EBRAINS Infrastructure (D5.7 SGA3)

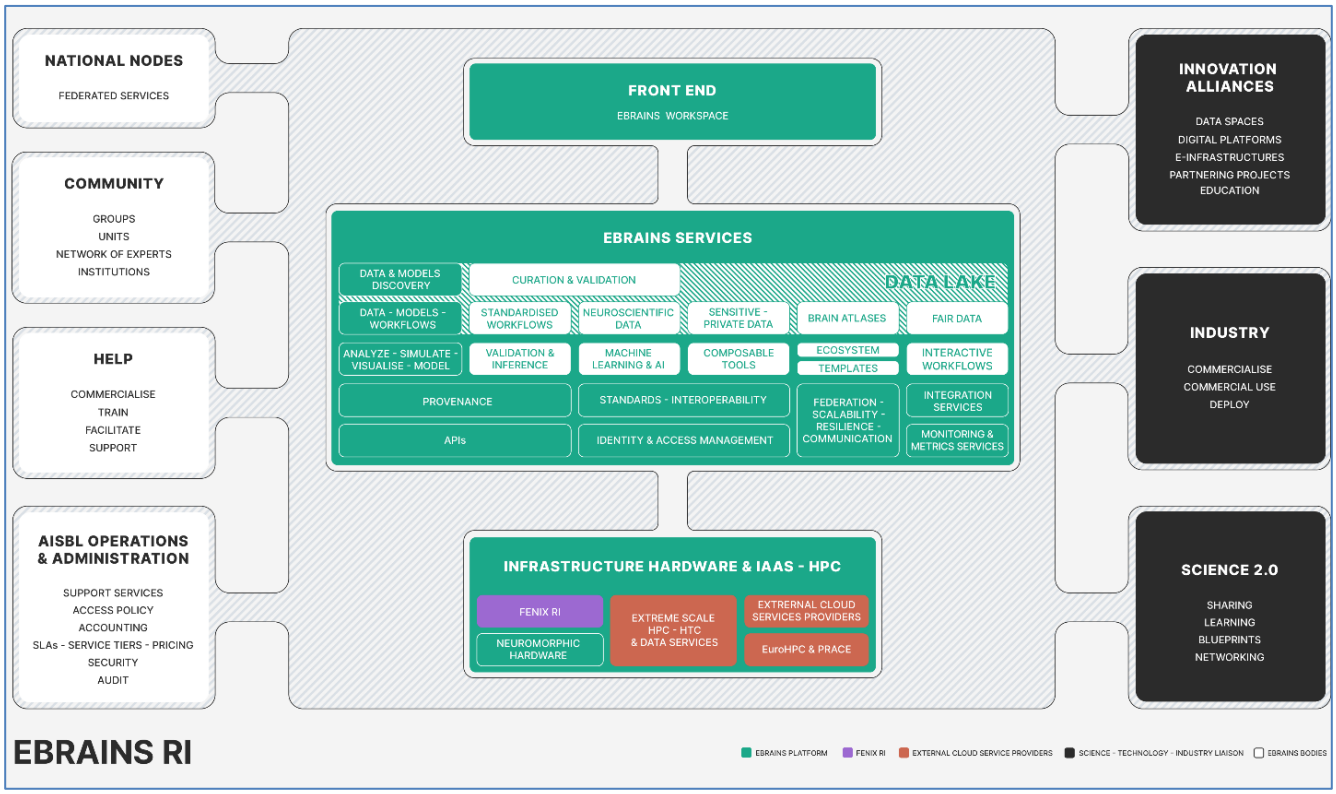

Figure 1: EBRAINS RI Overview

Project Number:	945539	Project Title:	HBP SGA3

Document Title:	EBRAINS Infrastructure
Document Filename:	D5.7 (D54) SGA3 M42 SUBMITTED 231011.docx
Deliverable Number:	SGA3 D5.7 (D)
Deliverable Type:	Demonstrator
Dissemination Level:	PU = Public
Planned Delivery Date:	SGA3 M42 / 30 Sep 2023
Actual Delivery Date:	SGA3 M43 / 11 Oct 2023
Author(s):	Amaryllis RAOUZAIOU, ATHENA (P133), Spiros ATHANASIOU, ATHENA (P133), Thanassis KARMAS, ATHENA (P133), Sofia KARVOUNARI, ATHENA (P133), Eleni MATHIOULAKI, ATHENA (P133), Orfeas AIDONOPOULOS, ATHENA (P133), Vassilios GEORGOPOULOS, ATHENA (P133), Rowan THORPE, ATHENA (P133), Nikolaos GEORGOMANOLIS, ATHENA (P133), Konstantinos TRIANTOS, ATHENA (P133), Michail ALEXAKIS, ATHENA (P133), Nikolaos PAPPAS, ATHENA (P133)
Compiled by:	Amaryllis RAOUZAIOU, ATHENA (P133), Spiros ATHANASIOU, ATHENA (P133), Thanassis KARMAS, ATHENA (P133), Sofia KARVOUNARI, ATHENA (P133), Eleni MATHIOULAKI, ATHENA (P133), Orfeas AIDONOPOULOS, ATHENA (P133), Vassilios GEORGOPOULOS, ATHENA (P133), Nikolaos GEORGOMANOLIS, ATHENA (P133)
Contributor(s):	Wouter KLIJN, JUELICH (P20), Andrew DAVISON, CNRS (P10), Dennis TERHORST, JUELICH (P20), Eric MULLER, UHEI (P47), Daniel KELLER, EPFL (P134), Daviti GOGSHELIDZE, JUELICH (P20), James Gonzalo KING, EPFL (P135), Sandra DIAZ, JUELICH (P20), Thorsten HATER, JUELICH (P20), Marmaduke WOODMAN, AMU (P78) Arnau MANASANCH, IDIBAPS (P93), Cosimo LUPO, INFN (P92), Irene BERNAVA, INFN (P92), Pier Stanislao PAOLUCCI, INFN (P92), Giulia DE BONIS, INFN (P92), Robin GUTZEN, JUELICH (P20), Michael DENKER, JUELICH (P20), Mario SENDEN, UM (P117), Tonio WEIDLER, UM (P117)
WP QC Review:	Evita MAILLI, ATHENA (P133)
WP Leader:	Yannis IOANNIDIS, ATHENA (P133)
T7.4 QC Review:	N/A
Description in GA:	(Prototype); includes the working EBRAINS infrastructure. An accompanying report asserts the completion of all implementation, quality assurance and delivery of the infrastructure. Interim releases of the infrastructure are presented at given milestones.
Abstract:	Deliverable D5.7 is the final deliverable of Technical Coordination and summarizes the delivered software and quality assurance performance of the EBRAINS RI, building upon deliverables D5.4 “Interim EBRAINS Infrastructure Implementation Report” and D5.3 “EBRAINS Technical Coordination Guidelines”. Towards this, we: (a) refine the conceptual, logical, and physical architecture to deliver the final version, (b) update and report on the integration implementation and planning of the RI, (c) evaluate the integration progress and associated KPIs of EBRAINS components towards a coherent and sustainable RI, (d) introduce new software offerings, methods and tools that facilitate the integration and operationalization of the RI, and (e) present some of the procedures that will remain in effect after the end of SGA3 to ensure that all available information remains accessible.
Keywords:	Architecture, integration, guidelines, workflows, software quality, software delivery, infrastructure, services, conceptual architecture, physical deployment, components, API, KPIs
Target Users/Readers:	computational neuroscience community, computer scientists, consortium members, HPC community, neuroimaging community, neuroinformaticians, neuroscientific community, neuroscientists, platform users, researchers, scientific community
NOTE:	If you cannot access a link to an EBRAINS collab in this document (URL starting https://wiki.ebrains.eu), please contact arc-ebrains@athenarc.gr .

Table of Contents

Table of Tables

Table of Figures

Introduction

In SGA3, the HBP built EBRAINS, a distributed digital research infrastructure that will endure as a functioning legacy after 2023. EBRAINS is at the interface of neuroscience, computing, and technology. The EBRAINS infrastructure is shaped by the principle of co-design, which means that (a) the needs of the scientists served as the basis for the development of tools and services, and (b) the insight and expertise of scientists flew into the conception and realisation of the EBRAINS infrastructure.

At the end of SGA3, we deliver the production version of the EBRAINS Research Infrastructure, a "one-stop-shop" offering scientists and developers the most advanced tools and services for brain research, including FAIR data services, next-generation brain atlasing, simulation platforms and AI-based analysis of big data. The EBRAINS RI includes innovative brain-inspired technologies and computing, enabling also digital applications for industrial and medical use, powered by the Federated Exascale Network for data Integration and eXchange (Fenix1) Infrastructure, itself a blueprint for other research communities. EBRAINS will serve brain research and brain medicine, research and development in AI, computing, and data science, as well as other technologies benefiting from insights into brain organisation.

The implementation planning and technical coordination for the EBRAINS infrastructure are led by WP5. WP5 is one of the infrastructure Work Packages (along with WP4 and WP6) and includes the core tasks of the technical coordination, integration, testing and delivery of the EBRAINS infrastructure. Deliverable D5.7 is the third and final deliverable of T5.11 and asserts the completion of all implementation, quality assurance and delivery of the infrastructure. Wherever there is a direct relation to the content of D5.4 or D5.3, there is a clear reference in the text, while any changes in relation to D5.4 are also detailed into the corresponding self-contained sections to assist the reader. Reviewers' comments from the M21 review have been considered, further analysed, and served as a basis for our efforts during this period. Specifically, in this document we: (a) refine the conceptual, logical, and physical architecture to reflect current progress and delivered software, presenting the final version of the different diagrams, (b) update and report on the integration implementation of the RI, (c) evaluate the integration progress and associated KPIs of EBRAINS components towards a coherent and sustainable RI, (d) introduce software offerings, methods and tools that facilitate the integration and operationalization of the RI and (e) present suggestions and plans for the post-SGA3 period, with some already in effect.

The delivery of the EBRAINS RI marks the next step in the decade-long scientific and innovation journey of the Human Brain Project, assembling and leveraging contributions from thousands of scientists and engineers. It is our ambition for the EBRAINS RI, its stakeholders and user community to continue this successful path in the future, further improving, extending, and using the integrated output of Human Brain Project, building on its legacy. Towards this, we have taken all necessary steps to ensure the technical sustainability of the EBRAINS RI, mitigate potential risks, and lay a roadmap for its further evolution.

The status of the EBRAINS RI at the end of SGA3 is summarised as follows:

- All (73) HBP software components have been successfully integrated into the EBRAINS RI. We have conducted a quantitative evaluation to objectively assess their progress and integration status.
- The conceptual, logical, and physical architecture of the EBRAINS RI have been revised to reflect current progress, our ambitions beyond SGA3, and the sustainability goals of EBRAINS.
    - All SGA3 TC instruments, decision-making processes, and co-design activities (TC Weeklies, TC-TF, Working Groups, ad hoc meetings) are being successfully applied and extended to further facilitate the scientific work of SGA3 Showcases.

[1] https://fenix-ri.eu/

- During the last period of SGA3, we have developed and introduced several new concepts, tools, and instruments aimed at facilitating integration, ensuring FAIR-ness, standardization, cost-effectiveness, observability, monitoring, and overall sustainability of the EBRAINS RI.
- We have updated the Integration Guidelines and provided post-SGA3 guidelines, all of which are presented in this document.
- The migration of the EBRAINS RI to the base infrastructure available post-SGA3 has been designed and initiated to ensure the continued operation of the RI.
- We have established procedures that will remain in effect after the end of SGA3 to ensure that all available information remains accessible.

Through these processes, at the conclusion of HBP SGA3, TC successfully led the delivery of the EBRAINS RI, a sustainable integrated platform prepared to continue serving as a vital research tool for neuroscientists, but also poised for further enhancement with new services and tools within the framework of future initiatives.

1.1 Document Structure

This document includes, apart from introductory and concluding parts, **six main sections** and several **annexes**, assembling and presenting all relevant information for the EBRAINS Infrastructure implementation and delivery.

Section 2 presents the updates of the **EBRAINS Architecture** at the end of SGA3. This section includes the **High-Level Overview** of EBRAINS meant to assist the understanding of EBRAINS by external stakeholders, the **Conceptual Architecture** of EBRAINS meant to assist the understanding of EBRAINS mostly by SGA3 stakeholders, the **Logical Architecture** of the EBRAINS RI at two levels of abstraction and detail, and the **Physical Deployment** of EBRAINS over the available computing, storage, and networking resources. Furthermore, we provide an overview of supported deployment options both for *current* and *future* EBRAINS component developers, aiming to support them in the deployment of their services in EBRAINS. We place special emphasis in this line of work, as it aims to lower the entry barrier for new EBRAINS offerings on a technical level, thus assisting in the realisation of the long-term vision for EBRAINS, as a constantly evolving marketplace of services and offerings for neuroscience and the brain.

