EBRAINS community and portray the scientific work at EBRAINS. Each of these services provide a plethora of tools that can be used in data simulation, data analysis, sharing and finding data, as well as in brain related research like atlases. In this subsection we are briefly going through each of the Service categories emphasizing how EBRAINS users from the neuroscientific community can benefit from by using these for their research.

Data and Knowledge is the most important and key element in the EBRAINS research infrastructure. An important element is the open and shared knowledge that all research communities need to adhere to and provide for the sake of open science. Researchers often make publications and associate their practical with their theoretical work for papers, symposiums and scientific magazines. One of the appealing features that EBRAINS provides to the journal researchers is the ability to associate datasets with a publication. Different stages exist where they can choose between either allowing public visibility and accessibility to the dataset before the paper has been published or allowing the public to check only the metadata but not the data itself before the paper has been submitted. In the latter, researchers will be able to provide public temporar links for the reviewers to be granted access to the data. In both cases a Digital Object Identifier (DOI) is associated with the dataset that is or will be associated with the publications.

Brain inspired technologies at EBRAINS leverages the understanding of computational capabilities of spiking neural networks. Brain is a very complex, huge and structured organ in such a way where simulating a whole unique brain is hard. This is why different ways of exploring layers of the brain need to be enhanced. Spiking neural networks do not require large amounts of energy and data, unlike standard deep neural networks require, thus it is the key point for neuroscientists to understand how the brain functions, and how to implement higher cognitive functions.

Medical data analytics services provide two platforms that cover key areas in clinical neuroscience research. Medical Informatics Platform federates large clinical datasets to process them in a privacy manner. These processes have to do with machine learning algorithms for data exploration, modelling and statics. Another platform called, Human Intracerebral EEG Platform, is designed to collect, store in a central clace, curate, share and analyze Intracranial electroencephalogram (iEEG) data focusing on investigating cognition, consciousness, connectomics and related disorders.

Simulation and Analysis service offer tools for simulation studies related to brain research. These services provide integrated workflows for model creation, simulation and validation including data analysis and visualization. Simulation is divided into different levels of the brain, from cellular to network level to whole brain ones. High performance libraries for simulations ranges from single-cell models to large networks. Simulators which simulate spiking neural network models of any size exist as well as whole platforms which create personalized brain models and simulate multi-scale networks.

EBRAINS provides and develops open access to 3D atlases for human, rat and mouse brain analysis. Atlases allow users to explore the brain, various facets, navigate, characterize and analyse the data on the basis of anatomical location within the brain. For EBRAINS, ATLAS related services can support neuroscience research in neuroanatomy, in experimental neuroscience, neuroimaging, computational neuroscience, brain - inspired Artificial Intelligence research, neuro synergy and medicine. With Atlases in neuroanatomy someone can explore brain architecture in 3D and in different spatial resolutions, they can study the shape and spatial relationships of brain regions as well as functional modules and their connections. Co-display experimental data in a 3D high resolution reference brain space, find spatially relevant features to interpret data and confirm support user studies in experimental neuroscience. Atlases combine whole brain imaging data with multimodal features at the cellular level.

3.3.3.2 Tools

Tools developed in the context of Service Categories from the upper layer of the EBRAINS architectural representation will take advantage of the bottom layer of EBRAINS which will be delivered by FENIX ICEI. High Performance Computing services from the underlying infrastructure will facilitate the scientific work and give the state of the art and cutting edge technologies for scientific communities to produce a significant impact to brain research while using tools associated to EBRAINS.

There are two different kind of tools (interactive vs non-interactive) that exist at EBRAINS. In general,

- Interactive tools: software that allow user interaction during runtime.
- Non-interactive tools: software, with well-defined inputs and outputs, that do not allow user interaction during the runtime. It is possible for the user to provide all parameters needed for the software to run prior to the execution. In this way, parameterisation of the results can happen on-the-fly.

For EBRAINS and for the current thesis, we are focusing on non interactive EBRAINS tools.

- Non-interactive EBRAINS tools: scientific data simulation or analysis tools bundled together with dependencies, binaries and libraries, capable of resolving misconfigurations of software, as well as of reusing and versioning purposes. These tools will have a strict input and output format that will be well documented in order to make EBRAINS tools interoperable.

3.3.3.3 Data

Another important asset of the upper layer in the EBRAINS architectural diagram is Data. Data at EBRAINS are FAIR and can be shared, found and (re)-used through Knowledge Graph which is a metadata repository holding information and links to where data are actually stored. All types of neuroscience data from imaging data, to electrophysiology, omics and informatics can be shared and found via the EBRAINS platform in order to be aligned and registered with brain atlases. It will be possible for data to be searched by Knowledge Graph as well as different atlases, to be used in analytical workflows and to extract features for simulation and robotics parts. Understandably, all these kinds of data from raw to derived, to models are by definition multimodal, heterogeneous and differently organized. Thus, a process needs to exist in order to transform the data to a standardized manner so that they can be effectively searched, compared and analysed via tools existing in the EBRAINS RI. EBRAINS provides a curation process in order to make data properly

documented and organised by using metadata management standards and formats and integrating them with EBRAINS Knowledge Graph and Interactive Atlas Viewer by using standard interoperable schemas. All neuroscientists can integrate their data with EBRAINS by mapping the data with a persistent identifier, for example a Digital Object Identifier (DOI).

3.3.3.4 Collaboration

The last asset of the Upper Layer of EBRAINS architectural representation is Collaboration. EBRAINS offers a central point of collaboration between different groups of researchers, the Collaboratory. It is a community platform for creating an environment where scientists can identify fellow researchers and developers, form teams, coordinate, develop live code via Jupyter Notebooks, document methodology and results in pages and files, store data, and publish work in a safe environment which is entirely self-hosted. It is built upon an integration of JupyterHub providing notebooks, OnlyOffice for collaborative editing, Seafile as a shared drive, and XWiki for documentation in wiki pages. In that sense, collaboration has become easy between individual people and teams. Collaboratory is also a way to do interactive work and share data with EBRAINS users.

3.3.3.5 Representation

As previously described, the upper level can be represented as:

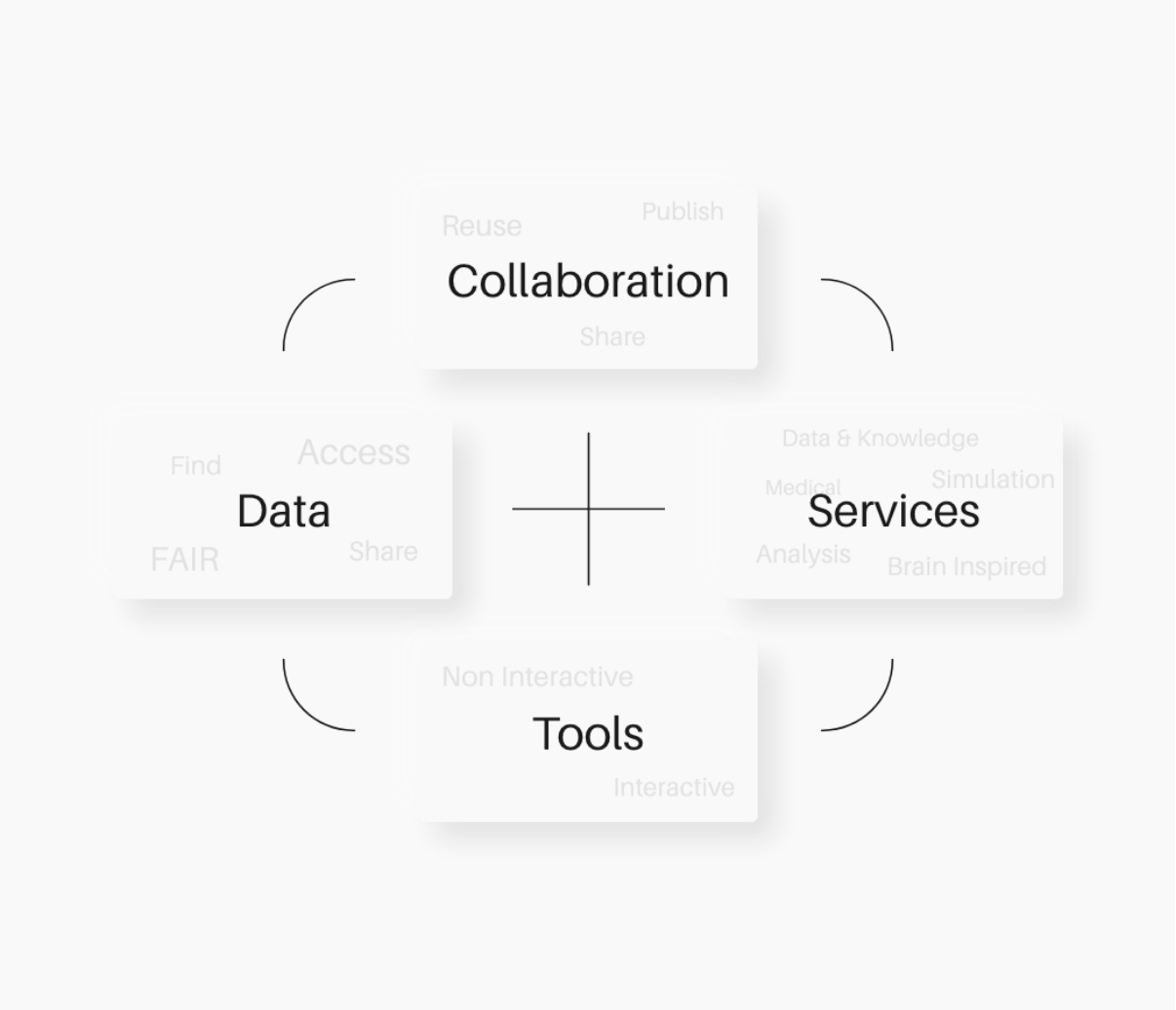


Figure 15:There are different tools, services as well as FAIR data that can be found under EBRAINS for its users' to use. As a main goal, collaboration at EBRAINS between different teams and communities is ensured.

Briefly, researchers and scientists at EBRAINS can associate FAIR data delivered by the Data and Knowledge service category with a plethora of EBRAINS non interactive tools delivered by one of the service categories in order to create their flow of work which is associated with data manipulation, either analysis or simulation. Scientists also have the opportunity to share, work, collaborate with more groups of scientists and researchers via the Collaboratory in order to introduce an Open and FAIR science and research focused, but not dedicated to the brain related scientific communities. Practical work could be associated with theoretical work in order to publish them in symposiums, journals as well as papers.

3.4 Scientific Computational workflow at EBRAINS

After describing the different architectural layers of EBRAINS we will present how scientists use EBRAINS tools, data, supplementary services as well as the underlying infrastructure in order to run their simulations, analysis and data manipulation processes, combining all three, bottom, middle and upper layer at once. We will introduce a real life example of scientists at EBRAINS finding tools and data from Knowledge Graph, copying data to where the analysis or simulation will take place, running their tasks in one of the HPC systems that FENIX ICEI underlying infrastructure offers to EBRAINS users', and getting data back to store it in the Archival Data Repository.

3.4.1 Getting data near the compute nodes

In order for scientists and researchers to associate their experiments with data related to brain, they need to find or use their own appropriate set. Knowledge Graph is the metadata management system that is used for finding data at EBRAINS. Data inside Knowledge Graph is either free, or under embargo which means that specific permissions are needed in order for scientists to access it. As previously stated, Knowledge Graph is just a metadata catalogue, thus data is associated with only metadata. Once scientists find the information of the data that they want to use, they will have to find where the real location of data is. Data is usually stored in one of the Archival Data repositories that offers reliable, federated and long term storage which FENIX ICEI underlying infrastructure supplies. Scientists will have to move data near the computation nodes in one of the reliable ways (scp or rsync), in order to run their analysis, simulations or any data manipulation process at either HOME or PROJECT filesystems. Once they are ready to launch their simulation, data need to be manually transferred (cp) to the SCRATCH filesystem which stands as the Active Data Repository that has good performance and it is closer to the HPC systems.

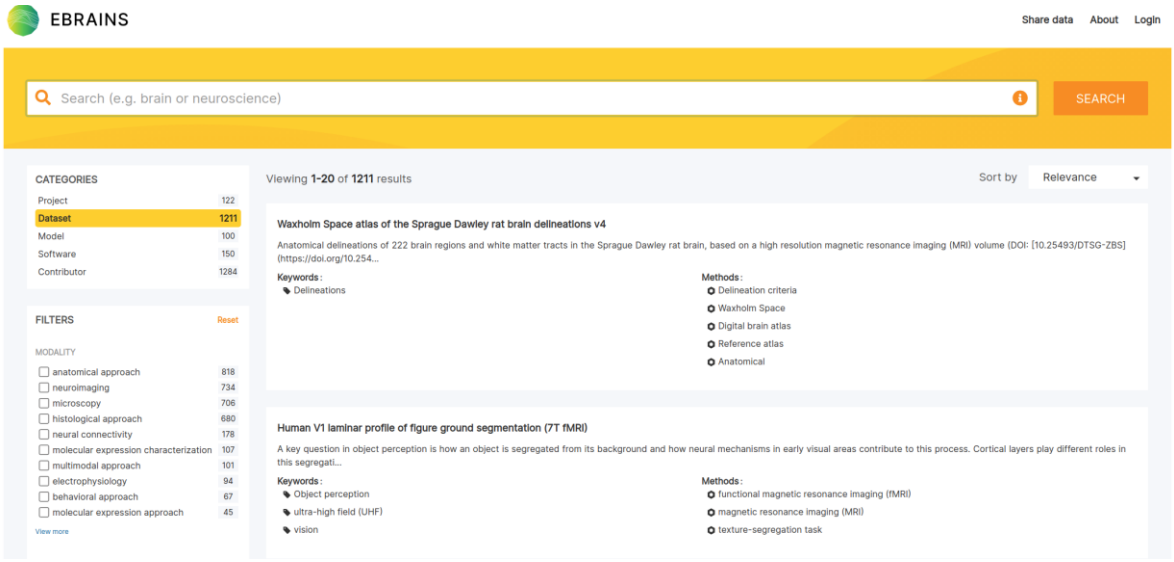


Figure 16: EBRAINS Knowledge Graph as a multi-modal metadata store that combines information from different fields on brain research, data, models and software existing at EBRAINS

3.4.2 Find tools for analysis and simulation

Other than data, scientists and researchers could search for tools provided by EBRAINS in order to use in their experiments. Different tools from the service categories are also accessible and findable inside the Knowledge Graph. Since in the current thesis, we are focusing on non interactive EBRAINS tools, scientists can search for software that has a specific type of input and output format and scientists can not interact with during the execution. There is a large variety of non interactive EBRAINS tools that exist, from either static services, libraries, applications or just API endpoints that can be used by EBRAINS users. Under the current thesis, we are mostly interested in software, libraries, APIs and tools that can be executed inside an HPC system. The heterogeneity of having non interactive tools in such different ways of representation from services to APIs, make it hard for scientists to know which of them is appropriate to be used and some times make the learning curve exponentially high. There are times where no documentation exists or times where email exchange and support between different teams need to be made in order for scientists to know how to use a tool found under EBRAINS Knowledge Graph.

3.4.3 Use HPC to run experiments

HPC systems are offered through the bottom layer of EBRAINS and are often used by the scientists because they have a good performance when it comes to big data produced and used and when time consuming jobs need to be executed. There are different ways for scientists to use HPC systems in order to run their experiments.

3.4.3.1 SLURM job

One of the available and reliable ways to connect via HPC systems is ssh-ing into a login node. As stated in a previous section, users must not run large, intensive and time consuming tasks in the login nodes since they are shared to all users and their purpose is to be used for submitting jobs in the batch system. After scientists ssh-ed into the the login nodes they can make use of the SLURM workload manager and scheduler that runs on top of the computing nodes of the HPC systems in order to submit their experiments.

In this first example, scientists need to write a batch job in a clear format using the appropriate syntax and command and use SLURM in order to submit the job. The job will be queued and executed once the resources needed are available. Once the job is executed, the output of the job is written in the SCRATCH filesystm wich is close to the computational resources and acts as the Active Data Repository. Scientists must copy the produced data in an Archival Data Repository where long term data are safely stored as soon as possible since the SCRATCH filesystem is purged within the next 30 days.

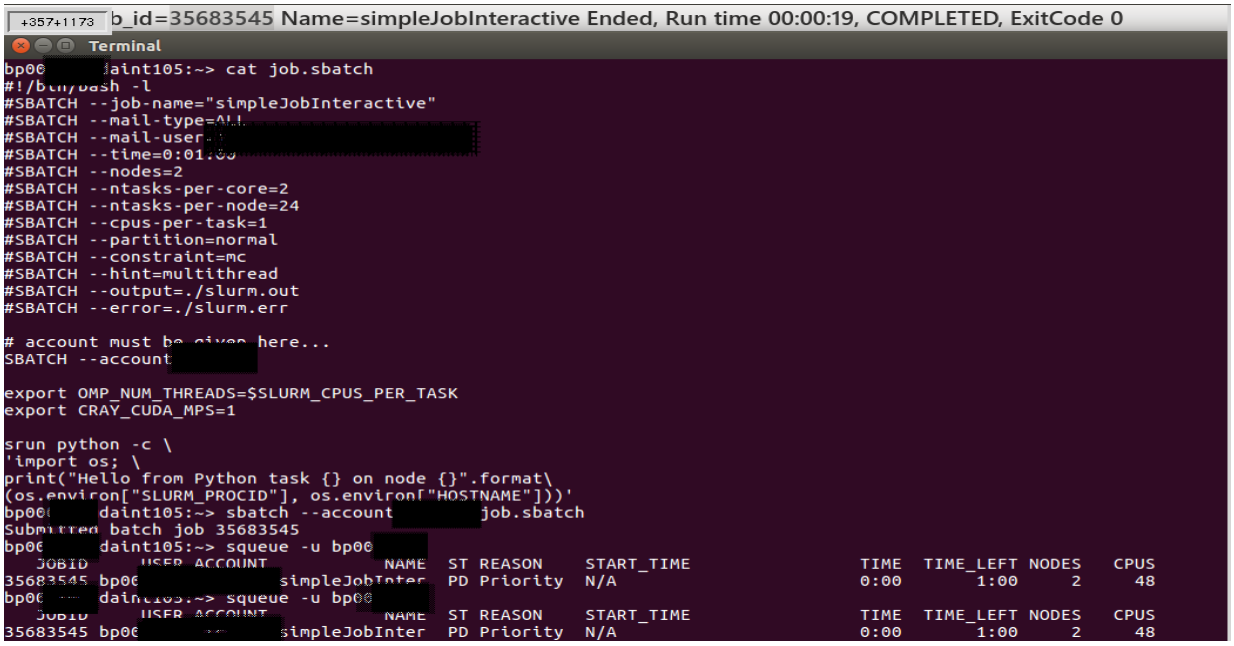


Figure 17: User executes a batch job from a login node inside an HPC system via Slurm batch system. An email notifies the user when the job is executed. The user needs to come back to check for results and move the outcome to the Archival Data Repository.

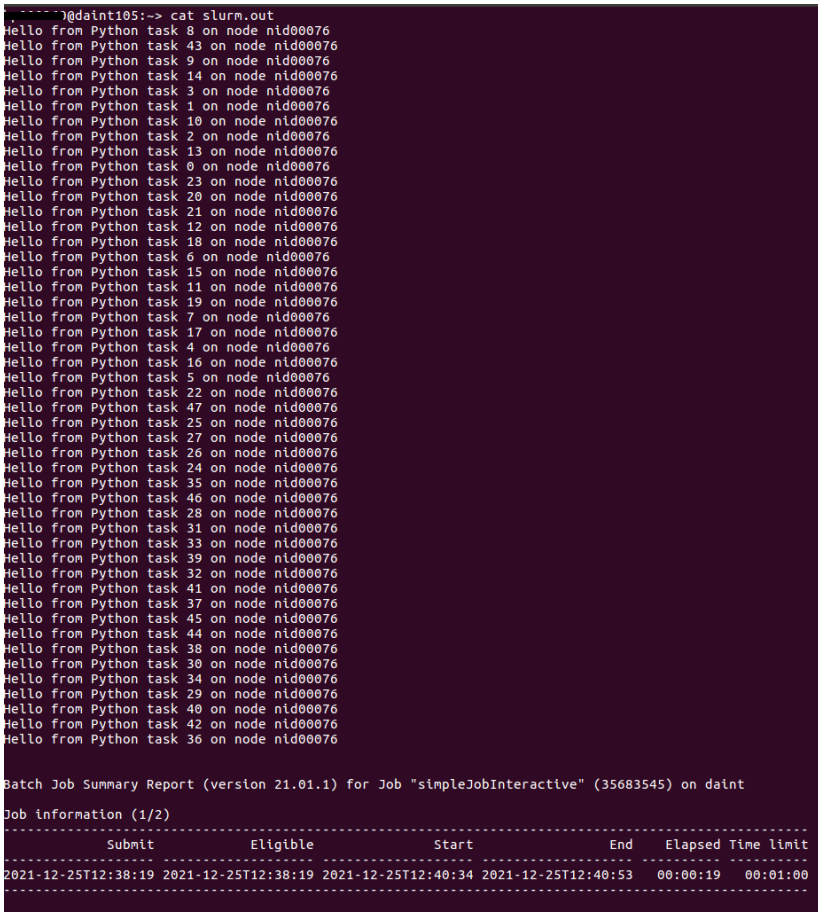


Figure 18: Output provided by the sbatch job executed via Slurm batch system.

3.4.3.2 Jupyter notebooks

In this second example, EBRAINS offers ways for fewer interactions between scientists and the SLURM workload manager and scheduler. After getting the data near the HPC systems (scp or rsync), scientists may open a pre-defined Jupyter notebook Figure [Figure] provided by JupyterHub inside the Collaboratory that offers a Graphical User Interface in order for scientists to send jobs in the HPC system in an opaque way. This notebook uses PyUnicore which is a python library for seamlessly interacting with HPC systems. Users can send jobs in HPC systems, request status, receive results back and more. As in the previous real example, SLURM again takes care of the real execution of the job itself as well as of sending the results and status back to the users. Scientists can choose one of the available HPC sites to run their experiment that again needs to be written in a clear and appropriate format for SLURM to understand. The job itself is also sent by PyUnicore to the HPC system that scientist has already chose.

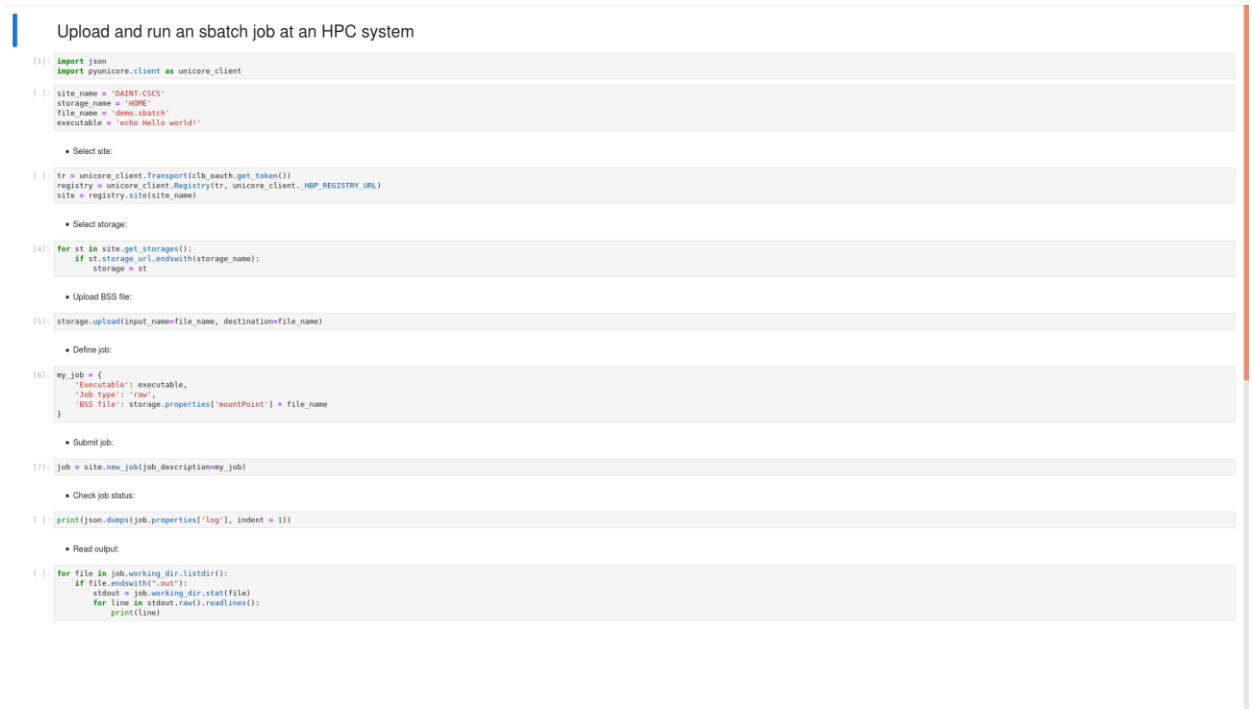
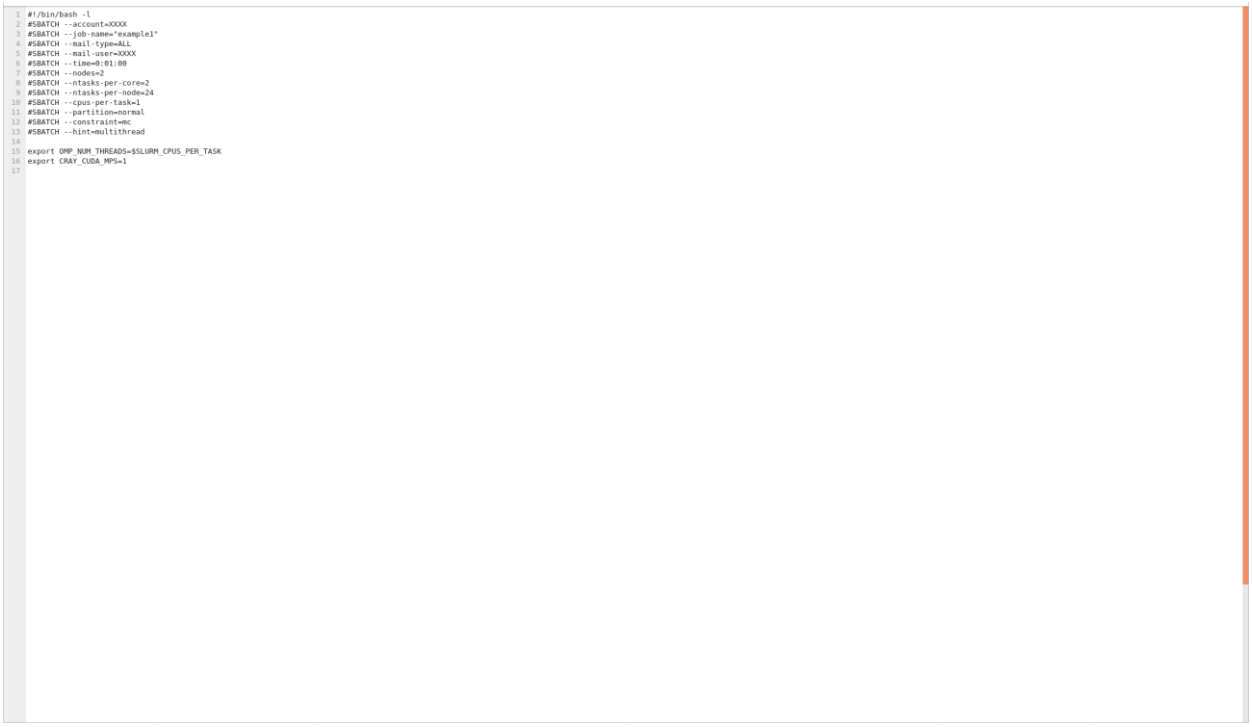


Figure 19: User enters a Jupyter notebook in order to launch SLURM jobs in HPC system for executing the experiment. Jupyter notebook consists of a Graphical User Interface for easier interaction between user and underlying infrastructure.



```
#!/bin/bash -l
#SBATCH --account=XXXX
#SBATCH --job-name="example1"
#SBATCH --mail-type=ALL
#SBATCH --mail-user=XXXX
#SBATCH --time=0:01:00
#SBATCH --nodes=2
#SBATCH --ntasks-per-core=2
#SBATCH --ntasks-per-node=24
#SBATCH --cpus-per-task=1
#SBATCH --partition=normal
#SBATCH --constraint=mc
#SBATCH --hint=multithread

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export CRAY_CUDA_MPS=1
```

Figure 20:SBATCH job is pre described in a the appropriate format and will be transferred by PyUNICORE inside the HPC system that scientist has already selected.

3.4.3.3 Snakemake

As a last example, a number of scientists have identified the importance of using workflow engines which can automate the execution, as well as properly describe different tasks that can be executed one after the another as a pipeline on top the HPC systems. For that matter, scientists often use Snakemake, a system that works well with SLURM in the HPC systems and is a friendlier way for scientists to describe their tasks via Snakefiles in addition to sbatch jobs. Snakefiles consist of rules with input, output fields as well as shell command line tools to be executed in a specific order. In that example, scientists should also interact in a way with the HPC systems in order to launch the Snakefiles, either via ssh-ing into the login nodes, or launching the Snakefile via PyUnicore inside a Jupyter notebook.

3.4.4 Move output back to Archival Data Repository

As the last part of the scientists' real example, a movement of the produced outcome from the Active Data Repository (SCRATCH) to the Archival Data Repository is needed in order for data to be safely stored in the long term storage. Since Active Data Repository only holds replicas of data and is purged periodically, it is important that scientists move data (scp or rsync) as appropriate as soon as the result is produced. If users would like to make results publicly available, a Digital Object Identifier (DOI) associated with the data, as well as a request for curation of data to a dedicated EBRAINS team are needed. Curation team decides if data adhere to specific attributes and then is shared inside the Knowledge Graph.

4. PILOT WORKFLOW SYSTEM AT EBRAINS

After exploring different Research Infrastructures associated with communities dedicated to Life Science, and assessing the current state in which scientists of EBRAINS are using tools, data and the underlying infrastructure, it is apparent that several features regarding standardization are missing. Standardized workflows provide accessibility, shareability, automation, reproducibility, portability to scientists' work, and FAIRness, OPENness and interoperability to EBRAINS.

There is a need to fill in the gap that exists between the bottom layer, where FENIX infrastructure provides services and the upper layer that EBRAINS researchers interact with. While adhering to the new technologies, researchers will be able to decouple the description of scientific computational workflows from the different adaptations and configurations needed to execute them in different infrastructures, systems or hardware. Scientists at EBRAINS will not need to strictly interact with the bottom layer in the extend that they are currently doing by ssh-ing into HPC systems or describing their experiments in batch formats.

In the current thesis we propose ways of authoring scientific computational workflows in a common, standard, widely known, and acceptable way that harnesses the needed level of automation, scalability, portability, reproducibility.

In this section we propose the establishment of a pilot workflow management system for EBRAINS focusing on the below mentioned topics:

- We introduce the **Common Workflow Language** open standard.
- We **package** non-interactive **EBRAINS tools** with dependencies, libraries and binaries via Docker.
- In addition, we **define** those packaged tools along with their inputs and outputs types in the **CWL** format. These tools described in open, common and standard ways can become workflow steps in scientific computational workflows also described in CWL format.
- Furthermore, we **deploy a primary workflow engine** on top of HPC underlying infrastructure system at EBRAINS in order to automatically execute, monitor and retrieve results of scientific workflows defined in CWL. EBRAINS users will use a Command Line Interfaces (CLI) in order to submit and monitor their scientific computational workflows.
- We propose a **central point of reference, an EBRAINS Hub, for finding, storing and accessing** the already described tools and workflows in **EBRAINS Knowledge Graph,** a metadata management system that EBRAINS offers.
- We also propose a **Graphical User Interface** that will replace the use of CLIs for submitting and monitoring scientific computational workflows. The proposed GUI has been fully designed but not implemented in the scope of the current thesis. The design process is based on high fidelity mock-ups.
- Finally, we provide a **real user story** taking advantage of all three- bottom, middle and upper- layers of EBRAINS, as previously described, along with the introduced standardized workflows.

In the scope of the current thesis:

- **Scientific Computational Workflows** can refer to series of non interactive **EBRAINS tools linked** in order to create graphs, loops or branches for **accomplishing scientific objectives**. Data flowing between the different tools depict the order of execution of the EBRAINS tools.
- **Non interactive EBRAINS tools** can refer to data manipulation, simulation or analysis tools packaged with dependencies, libraries, binaries and software via Docker. EBRAINS tools are defined via CWL Command Line Tool Description Specification [1] and scientists can not interact with them during the execution.
- **Standardized workflows** can refer to chains of EBRAINS tools as workflow steps connected in a specific way to create directed acyclic graphs (DAGs) of operations. CWL Workflow Description Specification [1] is used in order to define standardized scientific workflows as structured recipes along with all the steps, inputs and output data files and the execution details in a YAML format file. Standardized scientific workflows defined via CWL can be executed by CWL- compatible workflow engines, which are responsible for executing, monitoring and retrieving logs and outputs, running on top of a variety of computing platforms, ranging from individual workstations to cluster, grid, cloud, and High Performance Computing systems.

4.1 Current Limitations at *EBRAINS*

Taking into consideration how EBRAINS users interact with EBRAINS, we understand that from the scientists' point of view there are some limitations with respect to how different EBRAINS tools are described, how tools can be combined in order for scientists to create flows of data manipulations that are not executed necessarily in a specific order as pipelines, how users can execute and monitor these data flows in an opaque way, as well as how to automate some of the procedures that they often do manually.

With respect to defining data manipulation flows, Snakemake is currently a way that some scientists at EBRAINS use. This way encounters some limitations with respect to describing flows in order to be executed in a specific order, as a pipeline. The way of describing workflows executed by Snakemake, is not structured and common, and for scientists that are not already familiar with Snakemake or Python, the learning curve will be overwhelmed. As we previously described in another chapter, there are a lot of different workflow engines that exist in general for different kinds of data flows, and support different ways of describing flows of data manipulation tasks, thus a lot of different ways to define workflows.

Currently at EBRAINS, there is no standard way of monitoring large tasks that run in HPC systems and take a lot of time. Scientists often have to make requests in order to get the status and logs of the executed tasks, and restart them in case they have failed. There is also no known and standard way of pausing an execution of a task while running in an HPC system.

With respect to EBRAINS tools and services currently available inside Knowledge Graph, it is difficult for scientists to combine them together for creating acyclic graphs of data analysis and simulation tasks since tools are not always defined in an interoperable, standard and easy to be used as well as reused way. In other words, it is not straightforward for a scientist of EBRAINS to combine tools together in order to create workflows with data manipulation tasks as workflow steps.

Other than the definition, execution and monitoring of different workflows and tools, scientists would also like an easy way to combine their practical work with their theoretical work in a structured, well documented way for delivering their work to other groups of researchers from inside or outside EBRAINS Research Infrastructure. Currently, there is no way for workflows created by scientists in the scope of EBRAINS to be easily found, accessible, replicable or reproducible. Usually, data and models are the ones that can be shared inside the Knowledge Graph in order to be publicly associated with scientists' work, but often this takes time due to the curation procedures that need to be made. It would be easier and make much more sense for scientists to associate their analysis and simulation structured recipes as workflows inside the Knowledge Graph in order to be easily findable, accessible and replicable to other groups of scientists from the neuroscience community.

Last but not least, an important limitation that EBRAINS has, is the need for configuration adjustments in the description of workflows in order for them to run in different underlying infrastructures. Thus, scientists from other scientific communities need to especially configure their already defined workflows in order to be executed at EBRAINS whilst at the same time, scientists of EBRAINS would have to reconfigure their workflows if they would like to use another underlying infrastructure for testing purposes.

4.2 Why standardization is important?

Under the current subsection, we are going through why standardization is the answer to the needs of EBRAINS scientists, what the term means and what are the benefits of having technologies and tools in order to make it happen.

In general, scientists would like to use straightforward data manipulation tasks with specific inputs and output in order to reuse them in different work. Thus, tools need to be described in a standard and interoperable way as well as stored somewhere where they can be easily findable and accessible. Having tools described in a standard and interoperable way, makes definition of workflows easier for scientists. Workflows can be defined as recipes where inputs, outputs and steps are combined together in order to achieve a scientific objective. With respect to standardized workflows, it is important to be defined in a structured and common way by describing a recipe and have packaged tools as workflow steps in order to be easily combined to create directed acyclic graphs where nodes are the tasks to be executed and edges are the data flows. Standardized workflows also provide scalability to scientists, since defining a small or a very complicated workflow as a structure recipe with standard format will not increase exponentially the learning curve that scientists will have to familiarize with. In that way they will be easily findable, accessible, stored in a registry for different scientists to explore.

Also, scientists often need to restart or re-adapt parameters in their scientific work in order to achieve the correct output. For that, a tool that automates as many procedures as possible will help scientists not to be prone to errors because of the manual configurations, and will save time from automatically restarting tasks that have failed. Workflow management systems are the kind of tools that automate many procedures previously done manually, restart failed tasks [24] and take care of the logs of executed tasks [25]. The limitation comes from the fact that each workflow engine has its specific way of describing workflows, thus making the learning curve exponentially high with respect to the different engine that is currently used. There is a large list of workflow engines[37] existing already. Also in that sense, a lot of scientists do not want to have knowledge with respect to the underlying infrastructure in which some analysis or experiment of theirs will run and the different capabilities that it offers. They would like to describe their work without the need to intervene at any level with possible configurations or adaptations of their work with the underlying system or hardware. In that sense, tools are needed to create portable experiments that could run in different underlying systems that scientists need not to worry about configurations and capabilities. In that way the description of a scientific work will be decoupled by every technical detail that is needed in order to be executed. Standardization in that sense comes from the fact that this universal, structured and standard way of describing workflows also makes sense to be compatible with different workflow management systems already existing.