Section 3 summarizes the developed **EBRAINS Platform Services**. Specifically, the first subsection provides information on the **EBRAINS Software Delivery and Deployment**, focusing on the definition of a unified, consistent *EBRAINS software distribution* including all EBRAINS simulation engines and software tools, and the design and implementation of an optimized *delivery strategy* for deployment on both interactive (EBRAINS Lab) and HPC systems. The next subsection is dedicated to presenting the use of **Standardised Workflows.** It brings EBRAINS in line with other ESFRIs, enabling users to create and share reusable, interoperable, complex scientific workflows using EBRAINS data, tools, models, and software that can fully leverage current and future computing infrastructures in a cost-effective and scalable manner. The subsection also presents how the different EBRAINS use cases embraced the Standardised workflows approach. The Service for automated headless browser testing of Jupyter Notebooks is presented in the next subsection. The Automation and Back-end Tools and Services subsection presents our work on establishing **Monitoring Services**, while the last subsection provides information about the **Dashboard design**, focusing on the **Workflows Section Dashboard**.

Section 4 focuses on the **implementation of the integration process, presenting** the **latest developments** that facilitate the integration of the different services in EBRAINS and the main axes of **Software Quality guidelines**. The respective subsections have been developed under the feedback, contribution, and review of the **EBRAINS Software Quality (ESQ-WG)** and **EBRAINS Software Delivery (ESD-WG) Working Groups**. Further, we provide information about the functionalities that are relevant to **Sensitive data and integration**.

Section 5 presents the **KPIs** applied for **quality control** along with corresponding values during the whole SGA3 (four different Phases). It also presents the status of the development/implementation

phases that have been established to keep track of the evolution and progress during SGA3, where we are **now at the end of the project** and the **risks** we have faced.

In the concluding section, **Section 6**, we present some of the procedures that will remain in effect **after the end of SGA3** to ensure that all available information remains **accessible.**

Finally, several annexes are provided in this report, offering additional insights and details to the reader.

1.2 How to read this deliverable - Relation to other deliverables and documents

D5.7 presents our work for the implementation of EBRAINS Infrastructure. The Deliverable is addressed to all the SGA3 Partners, presenting our current vision and directions regarding the development, integration, and delivery of EBRAINS Infrastructure.

The reader can find in this document all information related to the EBRAINS technical approach. This information is already communicated to SGA3 partners through the different meetings, documents or presentations and was presented in both D5.3 and D5.4. Annexes at the end of D5.7 provide additional information about the various documents the reader may need for a better understanding of efforts, status, and progress. Finally, D5.7 references content from EBRAINS Collaboratory wiki pages and other Project Deliverables as appropriate.

D5.7 is closely related to D5.14 [8] and D5.15 [9], the two deliverables of the EBRAINS Chief Infrastructure Security Officer (CISO) that identify and assess infrastructure-level security risks and incident management (D5.14) and deliver a comprehensive proposal for the introduction of a formal security framework (D5.15).

Like D5.4, there is a direct link of D5.7 with D7.1 (White Paper on Quality Control - Version 2) which presents the different processes of the Project, including processes, documents, and Collabs of the Technical Coordination, offering them more visibility. Moreover, there is an organic link between the two Deliverables: D7.1 refers to QC of the project (processes, output), while D5.7 refers to updated QC/KPIs on technical level (development, testing, integration, operations). D5.7 is also connected to D7.3 (Governance Handbook - Version 2), since the latter includes a description of how TC is structured and presents the different bodies and procedures. The Showcase Deliverables (D1.1, D2.1, D3.1, D3.3 and their updated versions D1.2, D1.5, D2.3, D2.4, D3.5, D3.6, D3.14, D3.15) serve as an information source on possible RI requirements related to the Showcases, while D4.1, D4.2, D4.4, D4.7, D4.8, D4.9, D4.12, D4.13, D4.15, D4.16, D5.1, D5.2, D5.5, D5.6 D5.8, D5.9 and D5.11 are valuable sources of information for the different Service Categories. We have already set up the TC prioritisation process focused on the Service Categories (SCs) [1] in response to the need to prioritise EBRAINS-wide technical aspects and issues of critical nature for the delivery of the SCs' outputs.

D5.7, along with D5.14 [8] and D5.15 [9] complete WPO5.3 (EBRAINS software components integration, testing and delivery). Two more deliverables are connected to this WPO: the already delivered D5.3 - EBRAINS Technical Coordination Guidelines – M10 and D5.4 - Interim EBRAINS Infrastructure Implementation Report – M21.

1.2.1 Changes from D5.4

In comparison with D5.4 [2], significant enhancements have been made.

The architecture diagrams have been updated, with the current deliverable documenting the final version of the Architecture (Section 2).

As far as the delivery and deployment of EBRAINS tools is concerned (Section 3.1), having already defined a first delivery process for the Collaboratory Lab environment in D5.4, the focus shifted on the establishment of the EBRAINS software stack as a standardized, tested, version-controlled software ecosystem, deployable on multiple systems and architectures. Specifically, automated build tests were introduced to ensure sufficient testing and integration checks for new software

(Section 3.1.2.2), a release schedule including weekly experimental releases and quarterly, well-tested official releases was established (Section 3.1.2.3), and the build and deployment processes for the Lab were optimized and finalized (Section 3.1.2.1). Finally, the EBRAINS software stack was successfully deployed to the Fenix ICEI HPC systems, in collaboration with WP6 (as reported in D6.4) and representatives of the various Fenix ICEI sites and made available to EBRAINS users (Section 3.1.2.4).

Regarding the Workflows section, D5.4 provided definitions of terms related to workflows in a general as well as in the specific EBRAINS RI context, including preliminary definitions and actions for Standardised workflows approach. It also highlighted the benefits for both developers, scientists of EBRAINS and EBRAINS RI itself, in supporting a structured, common, and standard approach in defining workflows. During this time, specific and more concrete action lines have been established, for scientists and developers to integrate their work, focused on the EBRAINS Showcases and EBRAINS services and tools, into the Standardised approach (Sections 3.2.1 and 0).

Moreover, in D5.4 the connection with EBRAINS' Provenance and Knowledge Graph (KG) components was briefly elaborated, describing how these two elements could be integrated into the Standardised workflows approach. During this period, notable progress was made in leveraging EBRAINS KG to store, discover, and access Standardised workflows [2],[3] described via the Common Workflow Language (CWL). The provenance component has since evolved into four sub-components, each providing critical features. These include the enhancement of openMinds computational module metadata schema, the realization of EBRAINS Provenance API to a production-ready state, the development of a provenance visualization app, and the formulation of plans for a Python library to effectively capture prospective and retrospective provenance information (Section 3.2.2).

Furthermore, in D5.4 it was determined that the implementation of the Workflows Dashboard was critical. This decision led to the realization of significant features inside the Workflows Dashboard including the ability to create, submit, execute, and monitor standardised workflows in the EBRAINS underlying infrastructure, by making use of the CWL-compatible workflow engines already configured and deployed by the Technical Coordination team. Moreover, Dashboard features were soon enriched with users' capabilities to check workflow details, review previous executions of a workflow, mark workflows as Featured for public availability, and easily retrieve workflows to their personal workspaces for workflow editing and customization. Beyond standardised workflows features, additional features inside Dashboard, including user data and sharing via Nextcloud, integration of Buckets via the Data proxy as well as user profiling, were introduced. These enhancements will transition the implementation of a Workflow specific Dashboard to an upcoming unified Dashboard ensuring a comprehensive and user-centric experience (Section 3.5).

Lastly, compared to the previous deliverable D5.4, which outlined the foundational framework of the monitoring service, the current one presents the architecture in greater detail, explains how multi-level monitoring is achieved, and describes the dashboards per level of monitoring, all of which are detailed in Sections 3.4.1 and 8.2 (Annex).

In Section 4 there is the new -and final- version of Software Quality Guidelines and a short presentation of the possibilities and regulations within the Fenix infrastructure for storing and processing personal data.

Finally, the current deliverable includes the assessment of the EBRAINS Infrastructure integration process with the use of updated Progress indicators. The results are presented in detailed diagrams and tables (Section 5)

1.2.2 Terminology

Please note that in D5.7, as in D5.3 and D5.4 before it, to avoid confusion, the term '**EBRAINS RI**' refers to **the final output of SGA3** (i.e., the EBRAINS Research Infrastructure), delivered at the end of the Project.

[2] https://search.kg.ebrains.eu/instances/bd71c7c0-c2bb-454a-8d18-12ef96f45cdd
[3] https://search.kg.ebrains.eu/instances/86efc59a-b4d1-443d-a470-dd27ddc4465b

Table 1 lists the most important acronyms that the reader can find across the different sections.

Table 1: Acronyms

Acronym	Explanation
AAI	Authentication & Authorisation Infrastructure
ACL	Access-Control List
API	Application Programming Interface
CD	'Continuous Delivery' or 'Continuous Deployment' (based on context)
CI	Continuous Integration
CLI	Command Line Interface
CWL	Common Workflow Language
FAIR	Findability, Accessibility, Interoperability, Reusability
HPC	High-Performance Computing
IaaS	Infrastructure as a Service
IAM	Identity & Access Management
IdP	Identity Provider
KG	Knowledge Graph
KPI	Key Performance Indicator
MOOC	Massive Open Online Course
NMC	NeuroMorphic Computing
PaaS	Platform as a Service
REST	REpresentational State Transfer
SaaS	Software as a Service
SLURM	Simple Linux Utility for Resource Management
TRL	Technology Readiness Level
VM	Virtual Machine
Fenix	Federated Exascale Network for data Integration and eXchange
FURMS	Fenix User & Resource Management Service
HBP	Human Brain Project
HLST	High-Level Support Team
ICEI	Interactive Computing E-Infrastructure
KG	Knowledge Graph
NEST	NEural Simulation Technology
NRP	Neuro-Robotics Platform
NSG	Neuro-Science Gateway
SLU	Scientific Liaison Unit
TC	Technical Coordination
TVB	The Virtual Brain (project)
SLA	Service-Level Agreement
HDC	HealthDataCloud

2. Architecture

In this section, we present the latest and final (in the context of SGA3) version of the **EBRAINS RI Architecture**. The first version of the EBRAINS Architecture was detailed in "EBRAINS Technical Coordination Guidelines" (D5.3, Section 3,[1]). The evolution of the EBRAINS Architecture during Phase 2 (please refer to Section 5.2 for information in Phases) was detailed in the "Interim EBRAINS Infrastructure Implementation Report" (D5.4, Section 2,[2]). In accordance with the Technical Coordination's design and implementation plan for EBRAINS, the Architecture has been revised and updated to reflect the latest development status, evolution, progress, and necessary changes that occurred during the project. The Architecture presented in this section provides the current version of the EBRAINS RI and proves to be a state-of-the-art, scalable, and sustainable architecture upon which the EBRAINS RI can continue to expand as a production-ready research infrastructure to support the activities of the neuroscientific community.

For completeness and context purposes, it is important to recall that the EBRAINS RI is an **entire ecosystem** for brain research, not limited to a single "platform" in the strict sense of the term, but rather formed through the aggregation of several subsystems that are composed of a set of centrally managed essential backend infrastructure resources and services, with an **Enterprise System (ES)** of **federated components** on top of that (running services and/or hosted products), and further augmented by a **System-of-Systems (SoS)** of autonomous confederated services provided by other components. The process of understanding, documenting, identifying, and mapping all sub-systems, resources, services and components, as well as accurately pinpointing their interfaces, inter/intra-connections and data/control flows in order to effectively define the structure, behaviour and appropriate views of EBRAINS RI, has understandably been a challenging endeavour and one of the core outputs of our work.

The various presented views of the EBRAINS RI were continuously iterated, reviewed, and updated meticulously throughout SGA3 to accurately reflect the development status and capabilities of the EBRAINS RI.

This section is outlined as follows:

- Section 2.1 recalls from D5.4 and **solidifies the architecture diagrams design approach**, the process that directed the effort, as well as a plan for continuously updating the implemented architecture diagrams and creating new ones as the EBRAINS RI is further developed and extended.
- Section 2.2 presents a more engaging and improved version of the EBRAINS High-Level Overview, aimed at better communicating the **EBRAINS RI narrative** to the neuroscience community, future partners, and potential collaborators in an effort to extend the user base.
- Section 2.3 revises the **EBRAINS High-Level Overview**, meant to assist the understanding of EBRAINS by external (non-SGA3) stakeholders.
- Section 2.4 revises the **Conceptual Architecture** of EBRAINS, meant to facilitate a general overview of EBRAINS mostly by stakeholders who are closely involved in the project (SGA3 members and partners).
- Section 2.5 revises the **Logical Architecture** of the EBRAINS RI at two levels of abstraction and detail.
- Section 2.6 presents the **Physical Deployment** of EBRAINS over the available computing, storage, and networking resources.
- Finally, Section 2.7, revises the overview of supported deployment options both for **current** and **future** EBRAINS component developers, aiming to educate and support them in the operationalisation of their services in EBRAINS. We place special emphasis on this line of work, as it aims to lower the entry barrier for new EBRAINS offerings on a technical level, thus assisting in the realisation of the long-term vision for EBRAINS, as a constantly evolving marketplace of services and offerings for neuroscience and the brain.