With that in mind, having a standardized way of defining workflows that are not strictly associated with specific workflow engines, gives the right amount of portability when it comes to different underlying infrastructures. In that way, the description of a workflow is completely decoupled from the execution of one, making scientists worry about the nature of the scientific problems and not how to execute them. This common way of describing workflows together with the portability that standardization offers, will make other scientists reproduce workflows in order to have the same results, or re-execute them in different infrastructures while getting the same results.

37 https://github.com/meirwah/awesome-workflow-engines

Last but not least, scientists in general want to publish their work and write papers that are associated with the experiments conducted by them. In other words, they would like to associate their practical with theoretical work. It would make more sense to them, if this work was well structured, documented and easily understandable by other scientists not only from the neuroscience community, but also from different research communities. In that sense, their research could be easily reproducible and replicable by others, facilitating Open and FAIR science. Thus, ways of introducing a standard, common and structured representation of their work is needed. In that sense, the necessary inputs, the steps that need to be executed to achieve a specific output, the output provided are all described in structured recipes that many can understand. Standardization in that matter comes from the fact that there is a need for a universal, common and known way of describing structured recipes as well as for a single point of reference where scientists of EBRAINS can store these structured recipes for future reference, for easier findability, accessibility and reproducibility. Scientists from different research fields, outside of EBRAINS, can also search for related work and even store their structured recipes for others to find.

4.3 Introducing standardized workflows

Due to the current limitations that EBRAINS has and after exploring different research infrastructures that already solved similar issues that were identified under the scope of EBRAINS, we introduce standardized computational workflows at EBRAINS. Standardized computational workflows play an important role since they are the connecting link between scientists' work conducted by the toolsets provided from the upper layer of EBRAINS, data, tools and services for data analysis, simulation, brain related tasks in general, as well as the powerful underlying infrastructure, FENIX ICEI, which is described as the bottom layer of EBRAINS.

Workflows for analysis and simulations created by scientists should be defined in a common, structured and standard way in order to supply well documented recipes for easier accessibility, collaboration and shareability with other researchers and communities inside as well as outside the EBRAINS.

With respect to different non-interactive EBRAINS tools found under the Knowledge Graph, a common and structured way of definition with specific input and output is also mandatory. In that way, non-interactive tools can be used as workflow steps in more than one workflows providing re usability and interoperability.

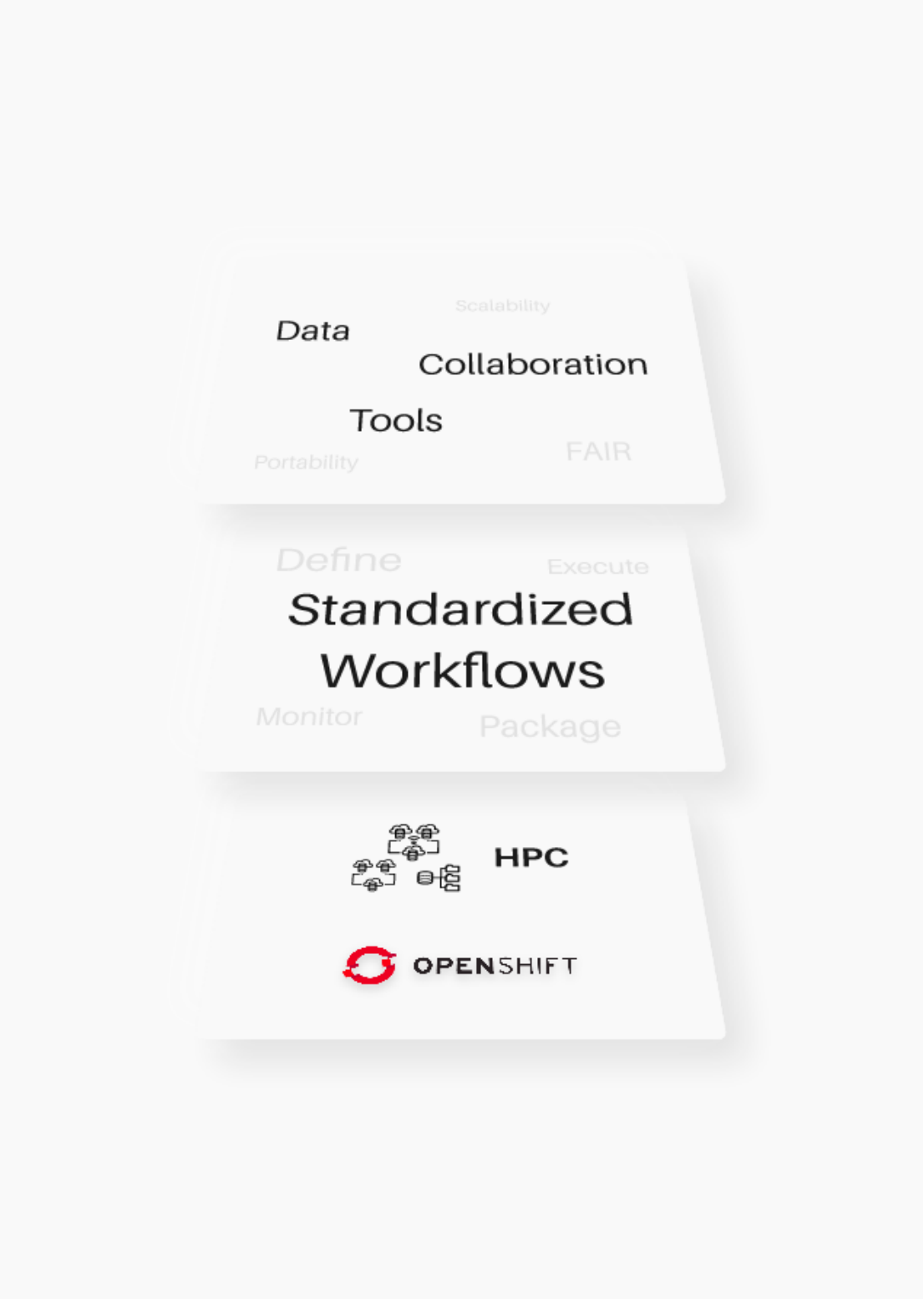


Figure 21: Standardized workflows as a middle layer for scientists to properly interact with both the upper as well as the bottom layer (Abstract). After introducing standardized workflows at EBRAINS.

Under the current thesis, we introduce standardized workflows in the middle layer of EBRAINS in order to fill in the gap that is created between the upper -where data, tools and services can be used for creating flows of data- and bottom layer – where these flows are executed via the powerful underlying infrastructure - of EBRAINS for collaborating, sharing, re-using work with other scientific communities. Researchers at EBRAINS can create workflows in a standardized way for offering outputs and outcomes valuable to the brain research, thus, making EBRAINS Open and FAIR.

Before introducing standardised workflows, EBRAINS tools were strictly associated with the EBRAINS services while at the same time they did not provide any ways for reproducibility or re usability in defining new workflows. The learning curve for a scientist could be exponentially high, since some tools were lacking the right documentation, the correct type of input as well as output data. Jupyter notebooks were used as the entry point of level for executing and monitoring workflows at HPC systems. Programmatic access via different Command Line Interface was also the case when scientists would like to access HPC for submitting, executing batch jobs strictly associated with the scheduler that was running underneath.

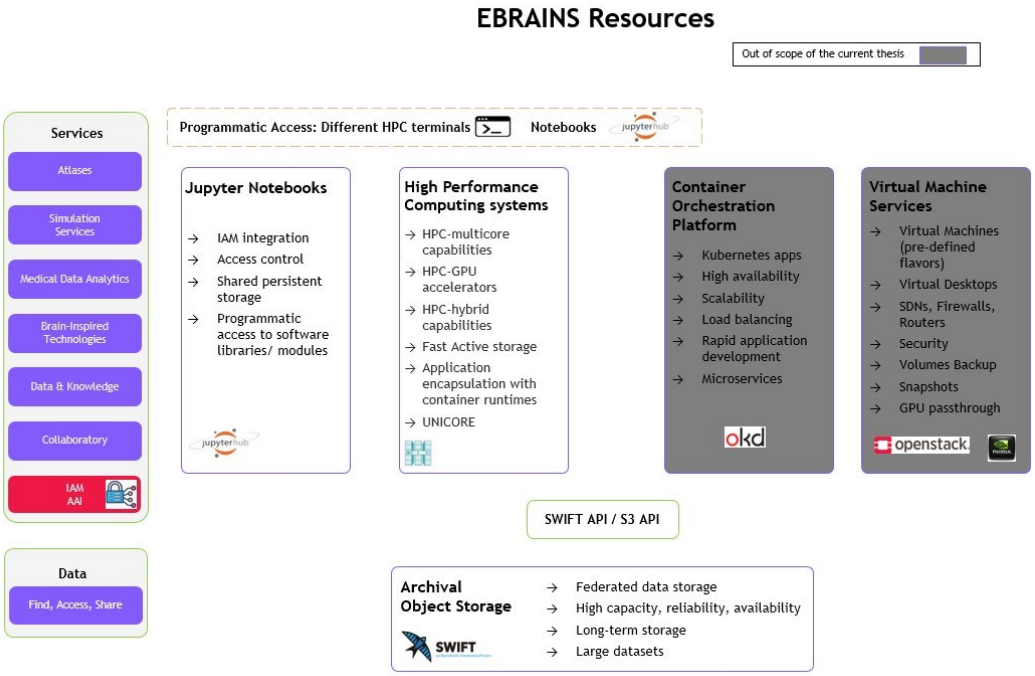


Figure 22 Before standardisation was introduced at EBRAINS via the pilot workflow management system. Jupyter Notebooks were used and direct access to HPC systems via different Command Line Interfaces was happening. Unstructured formats of defining scientific workflows was taking place by the scientists at EBRAINS.

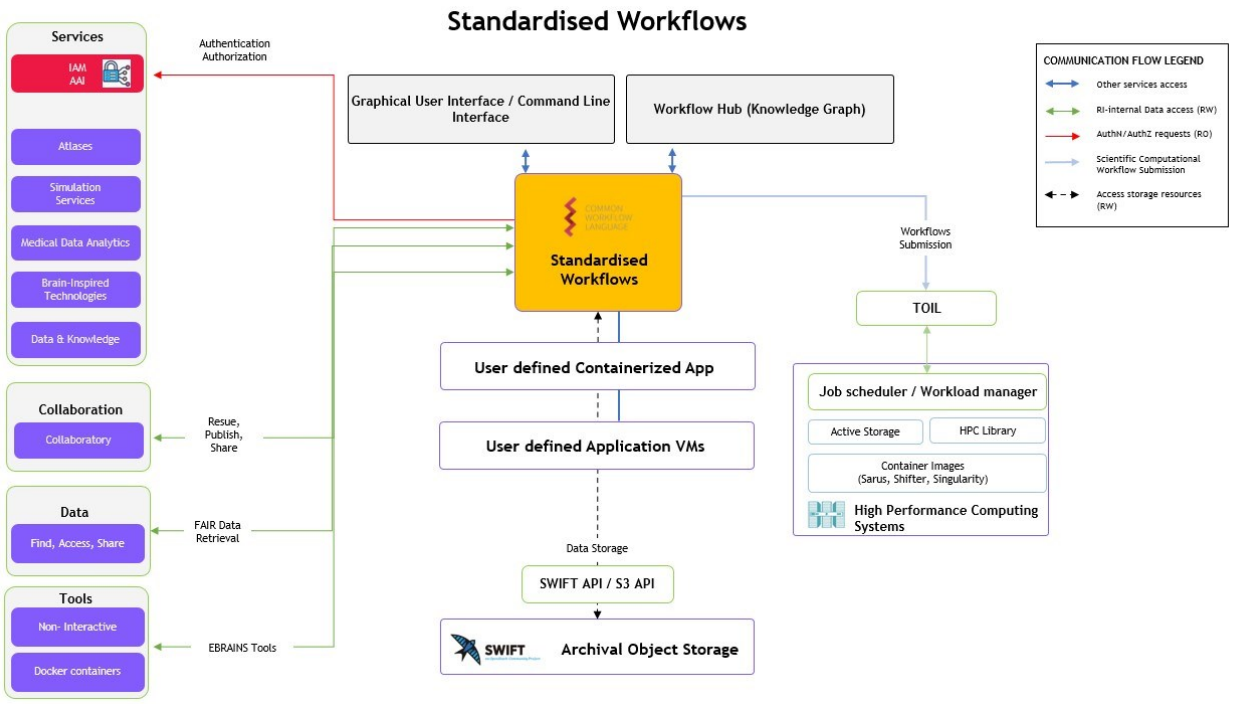


Figure 23: After standardisation introduced to EBRAINS RI via the pilot workflow management system. Standardised workflows filled in the gap between Services, Data, Tools and Collaboration (Upper Layer) and the HPC systems provided by the Bottom Layer of EBRAINS.

4.3.1 Common Workflow Language

Common Workflow Language (CWL) [1] is an emerging open standard designed to define tools and workflows for data analysis and simulation with specific input and output types, parameters and available resources needed in order to be executed for accomplishing a specific output. In that way, execution can be portable and scalable across a variety of software such as scientific workflow engines, and hardware environments, such as the cloud or HPC systems. CWL provides a standardized way to describe tools and combine them in order to create graphs, using groups of structured texts as YAML files.

Common Workflow Language is not a piece of software, but rather a specification that defines a set of standards that different implementations must conform to. It is designed to meet the needs of data intensive science, like Bioinformatics, Medical Imaging, Astronomy, Physics, Chemistry as well as Neuroscience. The vast majority of different scientific communities already use Common Workflow Language as a common and standard format because it is a common declarative format for describing tools and workflows, it can be very extensible since the development effort is laying in its large active community, and it supports containerization technologies like Singularity and Docker. In particular, Common Workflow Language is used by the European Union's BioExcel [55] Centre of Excellence for Biomolecular modelling, and by the Industrial biotechnology innovation and synthetic biology accelerator (IBISBA)[38] ESFRI for Industrial Biotechnology.

38 https://hub.ibisba.eu/

CWL is also participating in both Global Alliance for Genomics and Health (GA4GH[39]) Task Execution API[40] and GA4GH Workflow Execution API[41] projects.

CWL is compatible with a plethora of workflow engines [4] for executing and monitoring these common structured workflows and tools and work by their active community is on-going for making it compatible with even more. It also provides data locality where input and output files of a CWL workflow are modelled as rich objects with metadata and Uniform Resource Identifier(URI)/ Internationalized Resource Identifier (IRI) association. Also platforms like workflow engines that are compatible with CWL can use these URI/IRIs to send compute near the location of data or can fetch data from a remote location.

The most important asset of CWL to the scientific communities is the fact that descriptions of workflows and tools are decoupled from the execution, making definitions purely associated with scientific objectives and not with technical burdens, configurations, requirements and dependencies with different underlying infrastructures. This gives a certain amount of freedom and portability to scientists since they will not have to parameterize scientific workflows according to the different infrastructures they will run upon. Finally, describing workflows with a standard and common format offers scalability in the sense that definitions can vary from small to very complicated ones, with data manipulation steps combined to create graphs, branches or loops.

In this section we introduce ways for scientists to create structured recipes via CWL entailing data, tools and software for scientific work to be easily findable, reusable as well as associated with publications and citations. A common and standard way of describing data manipulation scientific tasks along with the data flows, offers portability, findability, accessibility and reproducibility. Portability is the sense that scientists will not need to adapt their work according to different configurations needed with respect to the underlying infrastructure, that can be a local cluster, an HPC system or a personal computer. Otherwise, miss configurations and configurations need to be taken care of by them explicitly. This makes process of defining and executing scientific work difficult, error prone and time consuming. Thus, a unified and homogeneous way of describing data manipulation workflows that can be executed effortlessly in different hardware and underlying infrastructure is needed. With respect to findability, accessibility and reproducibility, it is true that scientists wish to associate their scientific work with publications, papers and scientific magazines. By having a structured, well documented way of introducing their work to familiar as well as other scientific fields and communities, they can make their work able to be reproduced in a seamless and feasible way.

39 https://www.ga4gh.org/

40 https://github.com/ga4gh/task-execution-schemas

41 https://github.com/ga4gh/workflow-execution-service-schemas

4.3.1.1 CWL Workflow example

Below we present a simple example of a workflow described via CWL formatting. The workflow used in this example is part of the SOFA/AmbiX Binaural Rendering (SABER)[42] toolkit. SABRE toolkit is a collection of MATLAB functions that allows users to create custom binaural decoder presets for Matthias Kronlachner's ambiX binaural plug-in. This workflow[43] is used to perform segmentation and cell detection in X-ray volumes.

```yaml
cwlVersion: v1.0
class: Workflow

inputs:
    # i/o
    data: File
    membrane_classify_output_name: string
    cell_detect_output_name: string
    vessel_segment_output_name: string
    bucket: string
    # Cell detect
    detect_threshold: float?
    stop: float?
    initial_template_size: int?
    detect_dilation: int?
    max_cells: int?
    classifier: File
    # Membrane classify
    ram_amount: int?
    num_threads: int?
    segment_threshold: float?
    segment_dilation: int?
    minimum: int?

outputs:
    membrane_classify_output:
        type: File
        outputSource: membrane_classify/membrane_probability_map
    cell_detect_output:
        type: File
        outputSource: cell_detect/cell_detect_results
    vessel_segment_output:
        type: File
        outputSource: vessel_segment/vessel_segment_results

steps:
    membrane_classify:
        run: ../tools/membrane_classify.cwl
        in:
            bucket: bucket
            input: data
            output_name: membrane_classify_output_name
            classifier: classifier
            ram_amount: ram_amount
            num_threads: num_threads
        out: [membrane_probability_map]
    cell_detect:
        run: ../tools/cell_detect.cwl
        in:
            bucket: bucket
            input: membrane_classify/membrane_probability_map
            output_name: cell_detect_output_name
            threshold: detect_threshold
            stop: stop
            initial_template_size: initial_template_size
            dilation: detect_dilation
            max_cells: max_cells
        out: [cell_detect_results]
```

Figure 24:Scientific workflow part of SOFA/Ambix Binaural Rendering toolkit defined via Common Workflow Language format.

42 https://github.com/PrincetonUniversity/3D3A-SABRE-Toolkit

43 https://github.com/aplbrain/saber/blob/master/saber/xbrain/workflows/xbrain.cwl

First, a general script information section is provided:

```
cwlVersion: v1.0
class: Workflow                                                    ..
```

These fields are mandatory for the description of workflows as well as tools described in the previous section. They provide the class of the file, which in this case is Workflow, as well as the version of CWL so that the CWL compatible engine used for the tool execution can check for compliance. A short description of the workflow is an optional choice in order for other users to know basic information about the workflow, such as what are the inputs and the provided output.

With respects to parameters:

```
inputs:
  data: File
  membrane_classify_output_name: string
  cell_detect_output_name: string
  vessel_segment_output_name: string
  bucket: string
  (..)
```

The input parameters of the workflow are defined. Each input parameter definition includes the parameter name as well as the input type like string, number, boolean, File, Directory.

As for the outputs,

```
outputs:
  membrane_classify_output:
    type: File
    outputSource: membrane_classify/membrane_probability_map
  cell_detect_output:
    type: File
    outputSource: cell_detect/cell_detect_results
  (…)
```

In this format, the outputs that should be produced by the workflow are listed. Each output object definition includes the parameter name, the output type and, in the case of Files, the output file path.

With respect to the different workflow steps, a graph is constructed where nodes are the tasks that need to be executed and the edges are the data flowing from one node to another.

```
steps:
  membrane_classify:
    run: ../tools/membrane_classify.cwl
    in:
      bucket: bucket
input: data
output_name: membrane_classify_output_name
classifier: classifier
ram_amount: ram_amount
num_threads: num_threads
    out: [membrane_probability_map]
  cell_detect:
```

```
    run: ../tools/cell_detect.cwl
    in:
      bucket: bucket
input: membrane_classify/membrane_probability_map
output_name: cell_detect_output_name
      threshold: detect_threshold
      stop: stop
      initial_template_size: initial_template_size
      dilation: detect_dilation
      max_cells: max_cells
    out: [cell_detect_results]
  (..)
```

All tools that are used as steps of the workflow are defined via a CWL tool description. Each input or output entry is essentially a mapping from the workflow inputs/outputs fields to the tool input/output names values. Having all the information on each workflow step's inputs and outputs, makes it possible for the different execution engines to deduce the directed acyclic graph (DAG) of tasks that the workflow represents, and thus execute the steps in the correct order, or even in parallel when no dependencies exist between them.

4.4 Introducing standardized non interactive EBRAINS tools

In the scope of this thesis, we introduce non-interactive EBRAINS tools that are associated with data manipulation processes like analysis and simulation and can be used as steps in scientific workflows defined via CWL format. It is critical for each EBRAINS tools, described via CWL, to provide a concrete description of what it accomplishes, what are the input parameters and the type of the expected output.

Non-interactive tools can be defined as pieces of software that run programs in which users will interact with before the runtime. In that way, users will parameterize the inputs before the execution of the tool. On the other hand, interactive tools can be defined as pieces of software where users interact with the programs during the runtime in order to parameterize parameters on-the-fly. For the sake of standardization, we are mostly interested in non-interactive tools in the scope of EBRAINS RI.

Tools are usually associated with their dependencies, libraries and binaries, so it is important to have a packaging method in order to be easily (re-)used as workflow steps in scientific workflows in a flexible and resistant way. In the next subsections, we propose packaging EBRAINS non-interactive tools via Docker containerization method, pushing them into Harbor EBRAINS docker registry and storing them under EBRAINS Knowledge Graph for easy findability, accessibility and reusability.

4.4.1 Packaging EBRAINS tools with Docker containerization method

For the current pilot workflow system in the context of EBRAINS, we propose Docker as the method for non-interactive EBRAINS tools to be packaged together with dependencies, libraries and binaries needed in order for tools to be easily executable [38].

In general, software dependencies management and project isolation can be solved by deploying EBRAINS tools inside an isolated environment, such as a container. Container technologies utilize the host's kernel and thus have the ability to run contained applications with the same performance characteristics as native applications. The initial inception of containers shares a direct lineage from virtual machine technology, but is sometimes considered more efficient than the latter. Containers are isolated processes, with their own resources, network and file system hierarchy and are widely used for packaging applications together with all configurations needed in order to run. Containerized applications are portable with respect to the underlying infrastructure, which can be a local machine, a virtual machine, a supercomputing cluster or a cloud, providing the right levels of portability.

Specifically, Docker [45] is an open platform for developing and running applications isolated from the underlying infrastructure by using container technology. Docker, as well as every other containerization technology, offers versioning of different software, thus, making it possible to roll back to previous versions of the same software in order to deliver a previous output.

With respect to combining containerization methods with CWL, non-interactive EBRAINS tools packaged via Docker can be executed individually by CWL compatible workflow engines or can be used as workflow steps when workflows are described. This ensures software portability as well as software reusability, and thus, it is considered the best practice among scientific fields focusing on FAIRness and Openness.

As stated above, Docker containers can significantly simplify software installation and version control by providing a complete and pre-tested runtime environment for software and its dependencies. In CWL in particular, Docker containers are used for individual executions of tools that exchange files. EBRAINS tools wrapped in containers create a new temporary space that is removed after the execution of each tool or when the whole workflow is finished. In that space, bind mounts expose only the required working directory of that particular tool, having read only permissions on the host input files. CWL compatible workflow engines can handle automatically that input files will exist inside a Docker container for software to properly run, as well as output files will be mounted to the local filesystem and be removed from the container once the tool or the whole workflow runs. Since CWL takes care of input and output files, developers only need to wrap tools with their dependencies as separate containers. In that way, the complexity of invoking and managing Docker containers is avoided while tools or workflows with combined tools as steps are to be executed[44].

Although Docker is supported by a plethora of infrastructures, it is not supported by the HPC systems existing in the underlying infrastructure at EBRAINS. On top of HPC systems, Singularity takes over and support the execution of Docker containerized EBRAINS tools. From a developer's point of view, no changes are needed with respect to defining tools or workflows via CWL format. The primary workflow engine responsible for executing and monitoring the execution of workflows and tools, need to be compatible with

44 https://www.commonwl.org/user_guide/07-containers/index.html

Singularity containerization method. In that way, workflow engines compatible with Docker or Singularity running in different underlying infrastructures, can execute effortlessly the defined scientific workflows having as steps packaged EBRAINS tools via Docker.

4.4.1.1 Dockerfile

In order to package a tool, a simple Dockerfile is needed. Dockerfile is a read only text document template with instructions that contains all the commands a user should run to assemble a Docker image. A Docker image is a lightweight, standalone executable package of software that includes everything needed to run an application from the code, libraries, binaries, and other packages. This Docker image will be built and pushed into the Harbor EBRAINS docker registry and will become a container at runtime. All the following instructions can be applied to systems that have Docker already installed. A simple example is presented:

```
FROM python:3

# install dependencies
RUN pip install argparse numpy pandas matplotlib

# copy python script, make executable and add to path
COPY visualization.py /home/tool/visualization.py
RUN chmod +x /home/tool/visualization.py
ENV PATH="/home/tool:$PATH"

CMD [ "/bin/bash" ]
```

All Dockerfiles must start FROM a base image, like Ubuntu, Linux distribution or even more specifically made images for Python or Java and usually ends with the CMD (command) that will run the application itself. In the example above, the tool that needs to be containerized is a single python executable script. The base image used is python:3. The first step is the installation of all software dependencies which in this case are some python packages. Then, the tool needs to be installed in the new isolated environment by copying the file to the new filesystem, making it executable and adding its location to the $PATH environment variable.

Once the Docker image is ready, it can be built by using the following command:

```
docker build -t <name[:tag]> -f </path/to/Dockerfile> <PATH|URL|>
```

In our specific example, we will build the psd_workflow_fetching_data container image that will used later on in the Example [0] subsection.

```
docker build -t psd_workflow_fetching_data:latest
 -f </path/to/Dockerfile> <PATH|URL|>
```

The docker build command takes as input the image name and tag, the location of the Dockerfile and a PATH, which can be a local directory or a URL like a Git repository location that defines the build context. The Docker build context refers to the files and directories that will be available to the Docker engine.

4.4.1.2 Docker registries

As a last step, the already build Docker image needs to be pushed in one of the available Docker registries. DockerHub, is a public hub that works as a service for finding, storing, sharing and accessing container images publicly or privately between teams.

Users first need to log into DockerHub with their credentials, tag their container images and then push container images to DockerHub. At this point, only the specification of the images is pushed and not the container itself.

```
docker login
docker tag <old_name> <new_name:tag>
docker push <new_name:tag>
```

In our example, the command will look like:

```
docker push psd_workflow_fetching_data:latest
```

Specifically, for EBRAINS RI, a dedicated docker registry hub, Harbor, for registering, finding and accessing Docker images already exists. An EBRAINS user can login from the command line interface using the following docker command:

```
docker login docker-registry.ebrains.eu
```

In order to push an image to the registry, the user first has to tag it appropriately so that the new name corresponds to its location inside the registry.

```
docker tag <old_name:tag>
<docker-registry.ebrains.eu/project_name/new_name:tag>
```

In our example, the commands will look like:

```
docker tag psd_workflow_fetching_data:latest
docker-registry.ebrains.eu/tc/cwl-workflows/psd_workflow_fetching_data:
latest
```

Finally, pushing an image to the registry can be done using the docker push command:

```
docker push <new_name:tag>
```

In our example, the command will look like:

```
docker push docker-registry.ebrains.eu/tc/cwl-workflows/psd_workflow_fetching_data: latest
```

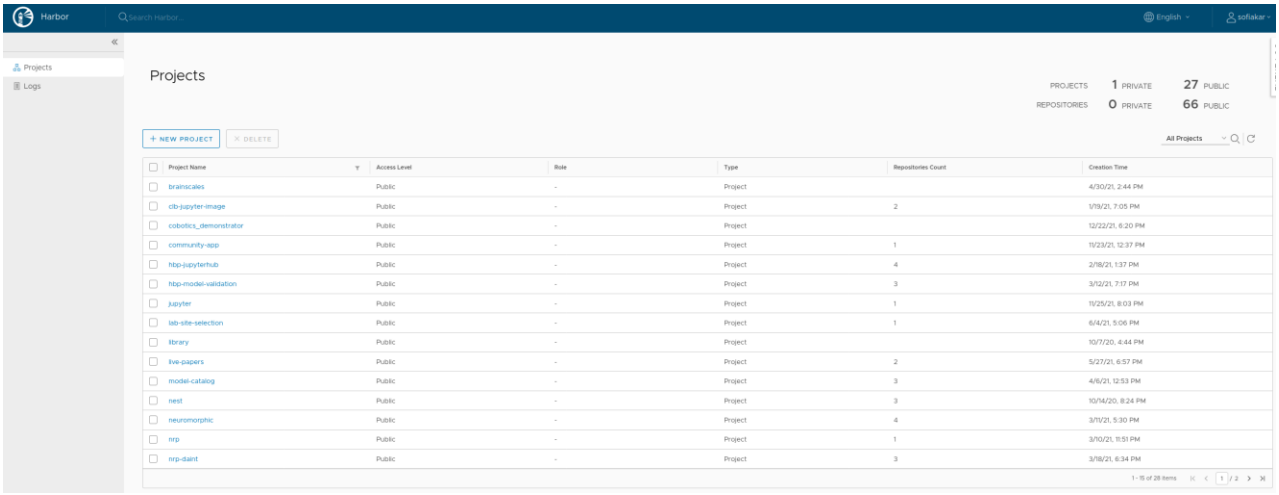


Figure 25: Harbor docker registry available for all EBRAINS users. Non-interactive EBRAINS tools are already packaged via Docker containers and stored here for future reference in the workflow descriptions.

4.4.2 CWL EBRAINS tool example

In this subsection, we present a real example of a definition of a non-interactive EBRAINS tool via CWL previously packaged via Docker. As previously stated, a command-line tool is defined as a piece of software that carries out a specific computational task and runs as a non-interactive program that terminates upon task completion. The CWL Command Line Tool description specification provides a common layer for describing the syntax and semantics of programs. Each tool is documented, along with all its parameters, inputs and output files, software dependencies and execution details in a YAML format file so that they can be shared across platforms and linked with public registries and hubs to form publications. This way, CWL tool descriptions can essentially turn POSIX command-line data analysis tools into user-defined functions with explicitly specified inputs and outputs. All the information necessary to run the tool is encoded in a single file, so that even users with no knowledge or understanding of its structure and functionality can use it for their computations. When it comes to the execution part, all tool software dependencies and runtime requirements are listed and known a priory. Each tool will be executed independently in a well-defined, isolated environment via Docker or Singularity method.

```
#!/usr/bin/env cwltool

cwlVersion: v1.0
class: CommandLineTool
baseCommand: fetching_data.py
hints:
  DockerRequirement:
    dockerPull: docker-registry.ebrains.eu/tc/cwl-workflows/psd_workflow_fetching_data:lates
  ResourceRequirement:
    ramMin: 2048
    outdirMin: 4096
inputs:
  bucket_id:
    type: string
    inputBinding:
      position: 1
  object_name:
    type: string
    inputBinding:
      position: 2
  token:
    type: string
    inputBinding:
      position: 3
outputs:
  fetched_file:
    type: File
    outputBinding:
      glob: $(inputs.object_name)
```

Figure 26 An EBRAINS tool defined via Common Workflow Language. The definition consists of 4 main sections. The tool is already packaged via Docker for reusability and reproducibility reasons and stored inside Harbor EBRAINS Docker registry for easy accessibility by EBRAINS users.

First, as seen in the Figure above, a general script information section is provided:

```
cwlVersion: v1.0
class: CommandLineTool
description: “This tool fetched data from Data Proxy by providing the name of bucket (string), the name of the file as an object (object_name), and the user token. The output file is the desired datafile.”
```

This section is mandatory for all CWL files and should include the script class which is CommandLineTool in case of a tool description. A short description of the tool is an optional choice for other users to know basic information of the tool, such as what are the inputs and the provided output. The version of the CWL specification (v1.0) should also be labelled, so that the CWL compatible engine used for the tool execution can check for compliance.

Then, a base command information is provided:

```
baseCommand: fetching_data.py
hints:
  DockerRequirement:
    dockerPull: psd_workflow_fetching_
  ResourceRequirements:
    ramMin: 2048
    outdirMin: 4096
```

This section contains the base command and hints connecting to the execution environment of the tool in order to run. The hints have to do with specific requirements like Docker image and Resources that are needed in order for the tool to be executed. Specifically, DockerRequirement is used to define the Docker image that is necessary for running the specific tool and should be pulled from the URL specified by the dockerPull tag. The dockerPull parameter takes as value the name of the container image as well as a specific tag.

In this example, we have already created the workflow_fetching_data Docker container image and pushed it into the Harbor EBRAINS docker registry, for future reference as well as for accessibility and findability by other users. There are also some fields dedicated to ResourceRequirements in order for the CWL compatible engine to know beforehand how many resources are needed in order for the tool to run. This is not a mandatory field, but will make the execution faster since workflow engines will typically use the minimum resources available and will keep increasing them until the execution has ended properly.

As a next step, inputs of the command line tool are provided:

```
inputs:
 bucket_id:
    type: string
    inputBinding:
      position: 1
  object_name:
    type: string
    inputBinding:
      position: 2
  token:
    type: string
    inputBinding:
      position: 3
```

For the tool to run, all required input parameters must be associated with specific values. Each input parameter definition should include the input type such as string, number, boolean, file, directory as well as its inputBinding which describe how to turn the input parameter into a command line argument. This can be described as the definition of the input types. Another YAML file stores the actual input fields that are used in order for the tool to be executed.

Scientists wishing to use this packaged tool defined via CWL as a step in another scientific workflow, will only need to care about adhering to different types of the inputs and provide the YAML file with the exact parameters needed.

```
bucket_id: 'fetching-data-from-bucket-and-kg'
object_name: 'data_mat/sub-1_chs-32_hem-RH_ana-ISO000_stim-SPN.mat'
token: 'COPY_TOKEN_HERE'
```

Figure 27: A YAML file consisting of the parameter inputs of the EBRAINS tool defined via Common Workflow Language.

Lastly, output of the tool is defined as:

```
outputs:
 fetched_file:
   type: File
   outputBinding:
   glob: $(inputs.object_name)
```

In the Outputs section, the names and types of the tool's outputs are defined, similarly to the input parameters. If the tool's standard output or standard error needs to be captured, this can be done by specifying the stdout or stderr object as an output respectively.

4.5 Workflow management system

With respect to executing standardized scientific workflows, a workflow management system is needed in order to monitor workflow steps, handle possible failures when they occur, retrieve logs and outputs and overall make procedures automatic.

In this subsection, we will propose a small number of workflow management systems that could be installed on top of HPC systems which offered by the EBRAINS underlying infrastructure. Since there are a lot of workflow engines that exist, we narrowed down the list to a fewer number of systems taking into consideration some critical aspects:

- First, a key point for selection was the level of compatibility with CWL that is used as standard and common way to describe scientific workflows.
- Second, workflow management systems must be easily installed on top of HPC systems and make use of the underlying workload manager running for scheduling of the tasks. In the current thesis, Slurm is used as the workload manager in the majority of the HPC systems existing in the EBRAINS underlying infrastructure.
- Third point, workflow management systems need to be compatible with different containerization methods such as Docker and Singularity, since Singularity is used as the containerized method on top os HPC systems.

In the table below, we conclude our metrics for choosing the most suitable workflow management system for our purpose [33].Under Underlying infrastructure column, we provide infrastructures in which workflow management systems can run upon. Under Compatibility with CWL column we present if workflow management systems are compatible with Common Workflow Language formatting, and lastly under Containerization method we provide which of the containerization methods, Docker, Singularity or both are supported during the execution of different steps in workflows.

	Underlying Infrastructure	Compatibility CWL	Containerization method
Nextflow	GridEngine, SLURM, LSF, PBS, HTCondor, Moab, Kubernetes, GCP, AWS, Azure, locally	no	-
Snakemake	Containers, HPC, cloud, locally	no	Docker, Singularity
CWL-Airflow	Linux	yes	Docker, Singularity
Toil	GridEngine, SLURM, LSF, PBS, HTCondor, Mesos, GCP, AWS, Azure, OpenStack, locally	yes	Docker, Singularity
CWL-TES	AWS, GCP, Alicloud, HPC, Spark, TES, locally	yes	Docker
CWLTool	locally, cluster, HPC	yes	Docker, Singularity

As seen in the table, Nextflow provides the minimum compatibility with CWL and thus it is not a candidate for becoming the primary workflow management system in the context of EBRAINS RI. Snakemake also provides small compatibility with CWL in the sense that users need to shift the workflow description in such a way that Snakemake can execute. On the other hand, CWL-Airflow [15], although compatible with CWL, is not supported on top of HPC systems. Lastly, CWL-TES client does not support Singularity as the primary containerization method.

By eliminating some of the workflow engines, we narrowed the candidates into two workflow engines, CWLTool and Toil, which adhere to all three **(3) aspects** for selecting a management system for the pilot EBRAINS workflow system. Both workflow management systems run on top of HPC systems and can take care of the workload manager that runs underneath (Slurm workload manager). They are compatible with Common Workflow Language in the level where no configurations are needed neither on the workflow definition nor during the execution of the workflow. Last but not least, both support Docker and Singularity as containerization methods in order to execute tasks of the workflow as packaged isolated steps.

4.5.1 CWLtool

Cwltool is the reference implementation of the Common Workflow Language, created and maintained by the CWL team. It is the most feature-complete and lightweight implementation of CWL. All workflow inputs in the specific execution of the workflow recipe must be specified in a YAML or JSON file. The command syntax to run a workflow is very simple:

```
cwltool <workflow_descr> <inputs_obj>
```

where <workflow_descr> is the CWL workflow or the command line tool description and <inputs_obj> is the file with the inputs parameters as previously described.