In essence, this section presents the natural evolution of the EBRAINS RI as originally presented in D5.4, following the implementation and developments during SGA3, and consequently the content structure is kept identical with D5.4. The section documents the final version of the Architecture by highlighting:

a) the design choices and capabilities that remained the same throughout the various stages of development,

b) the updates from the previous report, and finally,

c) deviations from the projected state to accommodate the evolving needs of the project.

2.1 Architecture diagrams design approach

As described in D5.4, the EBRAINS Architecture was influenced by the C4 model ([3], [4]) which is a lean graphical notation technique for modelling the architecture of software systems. Therefore, the EBRAINS RI Architecture was documented by showing multiple points of view that explain the decomposition of the RI into components, the relationship between these components, and where appropriate, the relation with the users. In essence, different architectural diagrams have been created featuring appropriate levels of abstraction for distinct purposes and audiences. These levels of abstraction can be viewed as different zoom levels on the RI moving from more abstract, wide views (Sections 2.2, 2.3, 2.4) to more representational, focused views (Sections 2.5, 2.6). A key concern was to not only accurately picture the different conceptual levels of the RI, but also to facilitate external service developers who want to integrate in the EBRAINS RI and need to grasp an extensive overview of the platform-level services and offerings.

Moreover, to better address the challenge of identifying and communicating the appropriate architecture components, relationships, and interdependencies, we relied heavily on *domain partitioning* of the architecture. This type of top-level partitioning of the architecture stands on the principles of Domain-Driven Design which is a modelling technique for decomposing complex software systems. This was crucial to our effort, as EBRAINS has an inherently complex structure. By following domain partitioning, the architecture revolved around top-level (i.e., fundamental) components focused on domains or workflows independent and decoupled from each other, rather than technical capabilities. This was particularly useful in the creation of the appropriate levels of abstraction and in turn the appropriate architectural diagrams, given also that one of the major contributions of EBRAINS are the scientific workflows that are provided to the community by exploiting the available tools and services.

The EBRAINS Architecture has been developed under the guidance, feedback, and review of the EBRAINS Architecture Infrastructure Working Group (**EAI-WG**), comprising scientific, engineering, and base infrastructure stakeholders, working together with the EBRAINS Technical Coordination (TC) responsible for detailing the EBRAINS RI Architecture. To ensure that the EBRAINS Architecture addresses requirements of all different stakeholders, the architecture mapping was an **ongoing** and **iterative** process with continuous structured input from all levels of the EBRAINS RI. Requirements were derived in both a **bottom-up** (from specific components requirements to infrastructure architecture) and a **top-down** (focus on large-scale infrastructure's requirements) approach, with validation/feedback performed by all stakeholders as necessary to identify missing elements, components, or functionality.

To mitigate the heterogeneity of the underlying components and the vertically focused priorities of each component team, as well as the need to address continuously evolving requirements and adhere to large-scale infrastructure best practices, we have opted for *loosely-coupled* integrations of sub-systems, components and services aiming to be agile, pragmatic and have moderate technical debt at all times: "*minimise existing and do not create new*". This approach in the integration process is also reflected in the architectural diagrams that are presented throughout this section. The architecture design, especially for the physical deployment diagrams, was performed with the aim of achieving demand-aware scaling capabilities, cost-effective deployments, and sustainable operation.

The following EBRAINS Architectural diagrams are available:

- RI Overview
- High-Level Overview
- Conceptual Architecture
- Logical Architecture
- Physical Deployment
- EBRAINS supported deployment options.

It is important to note that all the individual EBRAINS components diagrams were collected in the TC Collab, and at component-dedicated wiki pages, to fully document all aspects of the RI.

2.2 EBRAINS RI Overview

The EBRAINS RI Overview is presented in Figure 2, aimed at better communicating the **EBRAINS RI narrative** to the neuroscience community, future partners, and potential collaborators.

This diagram communicates the EBRAINS Architecture in a more abstract and engaging way compared to EBRAINS High-Level Overview and was considered an important addition in the various EBRAINS architectural views, with the need for it already documented in D5.4. In essence, it is an abstraction of the EBRAINS High-Level Overview of Section 2.3. As such, detailed documentation for all the entities that appear in the diagram can be retrieved from the following sections of this document, as well as from D5.4 in case it is needed (see D5.4, Section 2.2,[2]).

The EBRAINS RI Overview diagram also communicates the future role of the EBRAINS RI as a central hub coordinating a pan-European network of federated services delivered through National Nodes (NNs) -see also Section 8.5. NNs is a mechanism made up of EBRAINS members, who provide in an autonomous manner scientific expertise, software, services, and support addressing the thematic and local targeting of their communities. These services and facilities are also discoverable and accessible by the entire neuroscience community through the EBRAINS platform, thus assisting in EBRAINS user base growth, scientific exchange, and advancement.

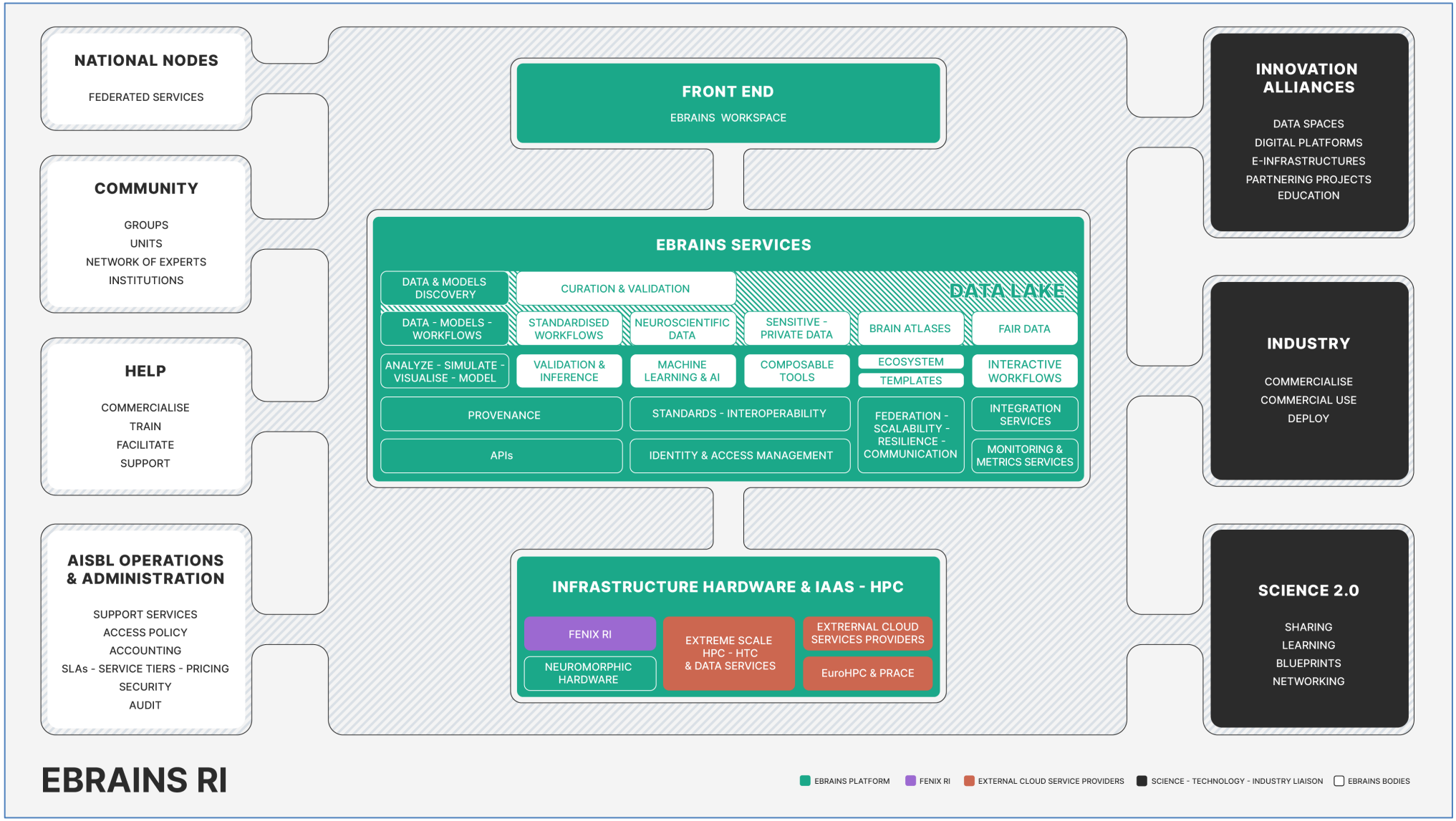

Figure 2: EBRAINS RI Overview

2.3 EBRAINS High-Level Overview

In this section, the evolution of the EBRAINS High-Level Overview after its initial conceptualization in D5.4 is presented. This diagram is an abstraction of the EBRAINS Conceptual Architecture (Section 2.4), which served as the first level of abstraction for the EBRAINS architecture during the initial stages of SGA3. The High-Level Overview diagram has proven particularly useful in conveying EBRAINS capabilities to external (non-SGA3) stakeholders.

It needs to be mentioned that the various components of the diagram have remained largely the same from D5.4. Therefore, the detailed documentation of the various entities presented in the diagram can be retrieved from that document in case it is required (see D5.4, Section 2.2, [2]).

The EBRAINS High-Level Overview diagram, which is presented in Figure 3, notably differs from the one presented in D5.4 in the following areas:

- The names of the EBRAINS categories of services were updated to reflect the recently updated EBRAINS Web Portal nomenclature. Namely, the changes are the following:
    - "Atlases" category was renamed to "Brain Atlases".
    - "Data & Knowledge" category was renamed to "Data".
    - "Simulation services" category was decomposed to two new categories to better convey the included offerings. The new categories are the following:
        - "Modelling, Simulation & Computing"
        - "Validation & Inference"
    - "Medical Data analytics" category was renamed to "Health research platforms".
- The most noticeable difference is that the "Sensitive Data" stack is not standalone anymore and has been placed inside the "Vertically Integrated Components' Stack" box (which previously was named "Applications Stack"). An adjustment was performed to reflect the fact that EBRAINS does not simply host applications but provides the required infrastructure and integration services (i.e., platform middleware) for hosting full-stack vertically integrated applications across all the provided EBRAINS categories of services.
- The capability of working with sensitive data mainly through the delivered Health Research Platforms which are GDPR-compliant and cover all the legislative requirements for handling sensitive data, is now adequately highlighted. It is worth mentioning that the "Human Data Gateway" service (inside the Sensitive Data stack of EBRAINS Services) is assumed by the Health Data Cloud (HDC) platform, which is part of Health Research Platforms.
- As discussed in Section 2.1, the focus in domain partitioning of the architecture is evident in the EBRAINS services layer. Reference to individual components in D5.4 in the Applications Stack (now Vertically Integrated Components' Stack) has been replaced with reference to the EBRAINS categories of services to better reflect the latest status of the architecture and be more inclusive.