While there is a wide range of options that can configure the way the workflows are executed, we chose to mention some of the available options.

In order to run a workflow without any containerized method, in case all software and file requirements are met in the local environment, the --no-container flag can be used:

```
cwltool –-no-container <workflow_descr> <inputs_obj>
```

As previously mentioned, most HPC systems provide Singularity as a containerized method, that can be used to build Docker containers. In order to use Singularity runtime for running containers, we can use the --singularity flag:

```
cwltool –-singularity <workflow_descr> <inputs_obj>
```

4.5.2 Toil

Toil [34],[29] is an open-source, portable, scalable workflow engine that supports many different contemporary workflow definition languages, including CWL, and can be used to run scientific workflows at large-scale efficiently, securely and reproducibly. It can run on Amazon Web Services (AWS), Google Compute Engine (CGE) and Kubernetes, and also supports various batch systems (SLURM, Torque, LSF).

The Toil runner command provides cwl-parsing functionality using cwltool and leverages the job scheduling and batch system support of Toil. To run in local mode, the user needs to provide the CWL file and the input object file:

```
toil-cwl-runner <workflow_descr> <inputs_obj>
```

To use Singularity runtime for running containers, we can use the `--singularity` flag:

```
toil-cwl-runner --singularity <workflow_descr> <inputs_obj>
```

To use Slurm workload manager on top of HPC systems, we can use the `–-batchSystem` flag:

```
toil-cwl-runner –-batchSystem slurm <workflow_descr> <inputs_obj>
```

4.6 Scientific computational workflows at EBRAINS via Pilot Workflow Management System

Using the same real example as was used in chapter [3.4], we will prove how standardised workflows introduced at EBRAINS RI throughout the pilot workflow management system, reduced the number of steps that the EBRAINS users would have to do manually, as well as centralizing all actions into a single point of truth, which were previously done by a number of different terminals and HPC endpoints. Currently, the scientific computational workflows as well as EBRAINS tools used as workflow steps are described in an open, common, broadly understandable way via Common workflow Language.

4.6.1 Getting data near the compute nodes

Before introducing the pilot workflow management system at EBRAINS, an EBRAINS user would have to manually ssh and copy (rsync, cp, mv) the data inside the HPC system in which the analysis or simulation will run. Now the only thing that is needed is an IRI/ URL of the remote location of where the data is stored. The workflow engine that is responsible for the execution of the scientific workflow inside the HPC system will be responsible to fetch the data prior the workflow execution into the right location. The EBRAINS user will only need to define the type of different input data needed and associate them with the appropriate values located in a separate file (YAML input file).

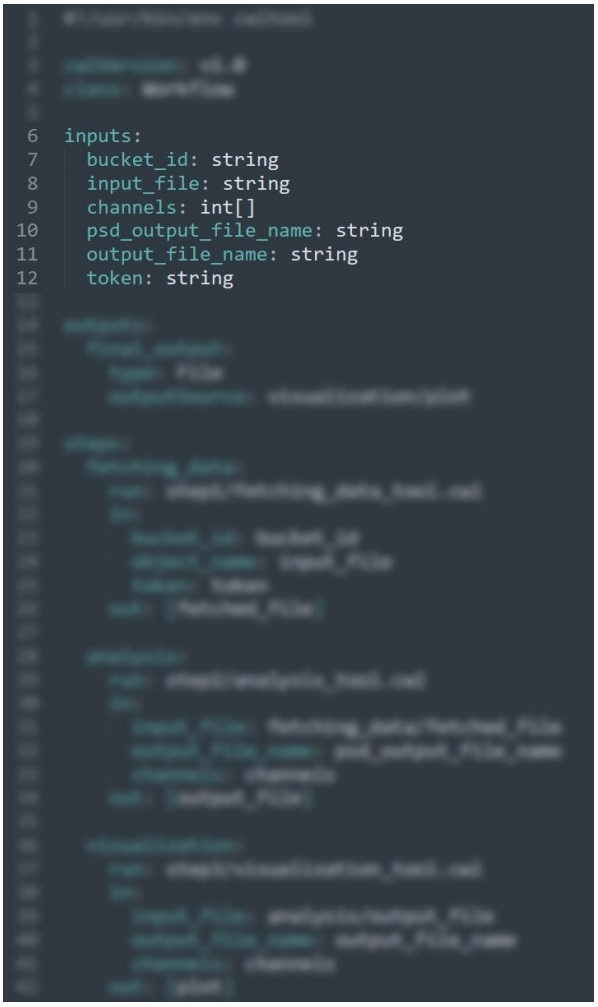


Figure 28: EBRAINS user must define the type of input data (inputs) in the CWL workflow specification (mandatory field).

Figure 29: The actual values of the input data are provided by another file (YAML input file). For the specific example, a URL of the File is provided.

4.6.2 Find tools for analysis and simulation

With respect to Finding Tools, while tools are authored via CWL they will adhere to some specifications in order to be added in EBRAINS Knowledge Graph by the curation team. There will be a documentation, description of what the tool is and how it is used. The type of the input data will be explicitly defined and maybe some test data can be associated with it in order to be easily used.

Figure 30: After pilot workflow management system is introduced to EBRAINS, users need to wrap analysis code into Docker containers and store them inside Harbor (reproducibility).

In the previous example, EBRAINS users would need to write sbatch jobs in a specific way or use Jupyter notebooks in order to better communicate with the HPC systems or even direct ssh into them to submit analysis and simulation tasks. Now, EBRAINS users would only need to package their code (python code for this example) in a Docker container and use the URL of the container that is pushed to the Harbor, the EBRAINS docker registry. Workflow engine will be responsible to fetch the Docker image to where the data analysis / simulation will take place. No slurm job specifications such as sbatch jobs is needed to be provided from the EBRAINS users.

EBRAINS tools defined via CWL will look like this:

```
#!/usr/bin/env cwltool

cwlVersion: v1.0
class: CommandLineTool
baseCommand: analysis.py
hints:
  DockerRequirement:
    dockerPull: docker-registry.ebrains.eu/tc/cwl-workflows/psd_workflow_analysis:latest
  ResourceRequirement:
    ramMin: 2048
    outdirMin: 4096
inputs:
  input_file:
    type: File
    inputBinding:
      position: 1
  output_file_name:
    type: string
    inputBinding:
      prefix: --output_file
      position: 2
  channels:
    type: int[]
    inputBinding:
      prefix: --channels
      position: 3
outputs:
  output_file:
    type: File
    outputBinding:
      glob: $(inputs.output_file_name)
```

Figure 31 After proposing the pilot workflow management system, EBRAINS tools are described via CWL format adhering to specifications. Code previously written in sbatch jobs for scheduler in HPC systems to execute it, now is wrapped inside a Docker container and stored in Harbor.

4.6.3 Use HPC systems to run experiments

From a dedicated Command Line Interface (CLI), EBRAINS users will only need to execute:

```
toil-cwl-runner --setEnv PATH=$PATH --disableCaching --batchSystem=slurm --singularity workflow.cwl workflow_info.yml
```

There is no need for Jupyter Notebooks to be used for executing and monitoring workflows after the introduction with the pilot workflow management system. Also, different programmatic access via terminals to different HPC systems is limited to one CLI for submitting the scientific computational workflows while at the same time observing logs and outputs.

4.6.4 Move output back to Archival Data Repository

Once the output is ready, an easy way to retrieve the data back is by explicitly defining the type in the workflow description, once again done by the CWL specification format.

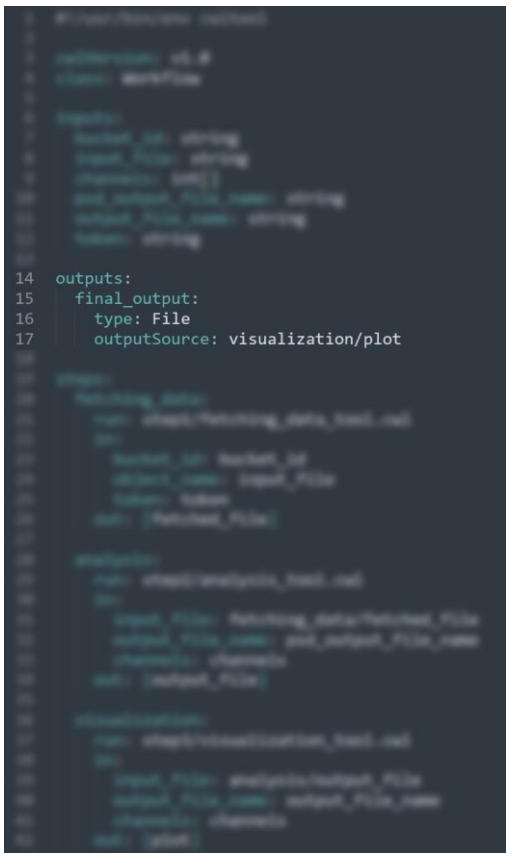


Figure 32: Exactly like the input data (inputs), the output data (outputs) are also defined in the CWL specification from EBRAINS users.

After introducing the pilot workflow management system to EBRAINS, the whole workflow will look like this:

```
#!/usr/bin/env cwltool

cwlVersion: v1.0
class: Workflow

inputs:
  bucket_id: string
  input_file: string
  channels: int[]
  psd_output_file_name: string
  output_file_name: string
  token: string

outputs:
  final_output:
    type: File
    outputSource: visualization/plot

steps:
  fetching_data:
    run: step1/fetching_data_tool.cwl
    in:
      bucket_id: bucket_id
      object_name: input_file
      token: token
    out: [fetched_file]

  analysis:
    run: step2/analysis_tool.cwl
    in:
      input_file: fetching_data/fetched_file
      output_file_name: psd_output_file_name
      channels: channels
    out: [output_file]

  visualization:
    run: step3/visualization_tool.cwl
    in:
      input_file: analysis/output_file
      output_file_name: output_file_name
      channels: channels
    out: [plot]
```

Figure 33: CWL workflow with input data, different workflow steps also defined via CWL specification with code wrapped inside Docker containers and stored into Harbor, as well as pre-defined output data for retrieving it back once ready.

4.7 EBRAINS Graphical User Interface / EBRAINS HUB

In the previous sections we introduced means for scientists to describe standardized scientific workflows by using Common Workflow Language (CWL) format and execute them via appropriate compatible workflow management systems. In this section, we propose ways for scientists to submit the already described CWL workflows into a dedicated entry point in order to monitor the execution, have access to logs and outputs. Thus, an EBRAINS graphical user interface dedicated to users is proposed. Furtherer more, a central place for storing, accessing and sharing scientific workflows among scientists and different scientific communities is mandatory.

For the pilot EBRAINS workflow management system, we propose a user dedicated workspace to be created in order for scientists to have an overview of the different workflows that can be executed, submit the ones that they would like to execute on top of the underlying infrastructure, check for logs and the available outputs. It will also be possible for them to parameterize the input fields of the workflows for the sake of their scientific objectives.

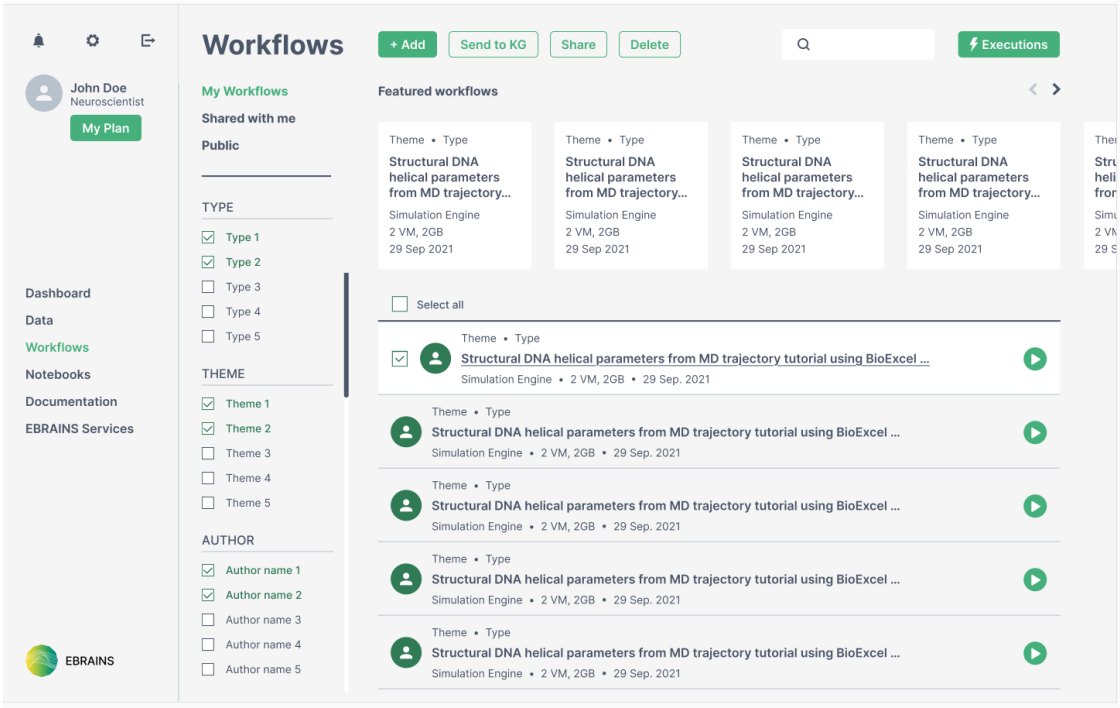


Figure 34:EBRAINS graphical user interface - Overview of the available workflows

.

Scientists will be able to check for current and previous executions of scientific workflows submitted and they will get notified once outputs are available. Logs for submitted workflows will be available for users to check.

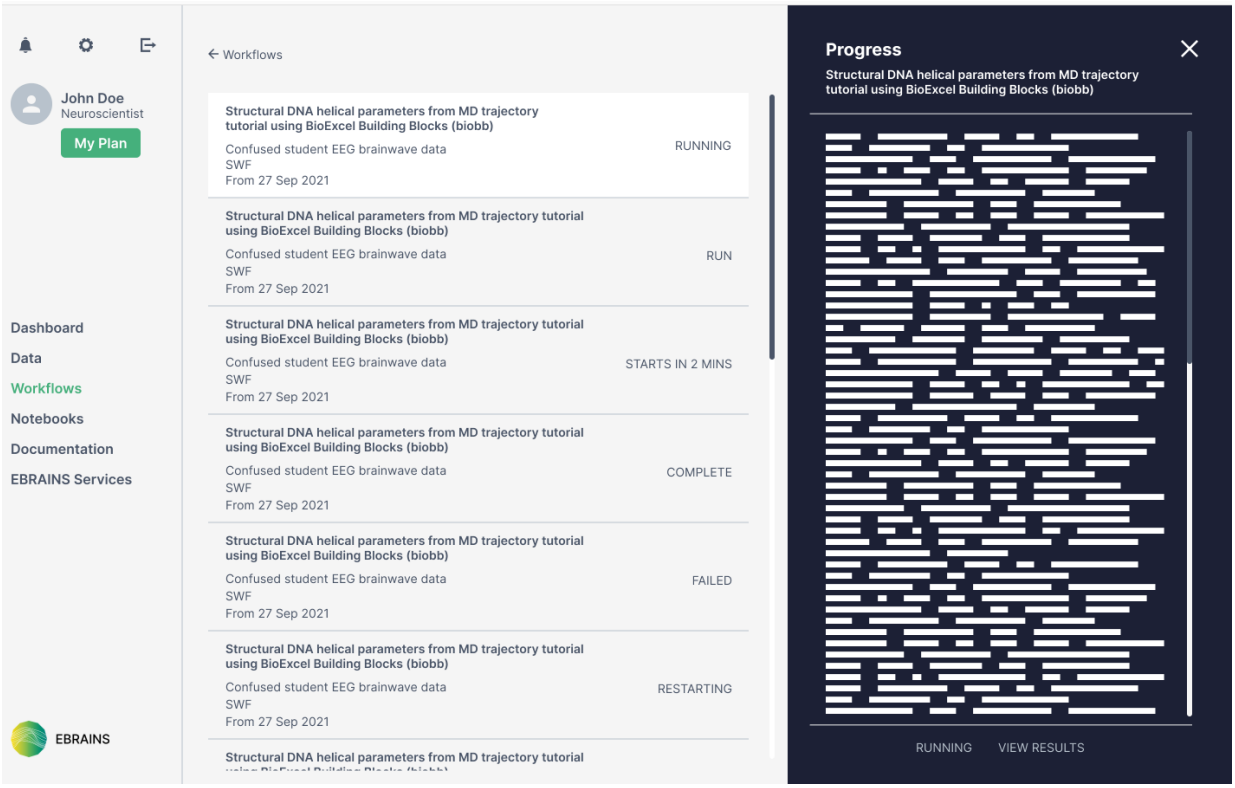


Figure 35:EBRAINS graphical user interface - On going, completed and failed executions are shown to the users. Logs for every workflow submitted may be available. Users will get notified once a workflow is completed.

EBRAINS Research Infrastructure already offers the EBRAINS Knowledge Graph, which is a multi modal metadata store that combines information from different fields on brain research, such as data, models and software existing at EBRAINS. Everything shared publicly in Knowledge Graph is annotated with standardized metadata tailored for neuroscience, facilitating discovery and reused by the broader research community. Therefore, for the pilot EBRAINS workflow management system we propose Knowledge Graph to be the central place for scientists to find, access and share standardised workflows and tools described via CWL. With the initialization of this scientific workflow and tool hub, OPENness, FAIRness, findability, shareability and reproducibility aspects are introduced to EBRAINS, such as in every other scientific related research [41]. The EBRAINS packaged non-interactive tools as well as the scientific computational workflows should be well defined in Common Workflow Language with all the documentation, inputs and outputs and should be publicly available. In that way, browsing and finding different tools to be used as workflow steps should be possible for scientists at EBRAINS as well as to external users and scientists, in order to conduct their own workflow definitions accomplishing scientific objectives using tools and data provided by EBRAINS. Thus, EBRAINS is introduced as a research infrastructure in which findable and accessible aspects are met.

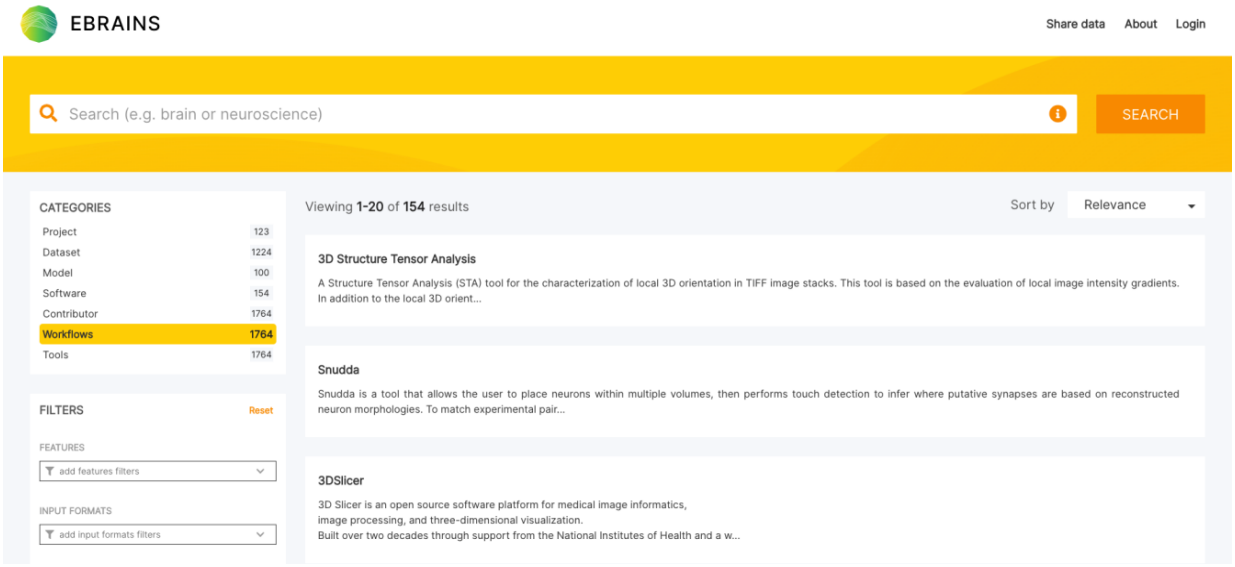


Figure 36: EBRAINS Knowledge Graph as the central HUB for CWL workflows and Tools. Scientists at EBRAINS and externals can browse, search and find standardized scientific workflows and tools defined via CWL in the scope of EBRAINS RI.

Both workflows and tools when stored in Knowledge Graph as digital objects are associated with unique identifiers. This makes it easier for scientists to refer their work with citations and publications so they can connect their practical with theoretical work. Especially for journal EBRAINS users, a dedicated page[45] describes the procedure of a pre-reviewed journal publication from which they can choose the level of metadata or data information to be made publicly available.

Specifically for EBRAINS users, they can store their workflow structured recipes in either their own private spaces in EBRAINS Knowledge Graph or in a public space. In the former, only a small number of EBRAINS users with specific permissions can have access to workflow recipes and tools. In the latter, users would need to share the CWL definitions with all EBRAINS users as well as external ones, after submitted the descriptions to the curated team for approving or denying them. EBRAINS users can submit and execute the available workflows in the EBRAINS underlying infrastructure in accordance with the available resources that are needed and are available. EBRAINS users could compose workflows by using EBRAINS tools found inside the EBRAINS Knowledge Graph via the Rabix composer.

45 https://ebrains.eu/service/share-data

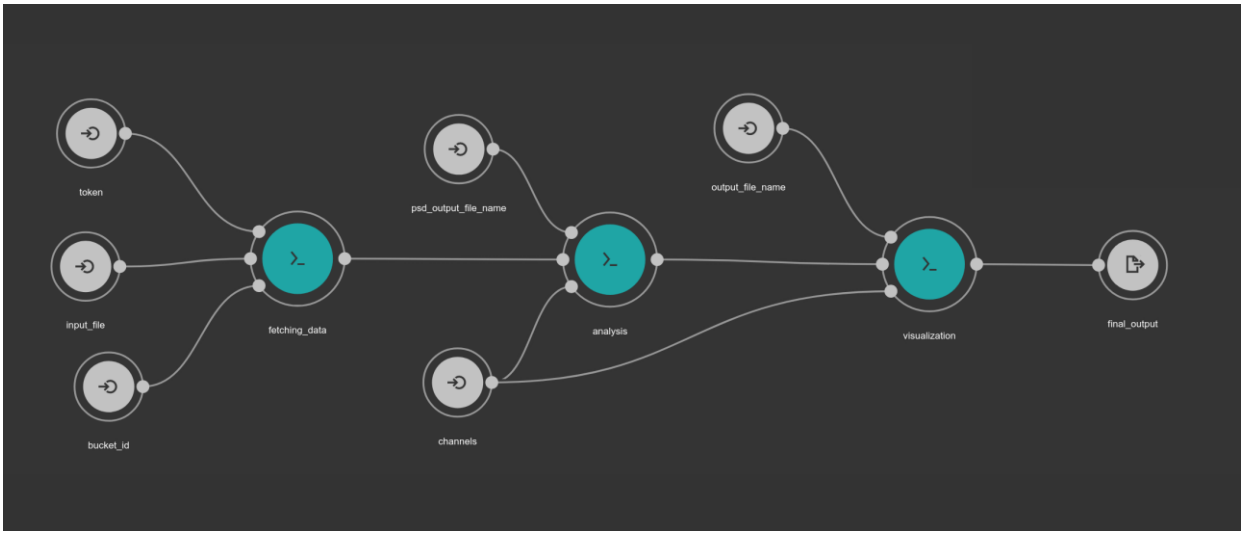


Figure 37: Rabix composer for composing CWL workflows by drag and drop EBRAINS tools already described via CWL

External users will only be able to navigate through the different publicly available CWL workflows and tools. They will be able to download publicly available workflow recipes, but they can only execute them in their local machines since they will not have access to the underlying infrastructure that EBRAINS offers. In that way external users can explore the capabilities that EBRAINS pilot workflow management system provides to the different scientific communities.

4.8 User story

In this last sub section, we combine all three layers of the EBRAINS architectural diagram, FENIX ICEI as the bottom layer, supplementary services and standardised workflows as the middle layer and data and EBRAINS tools from the upper layer in order to present the proposed EBRAINS pilot workflow management system by describing an EBRAINS user story.

Typically, a scientist at EBRAINS will have to find a scientific workflow via EBRAINS Knowledge Graph or compose one by using EBRAINS tools found at EBRAINS Knowledge Graph. In the first case:

- Browse EBRAINS Knowledge Graph by selecting the “Workflows” Category

EBRAINS Knowledge Graph is the central hub for finding and accessing scientific workflows. By navigating into the Workflows categories, a variety of public scientific workflows described via CWL appear. By selecting one, the user will access related metadata information such as a short description of the workflow, what type of input is needed, which is the provided output, how many resources are needed in order to run as well as a graphical representation of the workflow as a graph provided by CWLViewer [16].

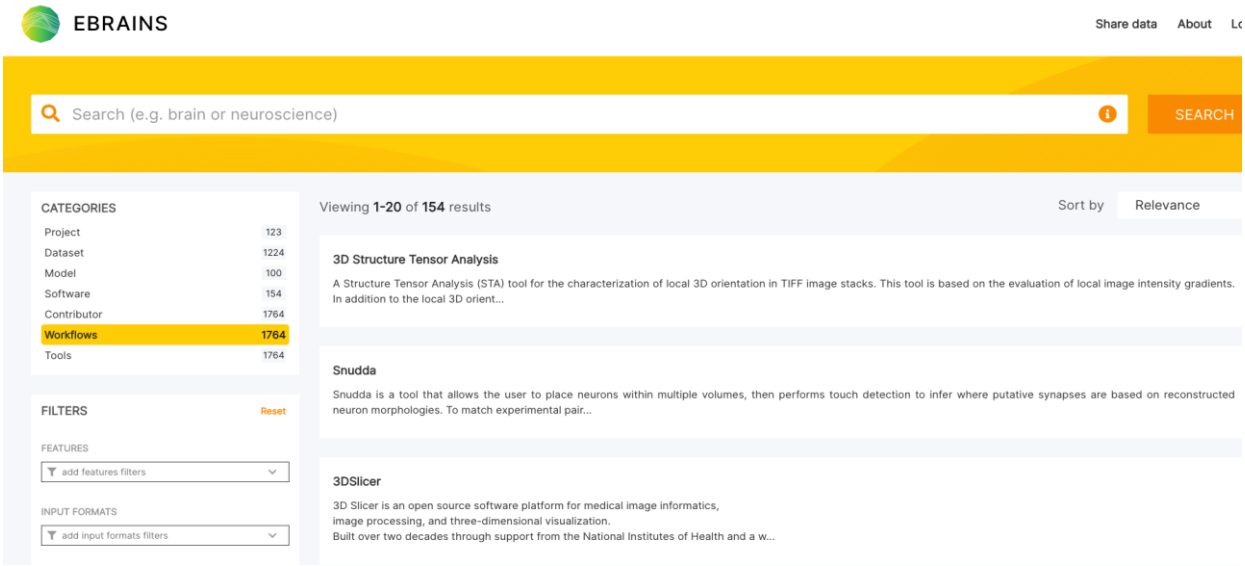


Figure 38: Users can browse "Workflows" categories for finding standardized scientific workflows that are publicly available.

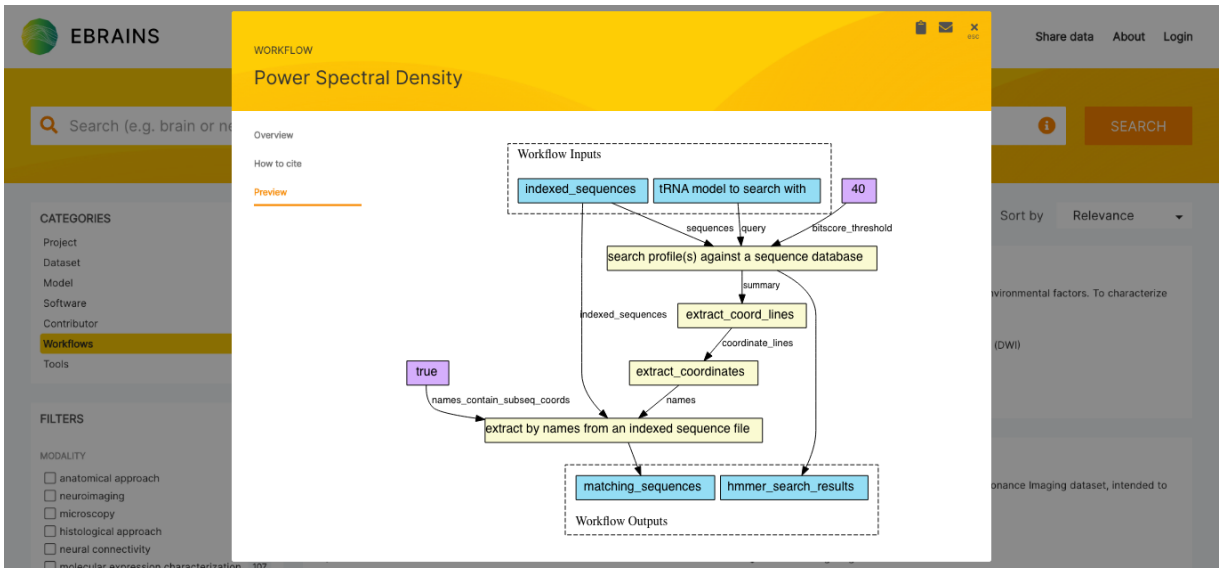


Figure 39 Preview of the standardized scientific workflow as a directed acyclic graphs provided by CWLViewer, where EBRAINS tools are the nodes of the graph, and data flowing between steps are the edges.

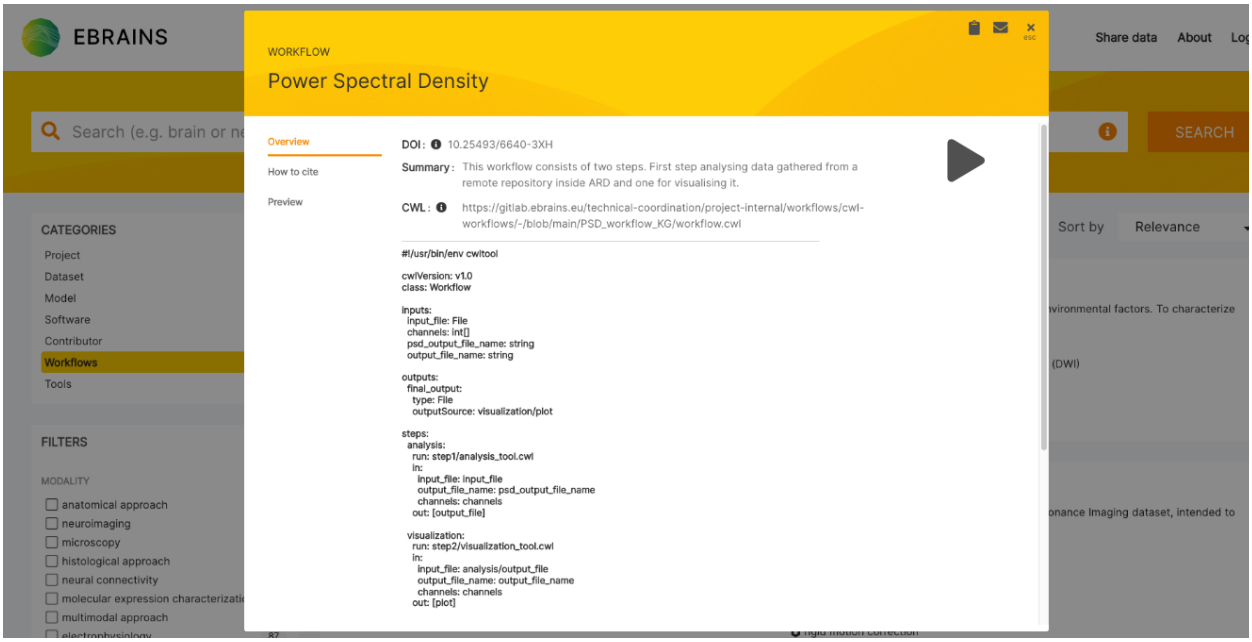


Figure 40: Additional information for standardized scientific workflows, such as the Digital Object Identifier (DOI), as well as the CWL definition. Users will be able to execute workflows in a dedicated endpoint (graphical user interface) by pressing the Play button

In the second case:

- Browse EBRAINS Knowledge Graph by selecting the “Tools” Category

As previously presented, a “Tools” Category under EBRAINS Knowledge Graph exists in which EBRAINS packaged non-interactive tools can be found, accessed and used in order to compose workflows. These tools are also defined via Common Workflow Language, thus, they are well documented, with a proper description of input and output types. The EBRAINS user can:

- Use the Rabix composer [48], which can be an embedded self-hosted environment inside the Knowledge Graph, to graphically compose scientific workflows by using CWL EBRAINS tools available

  Scientists can drag and drop available EBRAINS tools and connect the different inputs and outputs of the workflow steps to creating directed acyclic graphs, in which tools are the nodes and data flowing acts as the edges.

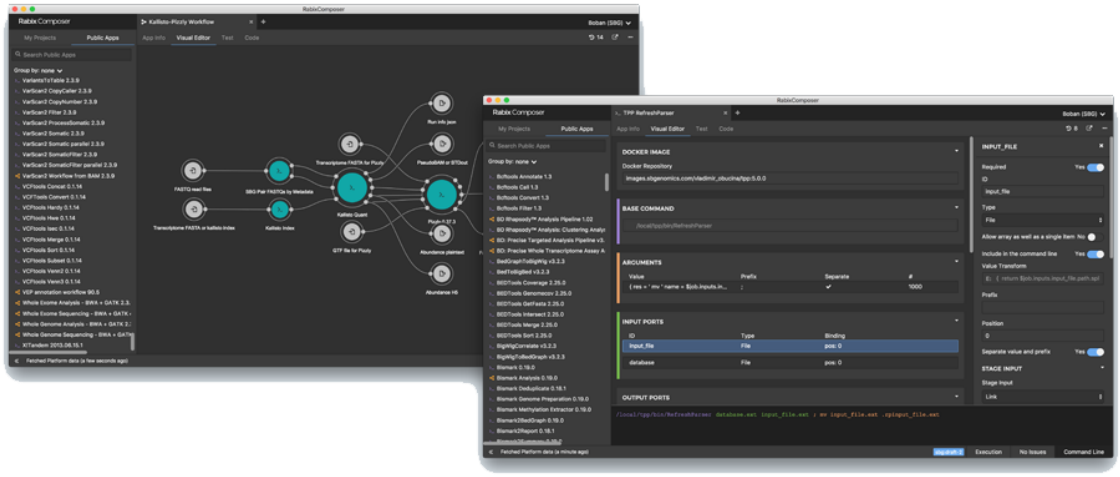


Figure 41: Graphic User Interface, Rabix, for visually describing workflows. Users simply drag and drop already defined tools and connects the inputs as well as outputs of one tool to the other. In that way a graph is created where with tools as nodes and data flowing between them as edges.

Once the user has either found or composed its own CWL scientific workflow from the EBRAINS Knowledge Graph then:

- Selects and submits scientific workflow for execution in the powerful underlying infrastructure.

In order for the EBRAINS user to submit the workflow to the underlying infrastructure, user needs to select the “Play” button. User will be redirected to a dedicated endpoint, the EBRAINS graphical user interface, in which the workflow will be imported. The user can use the default input parameters for the workflow to be executed. Another option is for the user to upload a specific type of file with all the input parameters or use a drop-down menu. The type of the input must be followed and only the input values can be changed. Once input parameters are in place, the workflow will be submitted and executed by the workflow management system, in an opaque from the user way, on top of the HPC system.

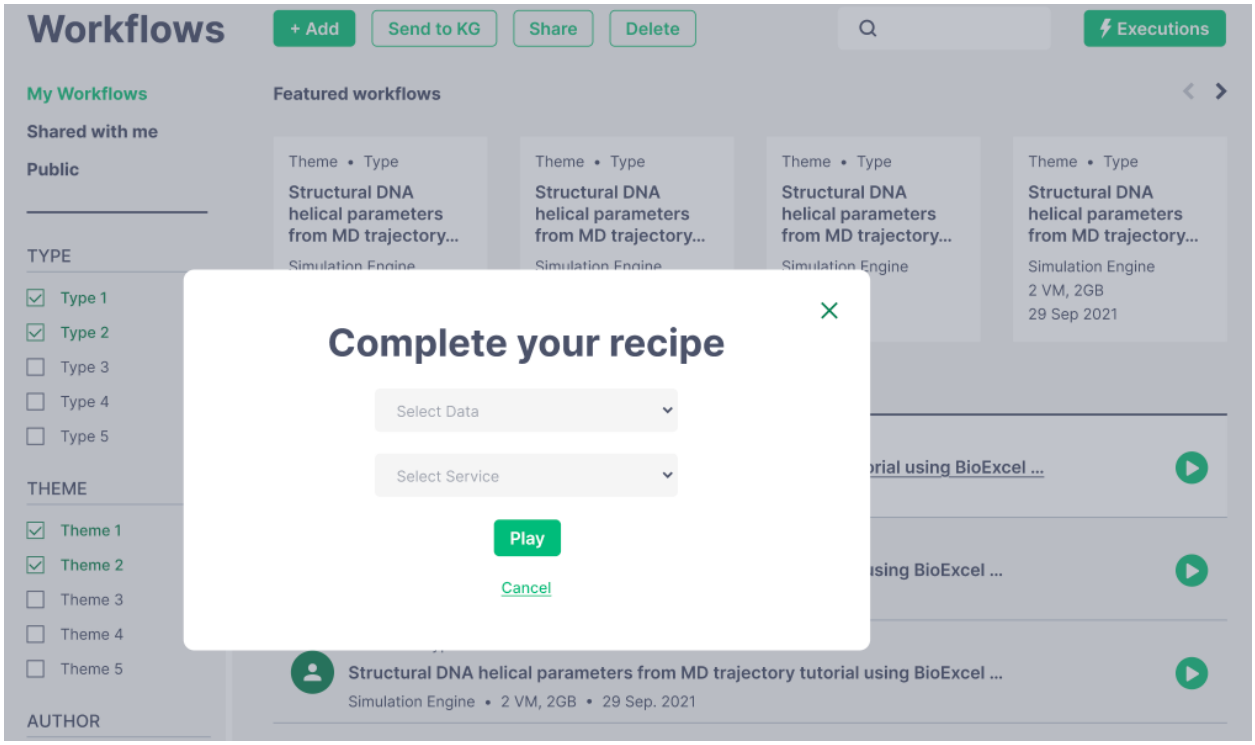


Figure 42:Users provides the input parameters and the input data for the scientific workflow recipe to be submitted.