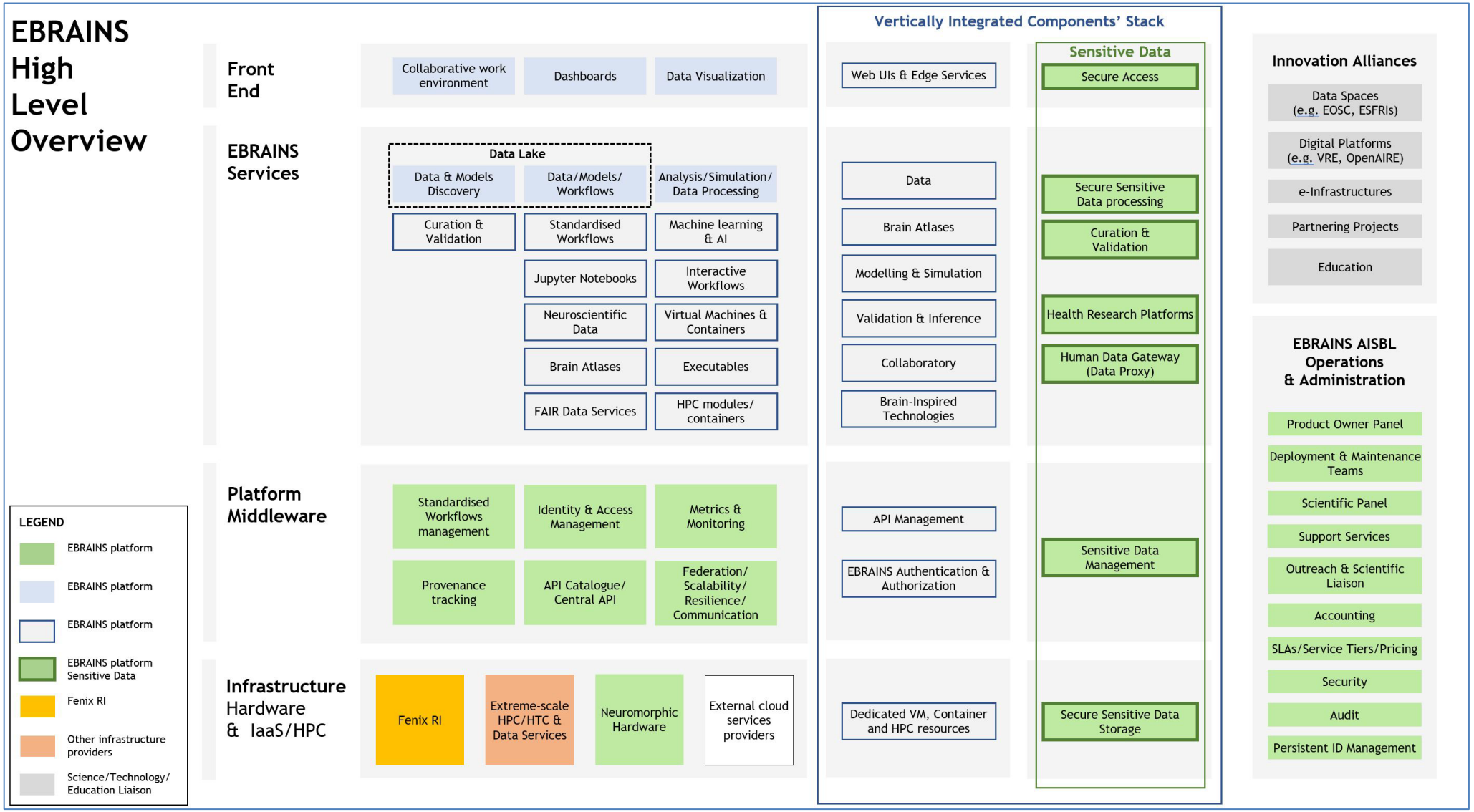

Figure 3: EBRAINS High Level Overview

2.4 EBRAINS Conceptual Architecture

In this section, the evolution of the Conceptual Architecture of EBRAINS (Figure 4) after its initial conceptualization in D5.4 is presented. This work is the direct output of the EAI-WG, which monitors the architecture infrastructure mapping efforts of EBRAINS (more information for the WG efforts can be found in the D5.3 (Annex V,[1]). It needs to be mentioned that the various components of the diagram have remained largely the same from D5.4. Therefore, the detailed documentation of the various entities present in the diagram can be retrieved from that document in case it is required (see D5.4, Section 2.2, [2]).

This view of the EBRAINS architecture is the next available zoom level right after the High-Level Overview of the previous section. It aims to better communicate the EBRAINS Services to internal end-users and others familiar with the EBRAINS ecosystem. High-level abstractions are favoured to facilitate communication, portray a common shared vision and function as a shared understanding between science practitioners and infrastructure experts.

The EBRAINS Conceptual architecture abstracts and collates the internal operational and computing requirements of each EBRAINS Service into a common set of software and computing components. What is presented here is *not* capturing the particular components of any EBRAINS services category, but rather the *types* of components that one or more categories of services have (or possible future offerings may have), thereby revealing an overall view of the EBRAINS RI. Each service offering is instantiated in the presented RI by mapping its structure onto the architecture, identifying where exactly the specific instances of its components correspond, and doing it in a manner that serves its scalable, trusted, and user-driven delivery. Furthermore, the required data assets and computing resources are reserved, allocated and provided to these instances in a unified manner by the platform ensuring secure, dependable, and uniform execution according to strict common policies (e.g. user privileges, data access, computing resources).

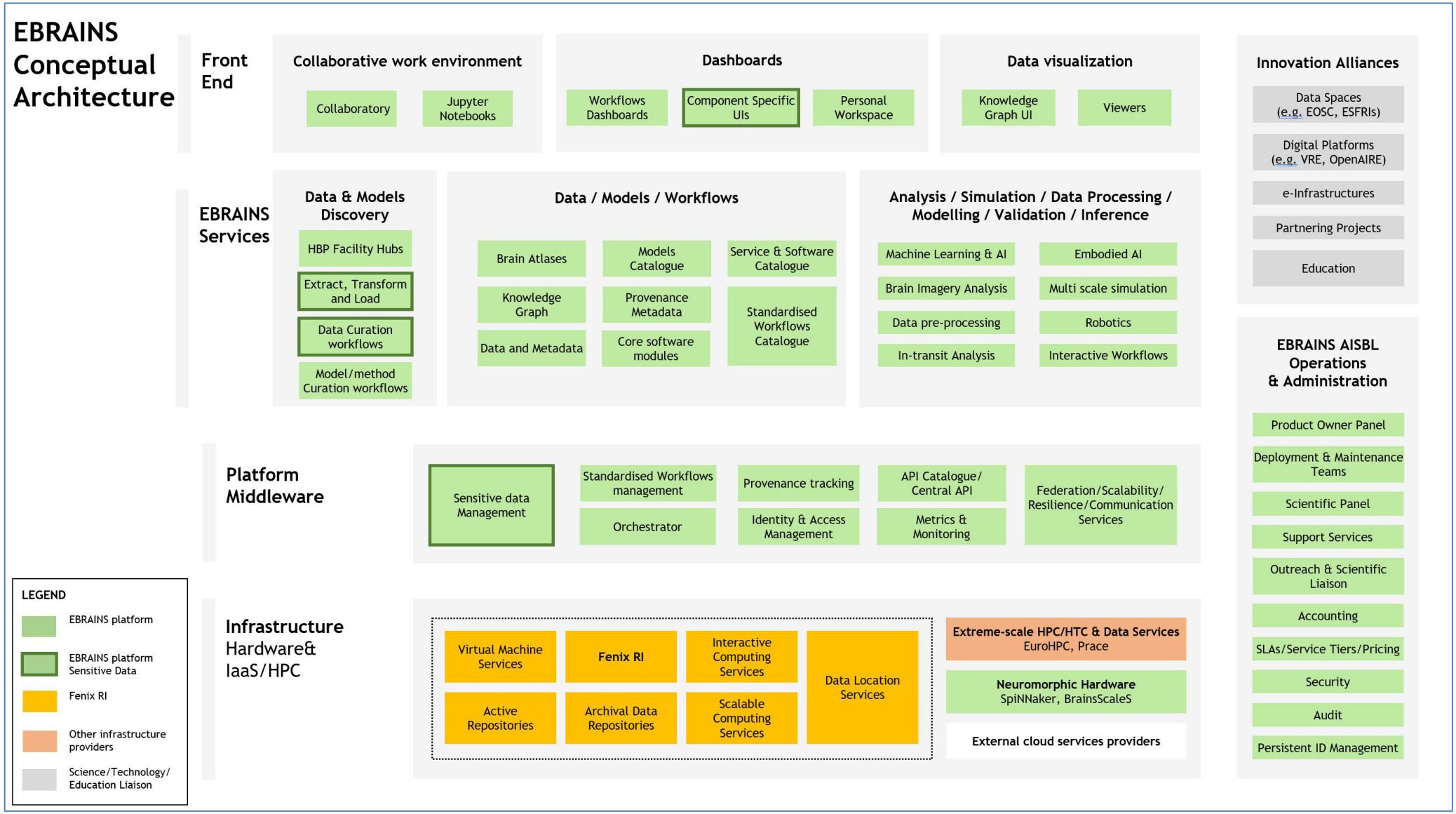

Figure 4: EBRAINS Conceptual Architecture

2.5 EBRAINS Logical Architecture

In this section the evolution of the EBRAINS Logical Architecture after its conceptualization in D5.4 is presented, which is the product of the continuous iterations on the logical architectural diagrams presented in D5.4 during late Phase 2 and Phases 3 & 4.

Please recall from D5.4 that the Conceptual Architecture was the result of an iterative process that followed a bottom-up approach (from the individual components, Service Categories and Work Packages to the infrastructure architecture) with some top-down influences from large-scale architecture requirements. Following this approach that detailed EBRAINS architecture on a high-level, the next step was to go deeper and identify the logical architecture. In essence, the main requirement was to utilise the Conceptual Architecture diagram of EBRAINS RI and based on that, move onwards with the logical architecture design of EBRAINS RI that would feature all the SGA3 software components in a single diagram.

Following the previous diagrams, the EBRAINS Logical Architecture displayed in Figure 5 and Figure 6 is abstracted into four layers. Namely the Front end, EBRAINS Services, Platform Middleware and Infrastructure layers that were thoroughly documented in D5.4. Moreover, the EBRAINS platform inherently supports: (i) **sensitive data** applications as it provides the required components for sensitive data handling, (ii) hosting of the **full stack of applications** (vertical integration) and services deployed alongside the core EBRAINS Services.

In Figure 5, the four abstraction layers of the EBRAINS architecture are displayed along with the EBRAINS Service Categories (SCs). Each SC provides several components that span the front-end layer to the infrastructure layer. Moreover, each component belongs to one or more application/service domains depending on the products it provides to the EBRAINS RI. The list of application/service domains offered in the EBRAINS RI is quite an extensive one and includes the following. Web Applications, Data Catalogue, Models Catalogue, Software Catalogue, Desktop Applications, Jupyter Notebooks, Online Office, Wiki, Software Libraries, Web Services, Web APIs, Data stores, Metadata management system, Software suites, Simulation Frameworks, Programmatic APIs, Modelling language, Workflow tools, Databases, HPC applications, Workflows recipes, Data platform, Authentication & Authorization services, Package management system, Software frameworks, JupyterLab and File Storage options.

In the diagram we slice the architecture per platform layer and SC and display the corresponding application/service domains that each SC offers in a particular platform layer. As in the previous sections, where necessary, the names of the EBRAINS categories of services were updated to reflect the recently updated EBRAINS Web Portal nomenclature.

Two notable additions of application/service domains that occurred during the later stages of SGA3 were:

- The addition of some components to the EBRAINS RI that contributed Community Apps (i.e. web applications inside the EBRAINS Collaboratory[4] [5]) to the "Brain Atlases" and "Modelling, Simulation & Computing" service categories.
- The operationalization of components from the "Health Research Platforms" service category contributed new Web Applications to that service category.

In Figure 6, we go one step further from Figure 5. Figure 6 places all EBRAINS components (for further details see D5.3, Section 3.7, [1] & D5.4, Section 2.4, [2]) in a logical architectural diagram to see what is in place, how components relate/interface with each other, the different processing levels and, ultimately, establishes a wider overview of EBRAINS RI. Moreover, it aims to display the current state of available products/services to provide a collective overview of EBRAINS offerings. Notable updates and differences from D5.4 are listed below:

[4] https://wiki.ebrains.eu/bin/view/Collabs/collaboratory-community-apps/
[5] https://wiki.ebrains.eu/bin/view/Apps/

- As in the previous sections, where necessary, the names of the EBRAINS categories of services were updated to reflect the recently updated EBRAINS Web Portal nomenclature.
- The logical architecture, as captured in D5.4, presented a total of 51 components that were formally communicated to the TC as eligible for integration during Phase 2. Since that point in time, the number of components increased **from 51 to 73**, with several factors contributing to this increase (please see Sections 5.1 and 5.2 for details). First, SGA3 output gradually matured and was delivered as independent components in the second half of the project. Second, several components revised their organizational structure, resulting in their split up into multiple new components. Third, new partners joined SGA3, offering their own software output as components to be integrated in the EBRAINS RI. All active components are documented in Figure 6.
- The "Monitoring Services" (see Section 3.4) developed by the TC team were added to the "Platform Middleware" layer.
- The Health Data Cloud (HDC) component that joined the integration during the later stages of SGA3 was added to the "Health Research Platforms" service category, along with MIP and HIP.

  The HDC provides EBRAINS with services for sensitive data since it allows to process GDPR-related data in compliance with GDPR structure data, annotate it with metadata, create versioned snapshots and provenance records, search data, stage data before exposure to Processors to comply with the principles of Data Minimization and Purpose Limitation introduced by the GDPR, collaboratively and securely process data within a team on dedicated VMs or HPC accounts as well as it provide auditability features to track all critical processing operations performed by the team of Controllers and Processors.

  Medical Informatics Platform (MIP) provides the possibility for a project team or scientific consortium to install and use a federated data platform that provides an interface for various investigators (clinicians, neuroscientists, epidemiologists, researchers, health managers) to access and analyse aggregated, federated medical data stored in hospitals, research centers and public databases, without moving the data from their place of origin, and without infringing patient privacy.

  Finally, Human Intracerebral EEG Platform (HIP), is an open-source platform designed for large scale and optimised collection, storage, curation, sharing and analysis of multiscale Human iEEG data at the international level.

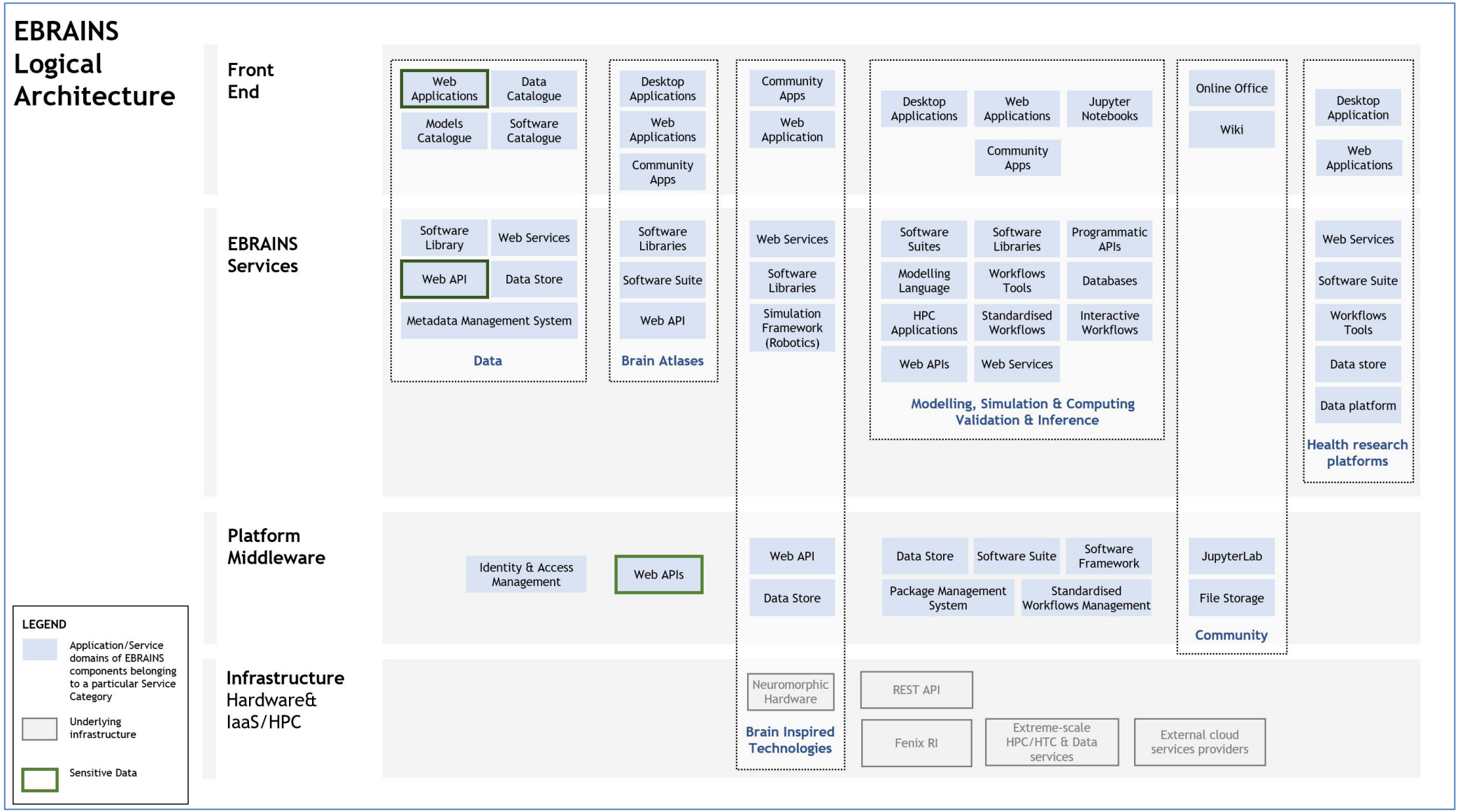

Figure 5: EBRAINS Logical Architecture

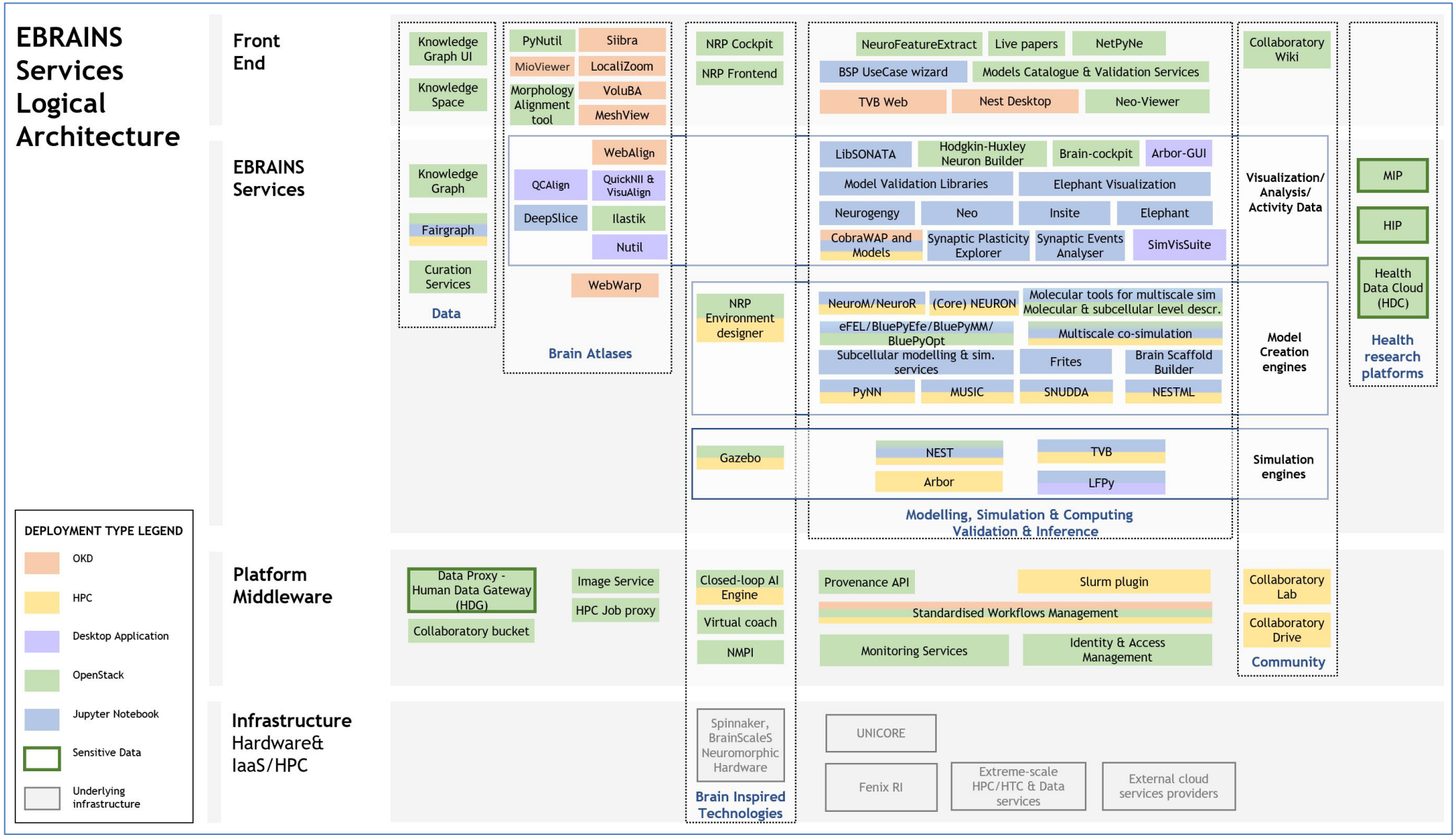

Figure 6: EBRAINS Services Logical Architecture

2.6 EBRAINS Physical Deployment

In this section, the evolution of the Physical Deployment of the EBRAINS RI after its conceptualization in D5.4 is presented. Since D5.4, all Fenix sites gradually came online giving the possibility to effectively distribute EBRAINS services wherever appropriate and necessary across the fully functional Fenix RI. Regarding the deployment, EBRAINS managed to fully utilise the dedicated resources and operate in a multi-site environment during SGA3 in order to scale efficiently (and thus accommodate increasing demand), offer load-balancing capabilities (balance load, improve QoS), be resilient (gracefully handle issues, increase uptime), and exploit data locality where possible (data and models need to be close to the processing environments).

In Section 2.6.1, the EBRAINS Physical Deployment at the end of Phase 4 (i.e., the end of SGA3) is presented in the diagram of Figure 7. As in D5.4, the necessary information that supplemented its creation was extracted from the latest updates to the "TC Collab" wiki pages for the EBRAINS components (D5.3, Annex IV,[1]), the timeline for ICEI services[6], ICEI available resources[7], Fenix AAI overview and updates (presentation slides from 1st FURMS workshop, SGA3 T6.6 Fenix AAI/FURMS Workshop), as well as individually collected EBRAINS components' physical deployment diagrams.

The same concept as in D5.4 is followed, which is to present the different available facilities (different locations) for EBRAINS as distinct vertical stacks and to display in each stack the installed/available platform/infrastructure services. External services as well as basic/core APIs are also indicated.

At the **bottom** of the stack the infrastructure services are presented (Fenix RI) featuring the VM services, Scalable/Interacting computing services, Storage services, Data mover/transfer or location services, as well as Accounting and Authentication services.

At the **middle** of the stack EBRAINS Platform middleware services are displayed. Middleware abstracts the infrastructure services to the EBRAINS platform services and provides the platform applications and services with easy access/utilisation of the underlying computing and storage resources in a scalable and fault-tolerant way.

At the **top** of the stack the EBRAINS platform applications/services are displayed.

Please recall from D5.4 that the different **sites** are represented with their respective acronyms, and are:

- Fenix RI sites
  - **CH, CSCS** (Centro Svizzero di Calcolo Scientifico/Swiss National Supercomputing Centre): Geneva, Switzerland
  - **DE, JSC** (Jülich Supercomputing Centre): Jülich, Germany
  - **ES, BSC** (Barcelona Supercomputing Centre): Barcelona, Spain
  - **FR, CEA** (Commissariat à l'Energie Atomique): Paris, France
  - **IT, CINECA** (Consorzio Interuniversitario del Nord Est italiano per il Calcolo Automatico): Bologna, Italy
- Neuromorphic Computing (NMC) sites
  - **UHEI** (Universitaet Heidelberg): Heidelberg, Germany
  - **UMAN** (University of Manchester): Manchester, UK

For illustration and completeness purposes, in the physical deployment diagrams the base/core EBRAINS infrastructure resources and services have been included in addition to the EBRAINS components. They are also shown in use-case context elsewhere in the logical diagrams, and

[6] https://wiki.ebrains.eu/bin/view/Collabs/tech-coordination-weeklies/Timeline%20for%20ICEI%20services
[7] https://fenix-ri.eu/infrastructure/resources/available-resources

important details are described in the appropriate architecture subsections as well as in D5.3, Annex IV, [1]. Some extra notes, not self-evident in the diagrams and not already covered elsewhere, are explained here.

Finally, in Section 2.6.2 the EBRAINS Physical Deployment after the end of SGA3 is presented. Great effort has been dedicated in securing the appropriate infrastructure resources to sustain EBRAINS RI after the end of SGA3 and transition to post-SGA3 operation in a streamlined manner that would make the transition seamless and transparent to the EBRAINS RI end-users.

2.6.1 End of SGA3

The state of EBRAINS physical deployment at the end of SGA3 is presented in the diagram of Figure 7. It captures the result of the evolution of Physical Deployment from Phase 2 (as presented in D5.4) through Phases 3 and 4. The major entities that appear in the diagram have already been documented in D5.4 so only the updates/differences/deviations are documented in this section.

The goal **for all sites to be fully operational** (all base infrastructure services online) with Accounting and Fenix AAI (federation services) running has been achieved as one can observe from the diagram. The EBRAINS core services, including Monitoring services (both internal and external) which are fully operational supporting the EBRAINS DevOps team, are strategically deployed at chosen sites to optimally serve EBRAINS end-users. This comes in contrast to the projected state of EBRAINS at D5.4 which prescribed that EBRAINS core services are installed on all Fenix sites. The projected state of D5.4 was revised and fine-tuned during Phases 3 and 4, based on the demand for capacity and the need to minimize operational workload for the DevOps team, thus, creating a scalable and efficient, yet sensibly distributed RI.