Once the workflow is submitted, the user can

- Check the status and logs of the executed scientific workflow.

The EBRAINS user will be able to check for logs and status for previous, future as well as current workflow executions. Once the scientific workflow is completed, the user will be notified in order to check the logs of the workflow, the different workflow steps as well as the produced output.

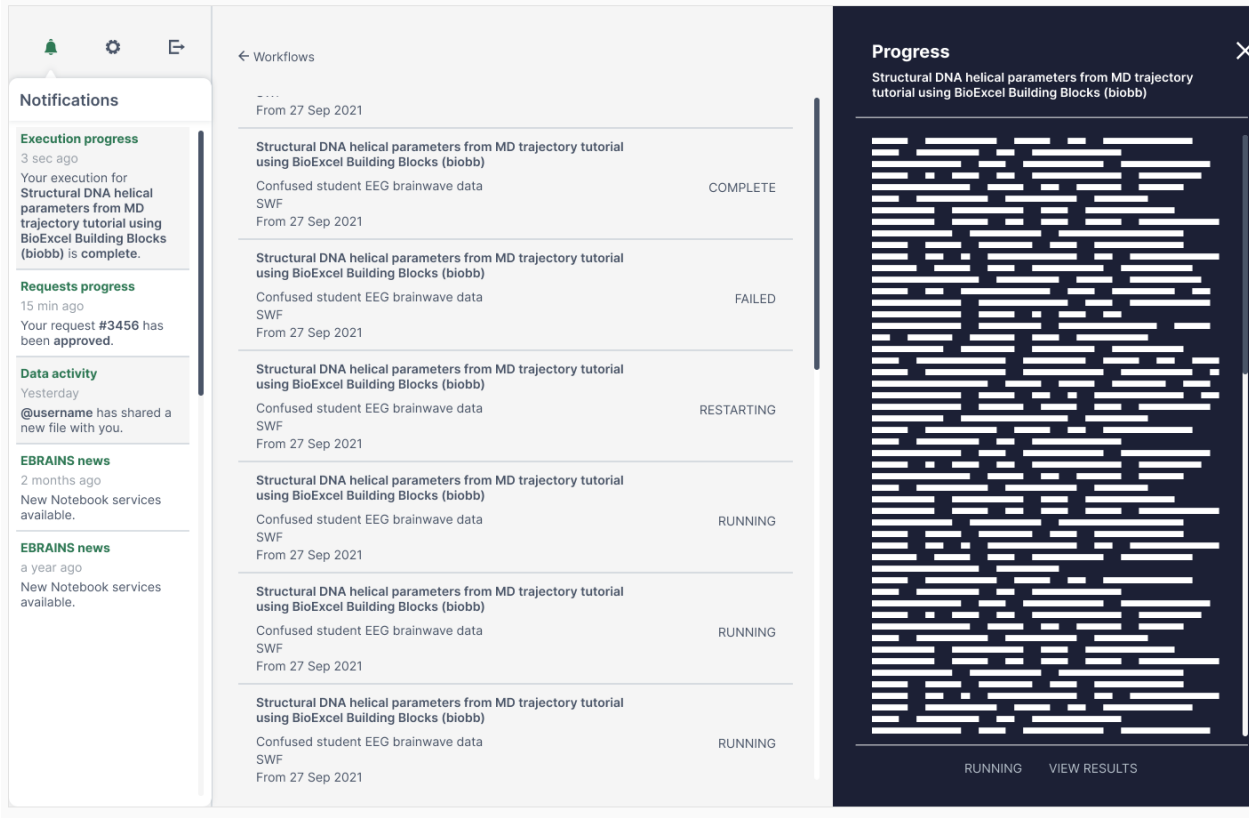


Figure 43:EBRAINS users can get notified and check the status, logs and outputs of the submitted scientific workflows.

The real execution of the submitted scientific workflow takes place in the underlying infrastructure on top of the HPC system.

- Execution is taking care of by the EBRAINS underlying infrastructure

A workflow management system (Toil) runs on top of HPC system in the underlying EBRAINS infrastructure and takes care of the workflow execution in an opaque from the users way. The command that triggered the execution when the user pressed “Play”:

```
toil-cwl-runner –-batchSystem slurm workflow_descr.cwl inputs_obj.yaml
```

where the `workflow_descr.cwl` is the CWL description of the workflow and the `inputs_obj.yaml` the input parameters.

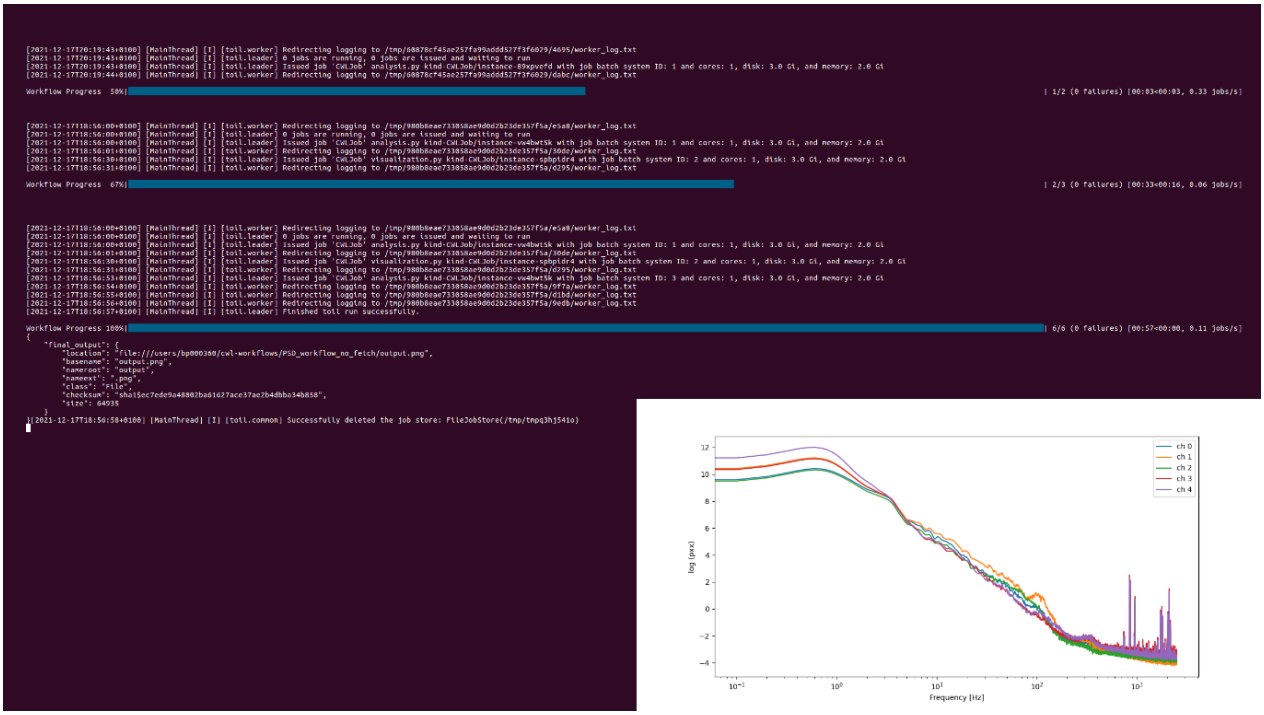


Figure 44:Logs and output provided to the user in an opaque way via the dedicated endpoint.

Once the scientific workflow is completed, the user gets notified and must move the data out of the Active Data Repository to the reliable Archival Data Repository.

- Move the scientific output produced to a dedicated data repository for long term stotage.

As a last step, the user must transfer output to a dedicated data repository for long term storage, since the data repositories associated with HPC systems will be purged soon enough, and the data will be lost. A dedicated CWL workflow can be composed and used by the EBRAINS users in order to move the data out of the HPC systems and associate the outputs with unique persistent identifiers.

Having all scientific assets available in EBRAINS Knowledge Graph from input data, to CWL workflows and tools to produced output data, scientists at EBRAINS make steps towards an Open and FAIR research infrastructure where different representations of the same scientific work will be linked together.

5. RELATED WORK

In this sub-section, we present work from other scientific fields, related to our thesis, that cover aspects of standardization of workflows and non-interactive tools. These aspects include the definition in a structured and common way of workflows and tools used as workflow steps, the execution via different workflow management systems that run on top of software and hardware systems, such as High Performance Computing, hiding the configuration and adjustments that are needed. With respect to storing different tools and workflows, central places for shareability, accessibility, findability and future reference have developed and are currently used by different scientific fields. Last but not least, different ways to visualize already described workflows as well as visualize the description of a created workflow on the fly exist.

There are numerous European Research Infrastructures (RIs) associated with Life Science that adhere to some of the practices of providing wrapped tools, describing workflows in a common standard way, executing them via different workflow engines and storing them in Hubs, providing Findability, Accessibility, Interoperability and Reproducibility. Some of the RIs and projects associated with Life Science are EOSC-Life46 that “brings together the 13 Life Science ESFRI research infrastructures to create an open, digital and collaborative space for biological and medical research”, Elixir47 which is “one of the 13 Life Science ESFRI research infrastructures and a coordinator of EOSC-Life" and Global Alliance for Genomics and Health (GA4GH)48 which is “a policy-framing and technical standards-setting organization, seeking to enable responsible genomic data sharing within human rights framework”. From the related work that each of the above mentioned infrastructures and projects provided, the concept of wrapping tools and the technologies of describing, executing and storing scientific computational workflows have been examined and adapted as a pilot workflow management system for EBRAINS RI.

The first interesting point emphasized in most relevant work is the importance of packaging tools in a standardized way that handles all their dependencies and configurations, so that they can be executed in different underlying infrastructures with minimum overhead. There are means to wrap tools by using either Docker [45] or Singularity [46], that fall under the containerization methods. The fact that wrapped tools can be executed in different underlying infrastructures provides portability in the sense that the execution details of the tool are decoupled from its description. Those wrapped tools can then be easily used in workflow steps in order to compose directed acyclic graphs (DAGs), loops or branches. As a real example in the bioinformatics field, BioContainers [34] is an open-source and community driven framework that provides platform independent executable environments for bioinformatics software. The project provides ways of installing bioinformatics software, maintaining different versions of the same software and combining tools creating analysis pipelines. It is based in Docker and RKT frameworks and has also been integrated with the BioConda [29] project that enables the automatic generation of containers from BioConda and Dockerfiles recipes.

With respect to having a common standard way of describing workflows using wrapped tools as workflow steps, RIs and projects associated with bioinformatics have already

46 https://www.eosc-life.eu/

47 https://elixir-europe.org/

48 https://www.ga4gh.org/

concluded in using common open standards like Common Workflow Language or Workflow Description Language [29]. Common Workflow Language [1] (CWL) is an emerging open standard designed to define analysis tools and workflows with specific input, parameters and resources needed in order to be executed in a portable, scalable way across a variety of software like workflow engines and hardware locally, in the cloud or in HPC environments for accomplishing a specific output. Common Workflow Language is not a piece of software but rather a specification that defines a set of standards that different implementations must conform to. It is compatible with a lot of workflow engines already and supports containerization methods like Docker and Singularity. Common Workflow Language is used by the European Union's BioExcel [55] Centre of Excellence for Biomolecular modelling, and by the IBISBA ESFRI for Industrial Biotechnology. CWL is also participating in GA4GH Task Execution API14 and GA4GH Workflow Execution API15 projects. In general, CWL aims to provide interoperability, extensibility, portability and reproducibility by open source standards for workflow based research. It provides declarative constructs for workflows and command line tool definitions and makes minimal assumptions about base software dependencies, configuration settings, software versions, parameters or execution environment. Defining a command line tool via common standard format that CWL provides, encourages reuse of steps across workflows, researchers as well as communities.

Workflow Description Language (WDL)49 is another way to specify data processing workflows with a human-readable and writable syntax. WDL makes it straightforward to define complex analysis tasks, chain them together in workflows, and parallelize their execution when this is feasible due to lack of dependencies. A variety of bioinformatics workflows and tasks written in WDL can be found in BioWDL50, which is a collection of workflows related to sequencing analyses and is developed at the Leiden University Medical Center by the Sequencing Analysis Support Core (SASC) team. WDL is mostly used in scientific fields like Bioinformatics but is a general-purpose workflow language able to help different fields in defining workflows in a structured way.

As far as the execution of workflows is concerned, and although numerous workflow management systems51 exist and can automate the execution of steps in workflows, can handle failures and can monitor workflows overall, each of these systems use their own syntax or method of describing workflows and infrastructure requirements. This approach inflicts limitations in computational reuse and portability, as well as in publication reuse and research collaboration. That is why there is the mandatory need to have standard ways of describing and executing workflows. In that same approach, many publications have proposed different metrics in order to choose what is the best workflow management system for different aspects. Taking the bioinformatics field as a reference, it is commonly accepted that CWL is more compatible with different workflow engines than WDL. Compatibility of CWL or WDL with different workflow engines is a critical point for choosing which open standard for describing a computational workflow will be used, since there are a lot of different infrastructures and workflow engines exist.

With respect to storing different tools and workflows created by the different Research Infrastructures and projects, aspects like accessibility, findability and shareability played an important role in the way different scientific communities introduced solutions. Biotools18,

49 https://openwdl.org/

50 https://biowdl.github.io/

51 https://github.com/meirwah/awesome-workflow-engines

developed and supported by Elixir, is a comprehensive registry of software and databases that facilitate bioinformatics researchers to find, access, and cite resources needed for their research. In Biotools, tools consist of a single command line from online services, to databases and complex analysis workflows too. WorkflowHub [57] is a new FAIR workflow registry [27] sponsored by the European RI Cluster EOSC-Life and the European Research Infrastructure Elixir. Although currently in Beta, WorkflowHub is a registry for describing, sharing and publishing scientific computational workflows. At the time of the writing, externals can browse up to fifty (50) workflows. With this registry, workflows can be accessed in an interoperable and accessible way due to the usage of open standards and tools like CWL, Research Object-Create, FAIR principles and more. Dockerstore52 provides a place where users can share tools encapsulated in Docker and described with CWL or WDL. myExperiment [56] is an attempt to create a workflow repository, for finding, sharing and publishing workflows with licenses. It credits authors when workflow designs were reused or repurposed, and packages workflows into collections as well as with other digital objects such as associated data files and publications. This work laid the foundations for workflow-based Research Objects that allows bundling of all the artefacts associated with an investigation or piece of research into one whole that can also be cited.

With respect to visualizing workflows, CWL Viewer is a richly featured web visualization suite for workflows written in CWL with the aim of facilitating sharing, understanding and discovery as well as encouraging best practices when writing workflows and their tooling [16]. CWL workflow developers can use the CWL viewer web application that fetches CWL files from remote locations and creates a page with relative information about the workflow itself, the steps, inputs and outputs. CWL viewer also creates visual diagrams of the respective workflows. As an example, the diagram below has been produced by CWL Viewer.

As for the on-the-fly visualization while creating workflows, Rabix Composer [48], created by Seven Bridges, allows users to create and edit CWL workflows in an intuitive way, either through a drag-and-drop Graphic User Interface or a CWL code editor. The workflow recipes produced can then be executed locally or uploaded to any other machine, local or remote, to be executed there using any CWL-compatible workflow engine. While creating a workflow, the user can switch between the visual editor and the CWL code view, and all changes are automatically synced between the two views. Tools that are already defined via CWL can be dragged and dropped into the editor in order to be steps in a workflow. Connections between step inputs and outputs can be created by hovering over the output port on the first step and dragging it to the input port the user wants to connect it to on the second step. Inputs of a step can be whole workflows, the sub workflows, too.

52 https://dockstore.org/

6. CONCUSION

Many research communities had been seeking ways for addressing complex scientific problems by harnessing the power of high-performant and scalable computing infrastructures. These complex scientific problems demand the execution of time-consuming computational tasks, such as analysis and simulation based on big-data and complex models.

In the current thesis, we presented the **EBRAINS Research Infrastructure,** which is dedicated to brain related research across Europe. EBRAINS aims to serve brain research and brain medicine, research and development in Artificial Intelligence (AI), as well as computing and data science. Its **underlying** computing **infrastructure, FENIX ICEI,** provides **a plethora of High Performance Computing services,** ranging from **scalable and interactive computing,** up to virtual machines and containerization services, as well as highly performant data storage services.

In the scope of the thesis, we define:

- **Scientific Computational Workflows** as series of non interactive **EBRAINS tools linked** in order to create graphs, loops or branches for **accomplishing scientific objectives**. Data flowing between the different tools depict the order of execution of the EBRAINS tools.
- **Non interactive EBRAINS tools** as data manipulation, simulation or analysis tools wrapped with dependencies, libraries, binaries and software via Docker. EBRAINS tools are defined via CWL Command Line Tool Description Specification [1]. EBRAINS users can not interact with these tools during the runtime.
- **Standardized workflows** as chains of EBRAINS tools used as workflow steps connected in a specific way to create directed acyclic graphs (DAGs) of operations. CWL Workflow Description Specification [1] is used in order to define standardized scientific workflows as structured recipes along with all the steps, inputs and output data files and the execution details in a YAML format file. Standardized scientific workflows defined via CWL can be executed by CWL- compatible workflow engines, which are responsible for executing, monitoring and retrieving logs and outputs, running on top of a variety of computing platforms, ranging from individual workstations to cluster, grid, cloud, and High Performance Computing systems.

We introduced standardisation methods and technologies (Common Workflow Language) in the already important work that has been done in brain-related research fields, enhancing FAIR, Open and knowledge exchanging aspects of EBRAINS RI. Standardisation entails clear, structured and well documented definitions of flows of data manipulation tasks running in different underlying infrastructures, from local machines to HPC systems, without parametrization. In the current thesis, we focused on HPC systems provided by the EBRAINS underlying infrastructure.