The federation of resources is transparent and as a result all workloads/services have the foundation to be deployed and scaled anywhere in the federation. Critical EBRAINS applications as well as platform middleware have been deployed in a “Highly Available” scheme for fail-over and load-balancing purposes in cases where it was deemed appropriate and feasible within the implementation plan.

At the end of SGA3, the following points relevant to the Physical Deployment need to be mentioned to fully document its final state:

- The EBRAINS components (not Fenix RI offerings or EBRAINS Platform middleware components) placed inside the vertical stacks do not depict the deployment of all active EBRAINS components as this would make the diagram too cluttered. Only a fraction of the components is included, with the aim to showcase the components dispersion among the sites for some key components. With that being said, most of the components from all service categories are deployed at CSCS followed by JSC. CINECA, CEA, and BSC host a small number of components as a result of their late introduction as available alternatives.
- All the Fenix RI sites are now operational.
- At the end of SGA3, CSCS is considered the de facto main site (supporting a better range of workloads, with the base infrastructure services all online) and JSC is considered as the de facto secondary site.
- Regarding the EBRAINS Platform middleware services:
    - NMPI, Collaboratory bucket, Image Service, HPC Job Proxy and Provenance API were deployed only on CSCS as the provided capacity was considered adequate to cover the end-user requests.
    - OKD container orchestration engine and JupyterHub were deployed at two sites (CSCS & JSC) for High Availability and load balancing purposes.
    - EBRAINS tool software stack was available at all sites. More specifically, the software stack has two deployment targets within EBRAINS RI. The Collaboratory Lab (CSCS & JSC) and the HPC systems (potentially at all sites, validated at JSC, CINECA & BSC, under testing at CSCS and CEA). This offering is further detailed in Section 3.1.

  - Standardised workflows management component is deployed at two sites (CSCS &JSC) as it is leveraging OKD deployment, and this is where most of the end-user traffic is directed. This offering is further detailed in Section 3.2.

- Advancements also happened with respect to sensitive data, with the MIP and HIP components rolling out to production and the Health Data Gateway (HDC) initiating the provision of sensitive data services that would eventually render the Human Data Gateway (HDG) component obsolete.
- A certain number of EBRAINS Services employed multi-site deployments to cover the end-user demand for capacity.
- TC Monitoring services can now render metrics from wherever components are installed within the federation.
- The creation of a dedicated global API Gateway prescribed in D5.4 did not eventually prosper due to the fact that the majority of APIs were mostly deployed on the EBRAINS middleware OKD cluster and as a result OKD's internal API service acted as the main entry point for handling API calls.
- Several components designated to feature multi-site deployments during Phase 2 (i.e., Projected Physical Deployment) did not fulfil this projection due to the different security policies across Fenix sites.
- The dispersion of deployments, both for EBRAINS Platform middleware services and EBRAINS Services, favors the sites that were operational at the first phases of SGA3. As identified in D5.4 (Section 5.4: "Delays and Risks"), a tendency founded on practical considerations was observed from the development teams; to attach/trust the sites that were fully operational early-on (CSCS, JSC), which already offered ample features, detailed documentation, and fully developed administrative processes. This facilitated the quick onboarding of new projects and created a trend in which most of the development teams were keen on deploying their applications on those two sites that "felt" to be more experienced, dependable, and user-friendly even though all sites of the Fenix RI ultimately expose the same set of services.

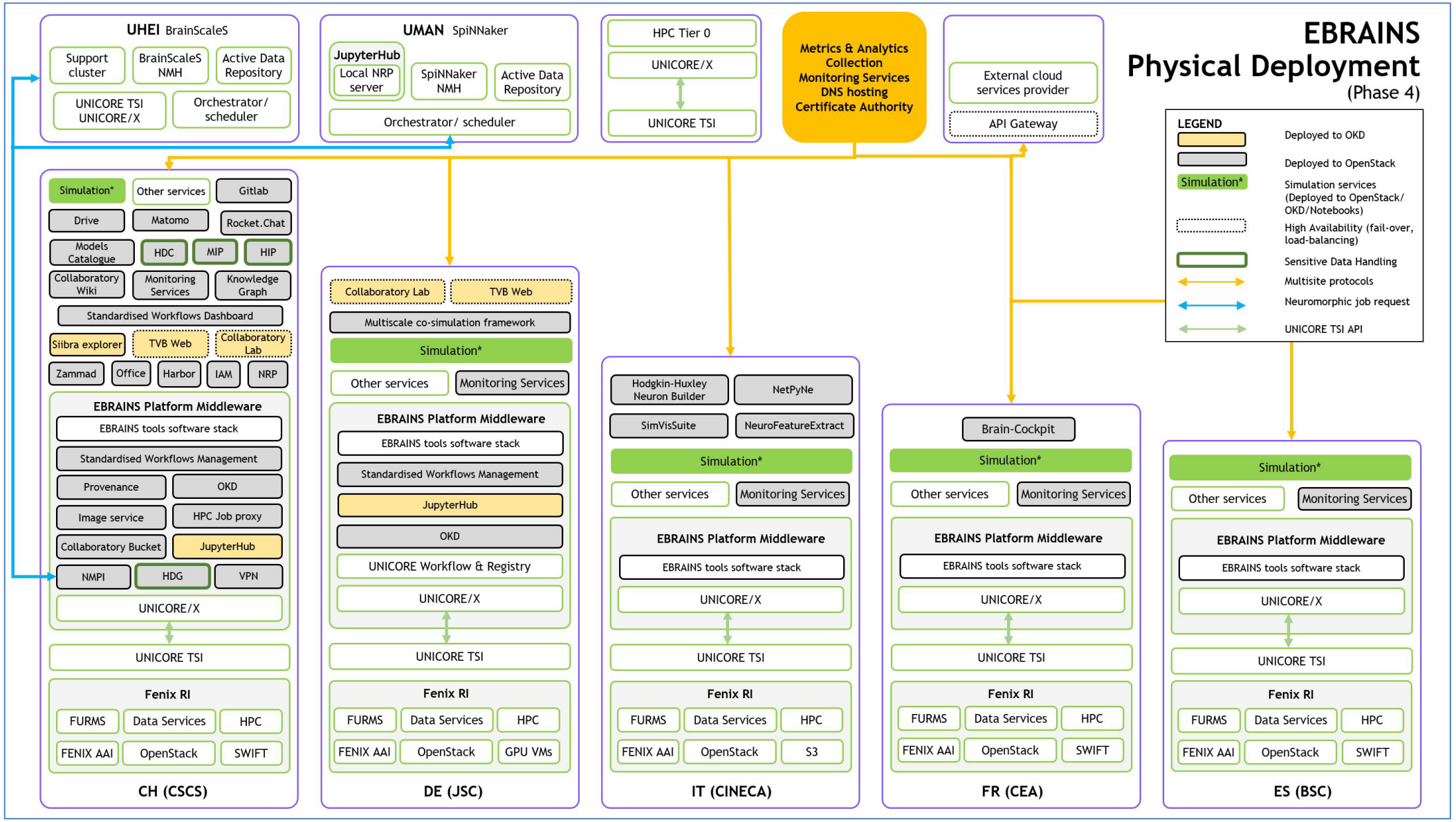

Figure 7: EBRAINS Physical Deployment Phase 4

2.6.2 Post-SGA3

In this section, the projected EBRAINS Physical Deployment after the end of SGA3 is presented. As mentioned in Section 6.2, significant effort has been dedicated in securing the appropriate infrastructure resources to sustain EBRAINS RI after the end of SGA3 and transition to post-SGA3 operation in a streamlined manner that would make the transition seamless and transparent to the EBRAINS RI end-users.

The currently allocated Fenix-provided SGA3 base infrastructure resources will not be available after the end of SGA3, and specifically by the end of 2023. However, the EBRAINS RI needs to continue its uninterrupted operation beyond that point. Currently, the base infrastructure committed for the continued operation of the EBRAINS RI are the JSC Cloud (from JSC) and ADA cloud (from CINECA). In Figure 8, we can inspect a preliminary view of the EBRAINS Physical Deployment post-SGA3 that will act as map to facilitate the migration process, guarantee optimal utilization of the committed base infrastructure resources, leverage the lessons learned from the SGA3, and ultimately ensure the uninterrupted provision of EBRAINS offerings to the end-users.

For more details regarding the available base infrastructure resources and the ongoing migration efforts towards this post-SGA3 operation of the EBRAINS RI, please see Section 6.2.

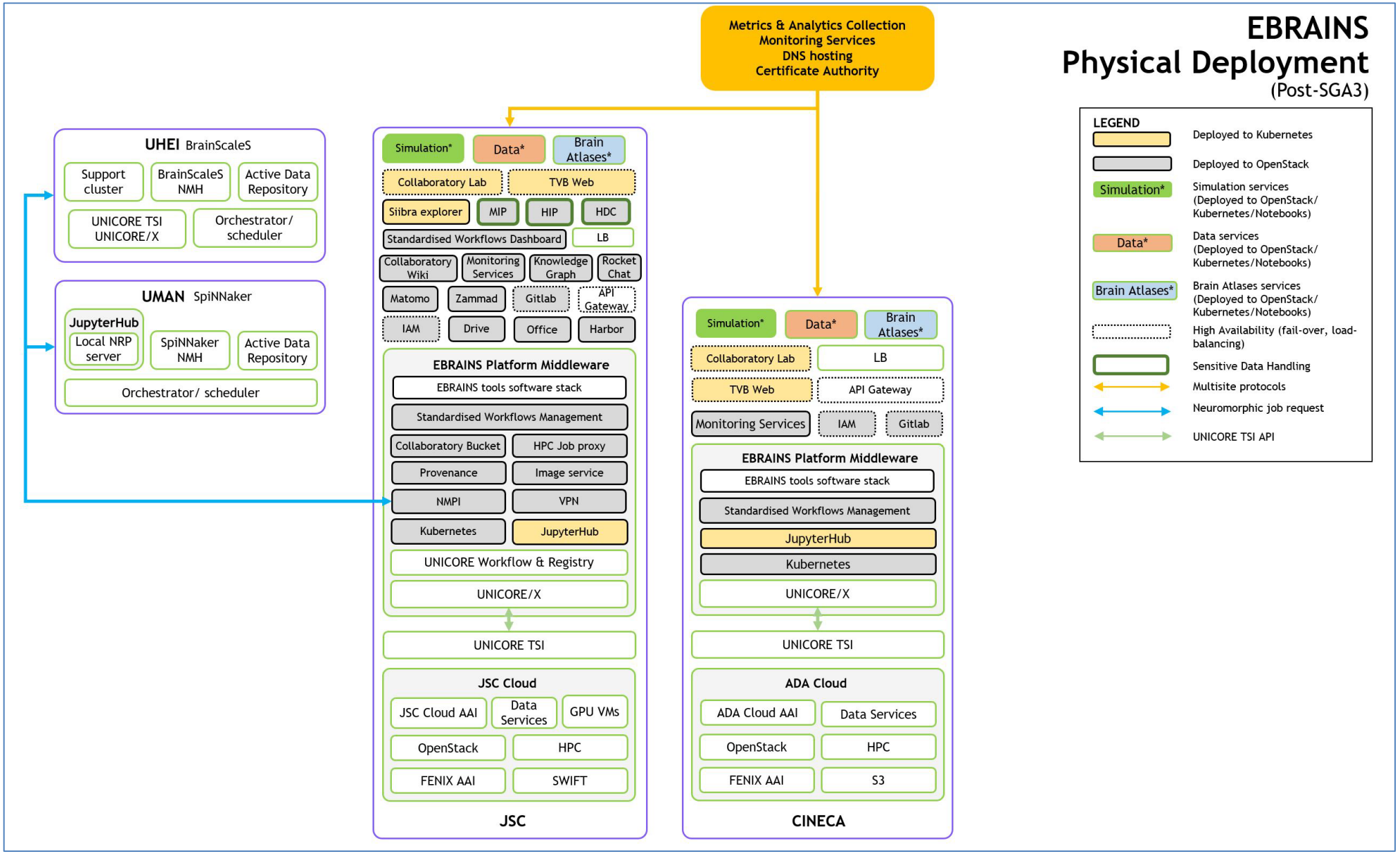

Figure 8: EBRAINS Physical Deployment post-SGA3

2.7 EBRAINS Deployment Options for Component Developers

This section, as in D5.4, provides EBRAINS application/service developers with an overview of the deployment options that are available in the EBRAINS RI. It also presents the different services, developer tools, runtime services and storage capabilities, and how to access those resources. This outlines every resource presently provided, but please note that some are provided currently for optional/recommended use, while others constitute required methods for interacting with the platform.

The initial version of this section was presented in D5.3, Section 3.8, [1]. The updated version of this section was presented in D5.4, Section 2.6, [2]. We include all the respective sections in the present document to outline the updates that occurred during late Phase 2 and through to Phases 3 & 4.

Significant updates are:

- The “Featured workflows” addition following the standardization of specific Showcases.
- The roll out to production of the “Standardised Workflows Section Dashboard”.
- The availability of “EBRAINS tools software stack” in the Collaboratory Notebooks and the HPC Libraries deployment types (see Section 3.1).
- The introduction of a dedicated publicly accessed API catalogue to facilitate all the available APIs [5].
- Where necessary, the names of the EBRAINS categories of services were updated to reflect the recently updated EBRAINS Web Portal nomenclature. This resulted in slight adjustments to all the diagrams regarding data and access control flows.

2.7.1 Overview of EBRAINS Deployment Resources

We need to recall for completeness, from D5.3 and D5.4, that EBRAINS is composed of a set of centrally managed essential back-end infrastructure resources and services, with an Enterprise System (ES) of federated components on top of that (running services and/or hosted products), further augmented by a System-of-Systems (SoS) of autonomously run confederated services provided by other components.

The centrally managed backend elements include resources like object storage, VM cluster, container cluster, HPC access, and API gateway. They also include runtime services like Data mover/transfer/location services, workflow-management, and monitoring, in addition to development services like registries and repositories for source code and packaging, a continuous integration and delivery platform and agile management tools. Because a large part of the RI, and essentially all the “domain logic”, even very “low level” logic, is provided as components, these constitute a complex network of “consumers” and “producers”, of which the only absolute boundaries are the non-components (the “base infrastructure” at the bottom and the “end users/researchers” at the top).

The component owners contributing to the ES accept the higher integration and standardisation requirements in order to benefit from deduplicating development-burden (for example offloading non-domain-logic like network and authentication management to provided shared services), reducing maintenance/Site-Reliability Engineering (SRE) burden by not self-hosting, and by participating in a broader shared quality-assurance process. Conversely, component owners contributing to the SoS do so when other factors offset the benefits of being in the ES, for example to retain greater managerial and/or operational autonomy, or to provide their functionality as an independent service in tandem with having it subsumed as part of the RI.

To assure minimum thresholds for the RI, there are still QA requirements of SoS components (see Section 5.2), but these inherently exclude some of the requirements of ES components. For an independently hosted "encapsulated" service many platform-level library-dependency/packaging issues become irrelevant, but as a balancing factor this also means the responsibility for continuity of that service remains on its component-owners.

2.7.2 How to integrate components into the EBRAINS RI

Except for edge-cases[8], there are **five deployment types** (Figure 9) available to component owners within the EBRAINS RI:

- Virtual Machines hosting long-running Services on top of OpenStack.
- Containers hosting long-running services on top of OKD Container Orchestration Platform.
- **Jupyter notebooks** hosted as ephemeral, on-demand services in the **Collaboratory** environment.
- The EBRAINS tools software stack, standalone HPC libraries, and container images (applications) hosted as reusable products within Scalable (and interactive) Computing Services.
- **Standardised workflows** hosted as structured defined recipes for automatic execution on top of **HPC systems** as well as **OKD**.

In many cases, a single "component" is in fact a "stack" of more than one of the above types. The long-running services can provide web-services, web-interfaces, or a combination of the two. One small exceptional addition to the "main" types would be software client-libraries as "downloadable products" for interoperating with the larger APIs, where such convenience and deduplication of effort is justifiable enough to slightly increase coupling by bending the "service calls only via API" rule. The coloured arrows (and their legend-box) in the following diagrams indicate dataflow and access relevant to each deployment-type.

Where necessary, the names of the EBRAINS categories of services were updated to reflect the recently updated EBRAINS Web Portal nomenclature. Namely, the changes are the following:

- "Atlases" category was renamed to "Brain Atlases".
- "Data & Knowledge" category was renamed to "Data".
- "Simulation services" category was decomposed to two new categories to better convey the included offerings. The new categories are the following:
    - "Modelling, Simulation & Computing"
    - "Model Validation & Inference"
- "Medical Data analytics" category was renamed to "Health research platforms".

Moreover, as previously mentioned, the "Featured workflows" were introduced as an offering of the "Standardised Workflows" platform resource following the standardization of specific Showcases. In particular, the capabilities of Showcase 3 - featuring the Mouse TVB Model notebook - and Showcase 4 - featuring the demonstration of the performance of 3-layer spiking neural network for predictive coding - were integrated in CWL workflows and offered in a standardised way to the end-users. For more information on this topic see Section 3.2.1.

Finally, it needs to be mentioned that the "VM registry" and "FaaS/Serverless" functionalities that were prescribed in the respective diagram in D5.4 were dropped as there was not adequate development traction to justify the operational overhead for providing such "Developer Services".

[8] One important edge-case comprises the Neuromorphic Compute Systems (NMC), consisting of (among other entities) of real hardware systems in Manchester (SpiNNaker) and Heidelberg (BrainScaleS) providing special services (i.e., hardware based accelerated analog physical model of spiking neural networks) as part of the EBRAINS RI

2.7.2.1 Deployment type I – VM-hosted services

This deployment type did not have any changes/updates/deviations from D5.4. We are reminding that Virtual Machines can be delivered as provisioning-playbooks/recipes, which are then built as images and associated with the specified VM-instance flavour. The images can then either be auto deployed to provide services, or deployed by other developers as part of theirs, on the OpenStack cloud computing platform. The VMs can be used to host any type of application - for example a web application, a web service/API, or a data store. Applications running in VMs deployed by accredited members can access all EBRAINS services, the DevOps tools, can submit jobs to HPC systems as well as to NMC systems (part of EBRAINS services) and can have access to object storage for long-term, safe, and secure storage. Project-level Administrator access to OpenStack for managing the VMs can be accomplished either via web dashboard, CLI, or programmatic access. At present, if automated config-management and high-availability for VM-based services is desired, it must be provided by the component owners (Figure 10).

2.7.2.2 Deployment type II – Container-hosted services

This deployment type did not have any changes/updates/deviations from D5.4. We are reminding that containerised applications can be delivered as a combination of “container images” (custom templates and customised upstream templates for building containers) and “Helm charts” (container-deployment templates - “container packages”). If Helm charts are not feasible, OKD Templates may be considered instead. They are then built and auto-deployed to provide services along with any associated high-availability/config-management settings (and/or made available for deployment by other developers as part of their own services, or for auto-deployment as part of user-workflows, notebooks, etc) all on the OKD Container Orchestration Platform. This format is recommended especially for stateless applications, or for stateful applications where high-availability, configuration-management, dynamic horizontal-scalability, or sandboxing is worth the extra abstraction-layer. Containerised applications deployed by accredited members can access all EBRAINS services, the DevOps tools, can submit jobs to HPC systems and can have access to object storage for long-term, safe, and secure storage. Project-level Administrator access to OKD for managing the containers can be accomplished either via web dashboard, CLI, or programmatic access (Figure 11).

2.7.2.3 Deployment type III – Collaboratory Notebooks

Jupyter notebooks can be delivered for manual ephemeral deployment by users in the Collaboratory environment for interactive computing. JupyterHub hosts sandboxed JupyterLab instances for running live code that can connect to all EBRAINS services and can submit jobs to EBRAINS HPC systems and additionally to EBRAINS-internal specialised computing resources like Neuromorphic hardware via dedicated middleware. Several HPC systems provide dedicated lower-latency job-queues to facilitate this interactive computing mode (those queues of course have stricter resource-restrictions than the typical heavyweight queues for standard HPC loads). Although not displayed in these diagrams, running live code, that can submit jobs to EBRAINS-external specialised computing resources like “HPC Tier 0 resources”, is also a possibility. Deployed notebooks can access EBRAINS software available through curated and tested software (whether libraries, modules, container/VM images, etc.) to run their experiments. Developers can bundle software in those formats and make them available to all EBRAINS users. Since late Phase 2 and through to Phases 3 & 4 the “EBRAINS tools software stack” (see Section 3.1) is available at the Collaboratory Notebooks offering transparently for use in the Collaboratory the majority of EBRAINS software libraries. (Figure 12)

2.7.2.4 Deployment type IV – HPC Library / Container runtimes

Executable code can be “deployed” (released, installed, and configured for runtime-activation by users rather than auto-activated) to the HPC resources, either as self-contained applications to be launched by service/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 like Sarus, Shifter or Singularity. Encapsulating an application in a container image can be easier than providing all dependencies for the application via operating system packages. Since late Phase 3 the "EBRAINS tools software stack" (see Section 3.1) is also available at the HPC facilities available to EBRAINS thereby offering the transparent use of the majority of EBRAINS software libraries (Figure 13).

2.7.2.5 Deployment type V - Standardised Workflows

During Phases 3 & 4 the Standardised workflows facility was further extended and solidified. Standardised workflows define computational workflows in a common and structured way, with steps associated with non-interactive command line tools receiving specific types of input parameters and provided output. Component developers can bundle non interactive command line tools in different ways (containerization methods, package managers, and more) to support dependency resolution, configuration management, software versioning and reusability of the workflow recipes as digital assets. Such an example is the standardisation of the workflows of Showcases 3 and 4. Standardised workflow management is responsible for taking care of the load balancing in the different underlying infrastructure (OKD, different HPC systems) in which workflow engines are deployed for automatic execution, monitoring, fetching logs, outputs, and handling restart of failed steps. From the user perspective, they are able to submit (deploy) standardised workflows to endpoints via a dedicated interface. This interface is the CWL-TES client for workflow submission to the backend, which is the TESK server installed on OKD for workflow execution. In this Deployment type, the Toil software is used instead of UNICORE for submitting workflows to the HPC backend. More information for the Standardised workflows can be found in Section 3.2 (Figure 14).