The pilot workflow management system that we propose to be introduced with EBRAINS RI, enables EBRAINS scientists to execute their CWL workflows in a single dedicated endpoint, in which workflows and their input parameters are submitted and monitored. Further, Knowledge exchange in a well-documented, easily understandable cross-scientific manner is an important aspect for making science FAIR. Thus, our goal was to enhance the findability, accessibility, and reproducibility of CWL scientific workflows introducing EBRAINS Knowledge Graph as the central point of reference, for EBRAINS and external users to find and access workflows and EBRAINS tools. In that way, scientific

work with all its components ranging from data, models produced to workflows and tools executed can be coupled together in order to be easily reproducible by EBRAINS scientists, as well as other scientific communities.

Our approach for introducing the pilot workflow management system at EBRAINS RI was based on four pillars:

- First, we chose **Common Workflow Language**, an **open, common and emerging standard** format, as a way for describing both workflows and tools for data manipulation. These structured workflow definitions are also portable and decoupled from runtime environments, i.e., executed across different infrastructures without the need to reconfigure or adapt them in any way.
- Second, we **packaged EBRAINS tools** with their required libraries, dependencies and binaries via Docker containers. We proposed **containerization** for packaging tools with their dependencies, that work well with CWL format for description, as well as with CWL-compatible execution engines. These tools can be used as workflow steps, significantly enhancing their interoperability and portability.
- Third, we **deployed and applied CWL-compatible workflow engines**, enabling scientists to harness the HPC underlying computing resources of the EBRAINS RI. Workflow execution in general is performed by workflow engines that automatically handle all execution complexity, such us automated restarts, monitoring, or fetching output. A plethora of workflow engines exist, which we assessed to select those that best fit the scientific and interoperability requirements of EBRAINS. Our evaluation criteria related to the degree of compatibility with the latest CWL standard, the support of containerization technologies, as well as the need to execute CWL workflows over HPC systems.
- Moreover, we proposed an **EBRAINS Hub** for scientists to easily find, access and store their scientific workflows. In the current thesis, this proposed EBRAINS Hub was designed taking into consideration the already available EBRAINS Knowledge Graph (KG) service. As a future step, integrating such a feature with the EBRAINS service (Knowledge Graph) will take place.
- Finally, we proposed an EBRAINS central place, a **Graphical User Interface** (GUI) in which scientists can **submit, monitor and parametrize workflows**. The proposed GUI has been fully designed but not implemented in the scope of the current thesis. The design process is based on high fidelity mock-ups. EBRAINS users will use this GUI instead of a dedicated CLI for a more user friendly experience. The real execution takes place in the FENIX ICEI underlying infrastructure in an opaque way, in the same way as using the CLI.

This pilot workflow management system was a test case for the real upcoming workflow management system, which will integrate features such as the EBRAINS Hub via Knowledge Graph and the Graphical User Interface (GUI) for a more user friendly experience. After assessing the EBRAINS users' requirements and making a thorough analysis of the state-of-the-art technologies and means that are used by other communities associated with Life Science, we were able to introduce that kind of pilot workflow management system to EBRAINS RI and suggest it to its users in order to provide means and technologies to their valuable scientific work.

The current thesis was partially supported by the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Framework Partnership Agreement No. 650003 (HBP FPA).

ABBREVIATIONS

FAIR	Findable, Accessible, Interoperable, Reproducible
API	Application Program Interface
I/O	Input/Output
EBRAINS	European Brain Research Infrastructure
GPU	Graphical Process Unit
CPU	Central Processing Unit
CWL	Common Workflow Language
RI	Research Infrastructure
ICEI	Interactive Computing E-Infrastructure
HPC	High Performance Computing
DAG	Directed Acyclic Graph
POSIX	Portable Operating System Interfase
AWS	Amazon Web Services
SQL	Structured Query Language
S3	Simple Storage Service
PC	Personal Computer
OpenMP	Open Multiprocessing
MPI	Message Passing Interface
GB	GigaBytes
FENIX	Federated Exascale Network for data Integration and eXchange
VM	Virtual Machine
URL	Uniform Resource Locator
IRI	Internationalized Resource Identifier

REFERENCES

[1] Michael R. Crusoe, Sanne Abeln, Alexandru Iosup, Peter Amstutz, John Chilton, Nebojša Tijanić, Hervé Ménager, Stian Soiland-Reyes, Bogdan Gavrilovic, (for the CWL Community). Methods Included: Standardizing Computational Reuse and Portability with the Common Workflow Language [cs.DC], 4 Aug 2021

[2] Y. Gil et al., "Examining the Challenges of Scientific Workflows," in Computer, vol. 40, no. 12, pp. 24-32, Dec. 2007, doi: 10.1109/MC.2007.421.

[3] Y. S. Tan, R. K. L. Ko and G. Holmes, "Security and Data Accountability in Distributed Systems: A Provenance Survey," 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, 2013, pp. 1571-1578, doi: 10.1109/HPCC.and.EUC.2013.221.

[4] Georgakopoulos, D., Hornick, M. & Sheth, A. An overview of workflow management: From process modeling to workflow automation infrastructure. Distrib Parallel Databases 3, 119–153 (1995).

[5] Malcolm Atkinson, Sandra Gesing, Johan Montagnat, Ian Taylor, Scientific workflows: Past, present and future, Future Generation Computer Systems, Volume 75, 2017, Pages 216-227, ISSN 0167-739X, .

[6] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J. Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, Kent Wenger, Pegasus, a workflow management system for science automation, Future Generation Computer Systems, Volume 46, 2015, Pages 17-35, ISSN 0167-739X, .

[7] Hunt, J. C. R. (1998). "Lewis Fry Richardson and His Contribution to Mathematics, Meteorology and Models of Conflict". Annual Review of Fluid Mechanics. 30 (1): xiii–xxxvi. Bibcode:1998AnRFM..30D..13H. doi:10.1146/annurev.fluid.30.1.0

[8] Grier, David Alan (2005). When Computers Were Human. Princeton University Press. ISBN 978-0-691-09157-0. Archived from the original on August 21, 2006. Retrieved January 24, 2006.

[9] Grier, David Alan (March 1, 2001). "Human Computers: The First Pioneers of the Information Age". Endeavour. 25 (1): 28–32. doi:10.1016/S0160-9327(00)01338-7.

[10] Kean, Sam (2010). The Disappearing Spoon – and other true tales from the Periodic Table. London: Black Swan. p. 108. ISBN 978-0-552-77750-6.

[11] Cayton, Andrew R. L.; Sisson, Richard; Zacher, Chris (2006). The American Midwest: An Interpretive Encyclopedia. ISBN 0253003490.

[12] "CDC 6600 – Historical Interlude: From the Mainframe to the Minicomputer Part 2, IBM and the Seven Dwarfs – They Create Worlds". November 8, 2014.

[13] Hannan, Caryn (2008). Wisconsin Biographical Dictionary. pp. 83–84. ISBN 978-1-878592-63-7. Retrieved 20 February 2018.

[14] Anthony, Sebastian (April 10, 2012). "The History of Supercomputers". ExtremeTech. Retrieved 2015-02-02.

[15] Michael Kotliar, Andrey V Kartashov, Artem Barski, CWL-Airflow: a lightweight pipeline manager supporting Common Workflow Language, GigaScience, Volume 8, Issue 7, July 2019, giz084,

[16] Robinson M, Soiland-Reyes S, Crusoe MR, et al. CWLViewer: The Common Workflow Language Viewer 18thAnnual Bioinformatics Open Source Conference (BOSC2017), F1000Research, 6(ISCB Comm J):1075 (poster). 2017,10.7490/f1000research.1114375.1.

[17] (PDF) Sharing interoperable workflow provenance: A review of best practices and their practical application in CWLProv. Available from: [accessed Dec 07 2021]

[18] Mölder, F., Jablonski, K.P., Letcher, B., Hall, M.B., Tomkins-Tinch, C.H., Sochat, V., Forster, J., Lee, S., Twardziok, S.O., Kanitz, A., Wilm, A., Holtgrewe, M., Rahmann, S., Nahnsen, S., Köster, J., 2021. Sustainable data analysis with Snakemake. F1000Res 10, 33.

[19] Zhou, N., Georgiou, Y., Pospieszny, M. et al. Container orchestration on HPC systems through Kubernetes. J Cloud Comp 10, 16 (2021).

[20] Bartusch, Felix, Maximilian Hanussek and Jens Krüger. “Automatic Generation of Provenance Metadata during Execution of Scientific Workflows.” IWSG (2018).

[21] Ji Liu. Multisite Management of Scientific Workflows in the Cloud. Distributed, Parallel, and Cluster Computing [cs.DC]. Université de Montpellier, 2016. English. fftel-01400625v2f

[22] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing. 2016. Jupyter Notebooks -- a publishing format for reproducible computational workflows. Positioning and Power in Academic Publishing: Players, Agents and Agendas. 87-90.

[23] (1911), The Principles of Scientific Management, New York, NY, USA and London, UK: Harper & Brothers, , . Also available from Project Gutenberg. (1911), The Principles of Scientific Management, New York, NY, USA and London, UK: Harper & Brothers, , . Also available from Project Gutenberg.

[24] Deepak Poola, Mohsen Amini Salehi, Kotagiri Ramamohanarao, Rajkumar Buyya, Chapter 15 - A Taxonomy and Survey of Fault-Tolerant Workflow Management Systems in Cloud and Distributed Computing Environments, Editor(s): Ivan Mistrik, Rami Bahsoon, Nour Ali, Maritta Heisel, Bruce Maxim, Software Architecture for Big Data and the Cloud, Morgan Kaufmann, 2017, Pages 285-320, ISBN 9780128054673, .
[25] Maria A. Rodriguez, Rajkumar Buyya, Chapter 18 - Scientific Workflow Management System for Clouds, Editor(s): Ivan Mistrik, Rami Bahsoon, Nour Ali, Maritta Heisel, Bruce Maxim, Software Architecture for Big Data and the Cloud, Morgan Kaufmann, 2017, Pages 367-387, ISBN 9780128054673, .
[26] Seinstra F.J. et al. (2011) Jungle Computing: Distributed Supercomputing Beyond Clusters, Grids, and Clouds. In: Cafaro M., Aloisio G. (eds) Grids, Clouds and Virtualization. Computer Communications and Networks. Springer, London.
[27] Carole Goble, Sarah Cohen-Boulakia, Stian Soiland-Reyes, Daniel Garijo, Yolanda Gil, Michael R. Crusoe, Kristian Peters, Daniel Schober; FAIR Computational Workflows. Data Intelligence 2020; 2 (1-2): 108–121. doi:
[28] Ludäscher B., Bowers S., McPhillips T. (2009) Scientific Workflows. In: LIU L., ÖZSU M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA.
[29] Fjukstad, B., Bongo, L.A. A Review of Scalable Bioinformatics Pipelines. Data Sci. Eng. 2, 245–251 (2017).
[30] Di Tommaso, P., Chatzou, M., Floden, E. W., Barja, P. P., Palumbo, E., & Notredame, C. (2017). Nextflow enables reproducible computational workflows. Nature Biotechnology, 35(4), 316–319.
[31] N. Liu et al., "On the role of burst buffers in leadership-class storage systems," 2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), 2012, pp. 1-11, doi: 10.1109/MSST.2012.6232369.
[32] "10 The Future of Supercomputing--Conclusions and Recommendations." National Research Council. 2005. Getting Up to Speed: The Future of Supercomputing. Washington, DC: The National Academies Press. doi: 10.17226/11148.
[33] Jackson M, Kavoussanakis K, Wallace EWJ (2021) Using prototyping to choose a bioinformatics workflow management system. PLoS Comput Biol 17(2): e1008622.
[34] Felipe da Veiga Leprevost, Björn A Grüning, Saulo Alves Aflitos, Hannes L Röst, Julian Uszkoreit, Harald Barsnes, Marc Vaudel, Pablo Moreno, Laurent Gatto, Jonas Weber, Mingze Bai, Rafael C Jimenez, Timo Sachsenberg, Julianus Pfeuffer, Roberto Vera Alvarez, Johannes Griss, Alexey I Nesvizhskii, Yasset Perez-Riverol, BioContainers: an open-source and community-driven framework for software standardization, Bioinformatics, Volume 33, Issue 16, 15 August 2017, Pages 2580–2582,
[35] Starlinger J., Cohen-Boulakia S., Leser U. (2012) (Re)Use in Public Scientific Workflow Repositories. In: Ailamaki A., Bowers S. (eds) Scientific and Statistical Database Management. SSDBM 2012. Lecture Notes in Computer Science, vol 7338. Springer, Berlin, Heidelberg.
[36] Herschel, M., Diestelkämper, R. & Ben Lahmar, H. A survey on provenance: What for? What form? What from?. The VLDB Journal 26, 881–906 (2017).
[37] Singh P. (2019) Airflow. In: Learn PySpark. Apress, Berkeley, CA.
[38] Hung et al., 2019, Cell Systems 9, 508–514 November 27, 2019 ª 2019 The Authors. Published by Elsevier Inc.
[39] Ling-Hong Hung, Jiaming Hu, Trevor Meiss, Alyssa Ingersoll, Wes Lloyd, Daniel Kristiyanto, Yuguang Xiong, Eric Sobie, Ka Yee Yeung, Building Containerized Workflows Using the BioDepot-Workflow-Builder, Cell Systems, Volume 9, Issue 5, 2019, Pages 508-514.e3, ISSN 2405-4712,
[40] Valentina Huber. 2001. UNICORE: A Grid Computing Environment for Distributed and Parallel Computing. In Proceedings of the 6th International Conference on Parallel Computing Technologies (PaCT '01). Springer-Verlag, Berlin, Heidelberg, 258–265.
[41] Sarah Cohen-Boulakia, Khalid Belhajjame, Olivier Collin, Jérôme Chopard, Christine Froidevaux, et al.. Scientific workflows for computational reproducibility in the life sciences: Status, challenges and opportunities. Future Generation Computer Systems, Elsevier, 2017, 75, pp.284-298. ff10.1016/j.future.2017.01.012ff. ffhal-01516082f
[42] Ludäscher B., Weske M., McPhillips T., Bowers S. (2009) Scientific Workflows: Business as Usual?. In: Dayal U., Eder J., Koehler J., Reijers H.A. (eds) Business Process Management. BPM 2009. Lecture Notes in Computer Science, vol 5701. Springer, Berlin, Heidelberg.
[43] Darmody, P.. “Henry L. Gantt and Frederick Taylor: The Pioneers of Scientific Management.” (2007).
[44] R. Ferreira da Silva, H. Casanova, K. Chard, T. Coleman, D. Laney, D. Ahn, S. Jha, D. Howell, S. Soiland-Reys, I. Altintas, D. Thain, R. Filgueira, Y. Babuji, R. M. Badia, B. Balis, S. Caino-Lores, S. Callaghan, F. Coppens, M. R. Crusoe, K. De, F. Di Natale, T. M. A. Do, B. Enders, T. Fahringer, A. Fouilloux, G. Fursin, A. Gaignard, A. Ganose, D. Garijo, S. Gesing, C. Goble, A. Hasan, S. Huber, D. S. Katz, U. Leser, D. Lowe, B. Ludaescher, K. Maheshwari, M. Malawski, R. Mayani, K. Mehta, A. Merzky, T. Munson, J. Ozik, L. Pottier, S. Ristov, M. Roozmeh, R. Souza, F. Suter, B. Tovar, M. Turilli, K. Vahi,

A. Vidal-Torreira, W. Whitcup, M. Wilde, A. Williams, M. Wolf, J. Wozniak, “Workflows Community Summit: Advancing the State-of-the-art of Scientific Workflows Management Systems Research and Development”, Technical Report, June 2021, DOI: 10.5281/zenodo.4915801.

[45] Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. Linux Journal, 2014(239), 2.

[46] Kurtzer GM, Sochat V, Bauer MW (2017) Singularity: Scientific containers for mobility of compute. PLoS ONE 12(5): e0177459. https://doi.org/10.1371/journal.pone.0177459

[47] Benedicic L., Cruz F.A., Madonna A., Mariotti K. (2019) Sarus: Highly Scalable Docker Containers for HPC Systems. In: Weiland M., Juckeland G., Alam S., Jagode H. (eds) High Performance Computing. ISC High Performance 2019. Lecture Notes in Computer Science, vol 11887. Springer, Cham. https://doi.org/10.1007/978-3-030-34356-9_5

[48] Kaushik G, Ivkovic S, Simonovic J, Tijanic N, Davis-Dusenbery B, Kural D. RABIX: AN OPEN-SOURCE WORKFLOW EXECUTOR SUPPORTING RECOMPUTABILITY AND INTEROPERABILITY OF WORKFLOW DESCRIPTIONS. Pac Symp Biocomput. 2017;22:154-165. doi:10.1142/9789813207813_0016

[49] Vivian, J., Rao, A. A., Nothaft, F. A., Ketchum, C., Armstrong, J., Novak, A., … Paten, B. (2017). Toil enables reproducible, open source, big biomedical data analyses. Nature Biotechnology, 35(4), 314–316.

[50] Yoo A.B., Jette M.A., Grondona M. (2003) SLURM: Simple Linux Utility for Resource Management. In: Feitelson D., Rudolph L., Schwiegelshohn U. (eds) Job Scheduling Strategies for Parallel Processing. JSSPP 2003. Lecture Notes in Computer Science, vol 2862. Springer, Berlin, Heidelberg.

[51] Garrick Staples. 2006. TORQUE resource manager. In Proceedings of the 2006 ACM/IEEE conference on Supercomputing (SC '06). Association for Computing Machinery, New York, NY, USA, 8–es. DOI:https://doi.org/10.1145/1188455.1188464

[52] Calzolari A, Valerio A, Capone F, Napolitano M, Villa M, Pricci F, Bravo E, Belardelli F. The European Research Infrastructures of the ESFRI Roadmap in Biological and Medical Sciences: status and perspectives. Ann Ist Super Sanita. 2014;50(2):178-85. doi: 10.4415/ANN_14_02_12. PMID: 24968918.

[53] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. 2006. Ceph: a scalable, high-performance distributed file system. In Proceedings of the 7th symposium on Operating systems design and implementation (OSDI '06). USENIX Association, USA, 307–320.

[54] Wilkinson, M. D. et al. The FAIR Guiding Principles for scientific data management and stewardship. Sci. Data 3:160018 doi: 10.1038/sdata.2016.18 (2016).

[55] Andrio, P., Hospital, A., Conejero, J. et al. BioExcel Building Blocks, a software library for interoperable biomolecular simulation workflows. Sci Data 6, 169 (2019).

[56] Goble CA, Bhagat J, Aleksejevs S, et al. myExperiment: a repository and social network for the sharing of bioinformatics workflows. Nucleic Acids Res. 2010;38(Web Server issue):W677-W682. doi:10.1093/nar/gkq429

[57] R. F. d. Silva, L. Pottier, T. Coleman, E. Deelman and H. Casanova, "WorkflowHub: Community Framework for Enabling Scientific Workflow Research and Development," 2020 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS), 2020, pp. 49-56, doi: 10.1109/WORKS51914.2020.00012.