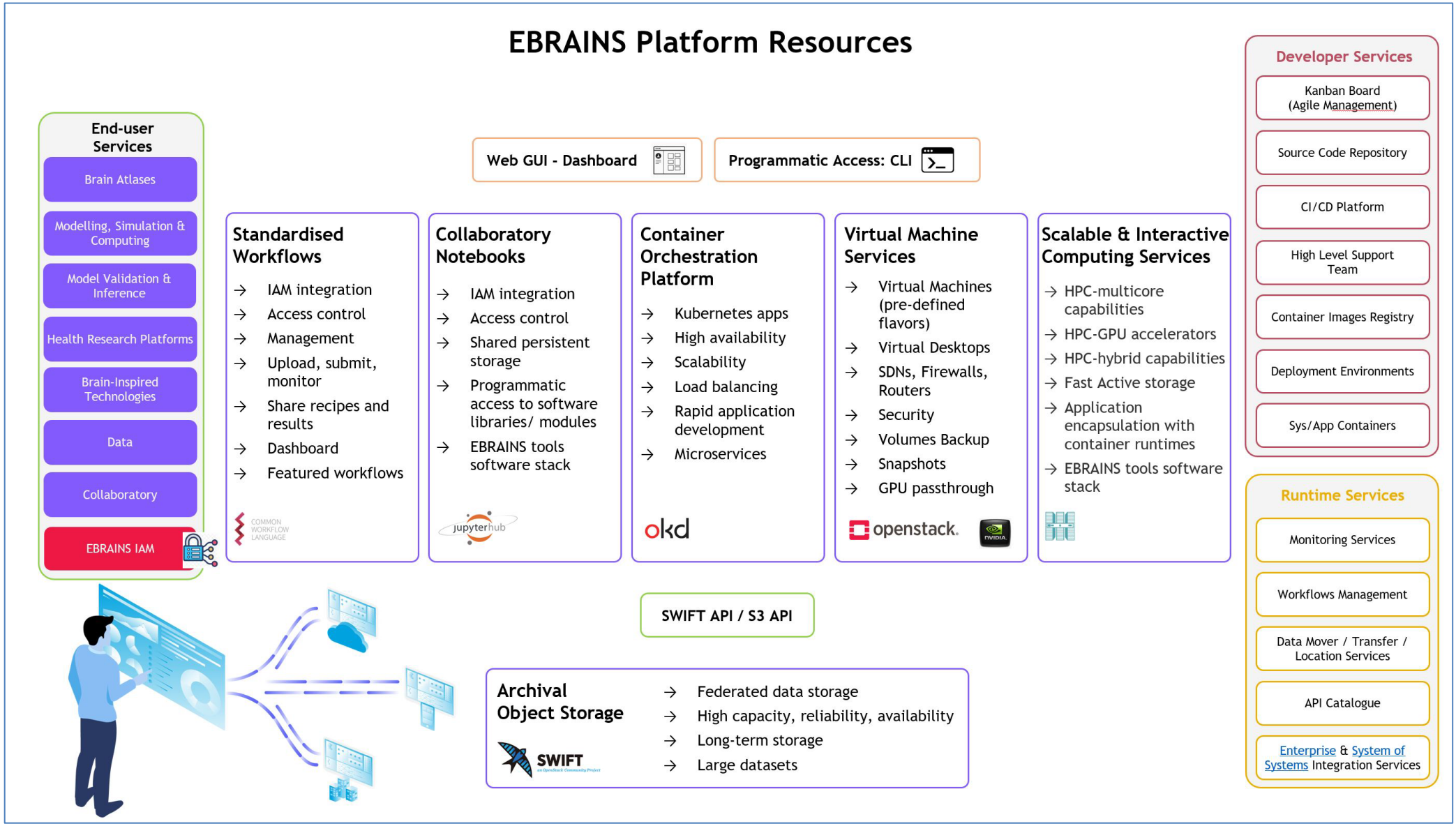

Figure 9: EBRAINS Deployment Resources

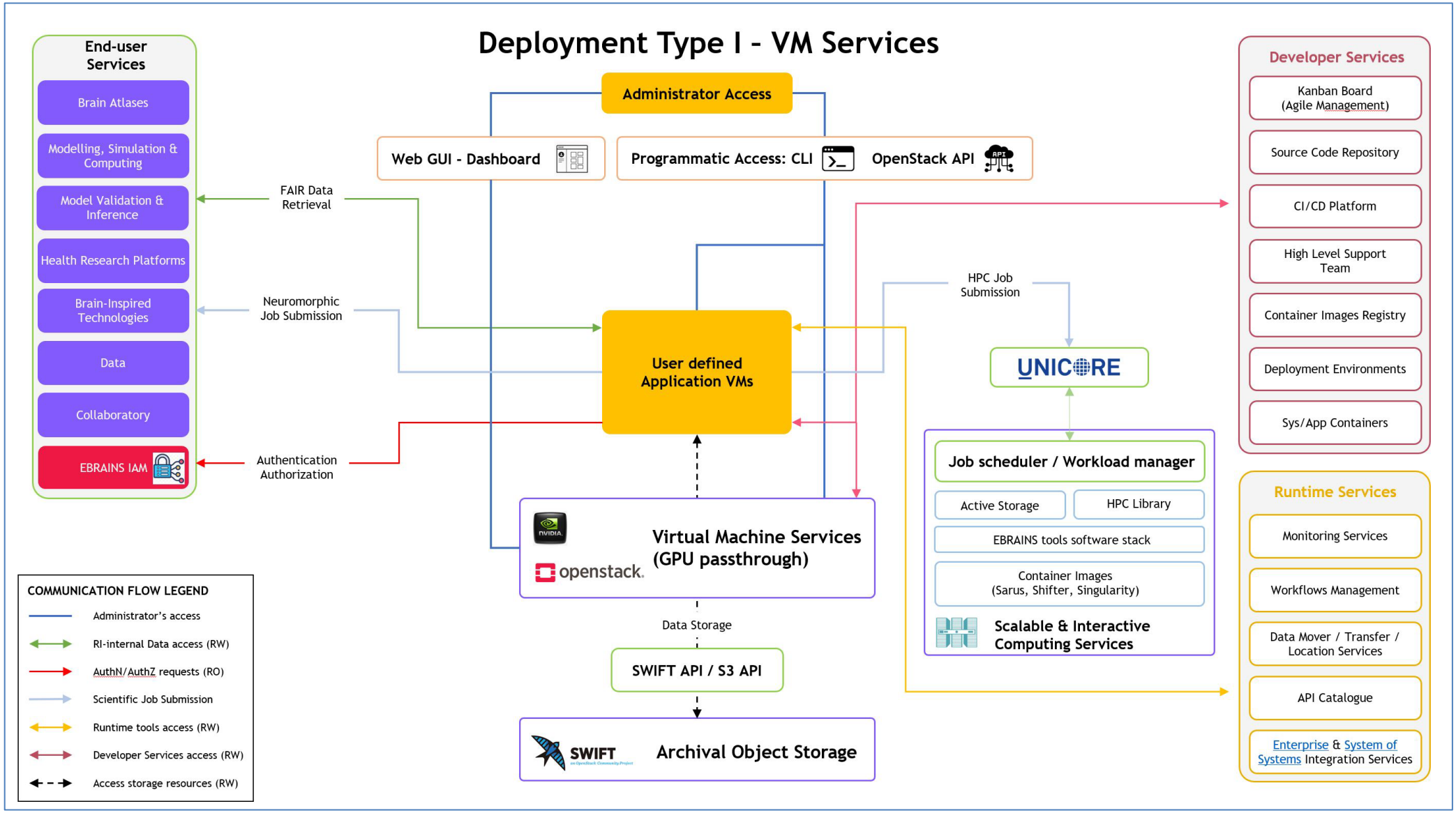

Figure 10: Deployment Type I - VM Services

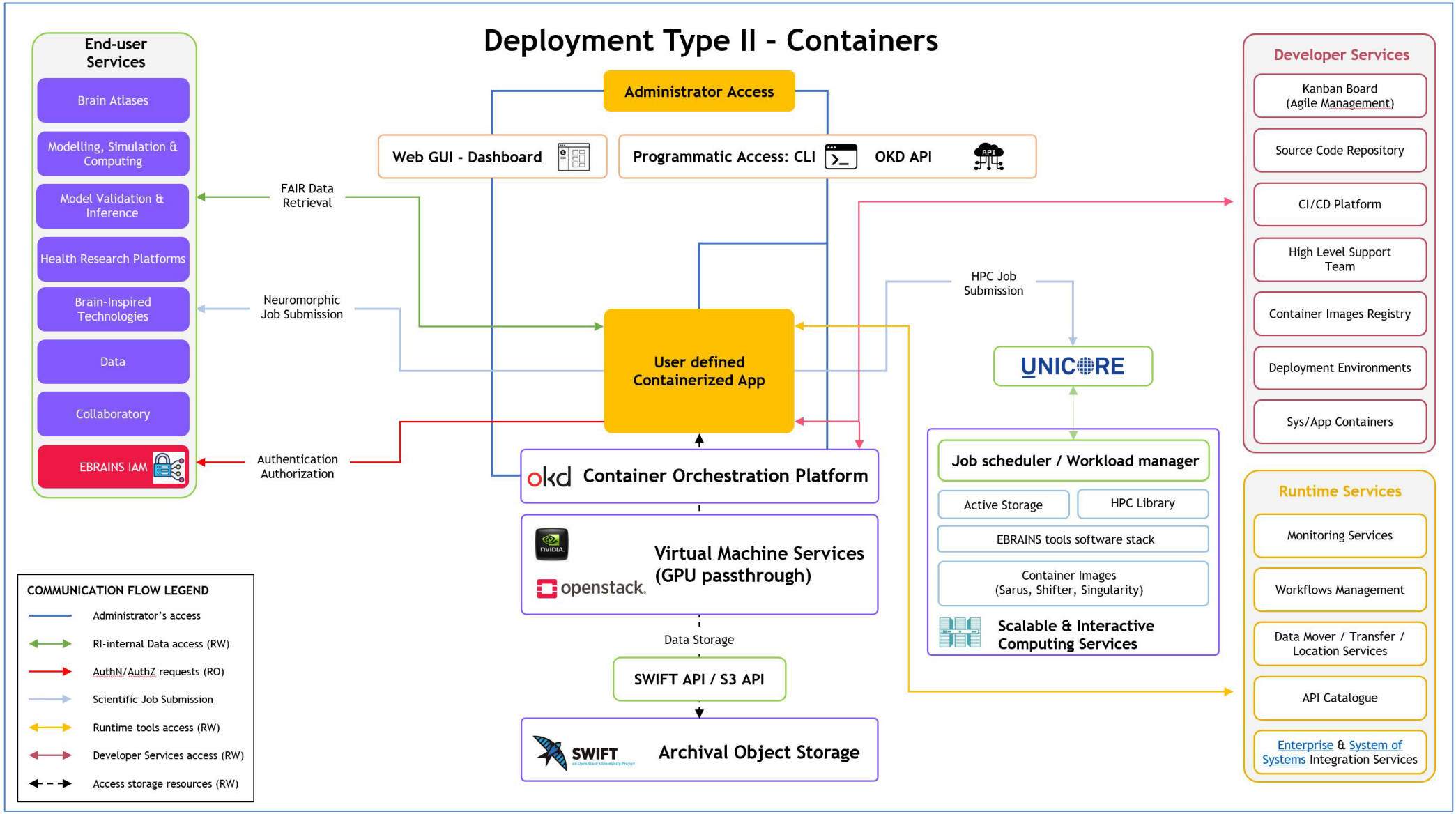

Figure 11:Deployment Type II - Containers

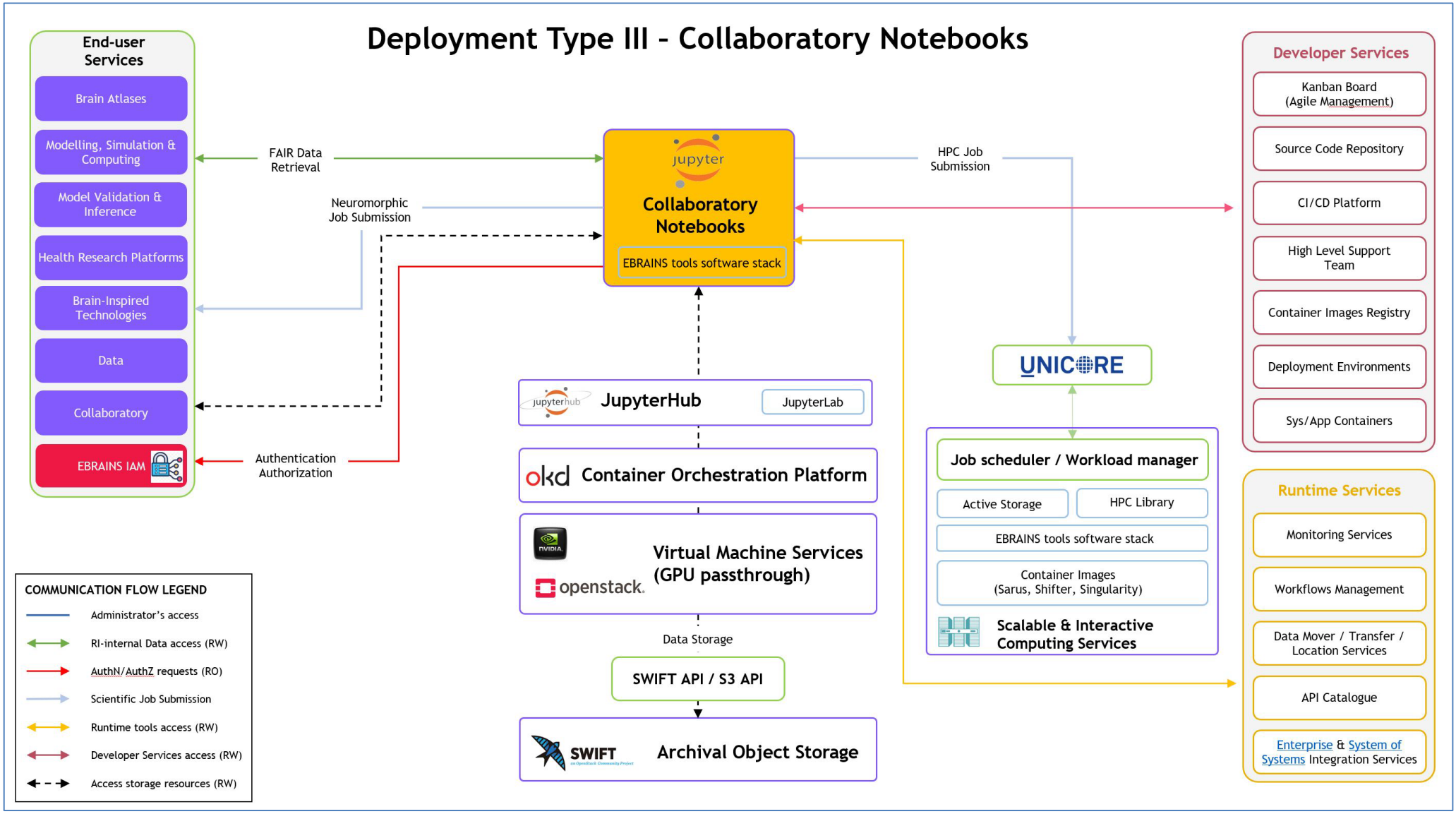

Figure 12: Deployment Type III - Collaboratory Notebooks

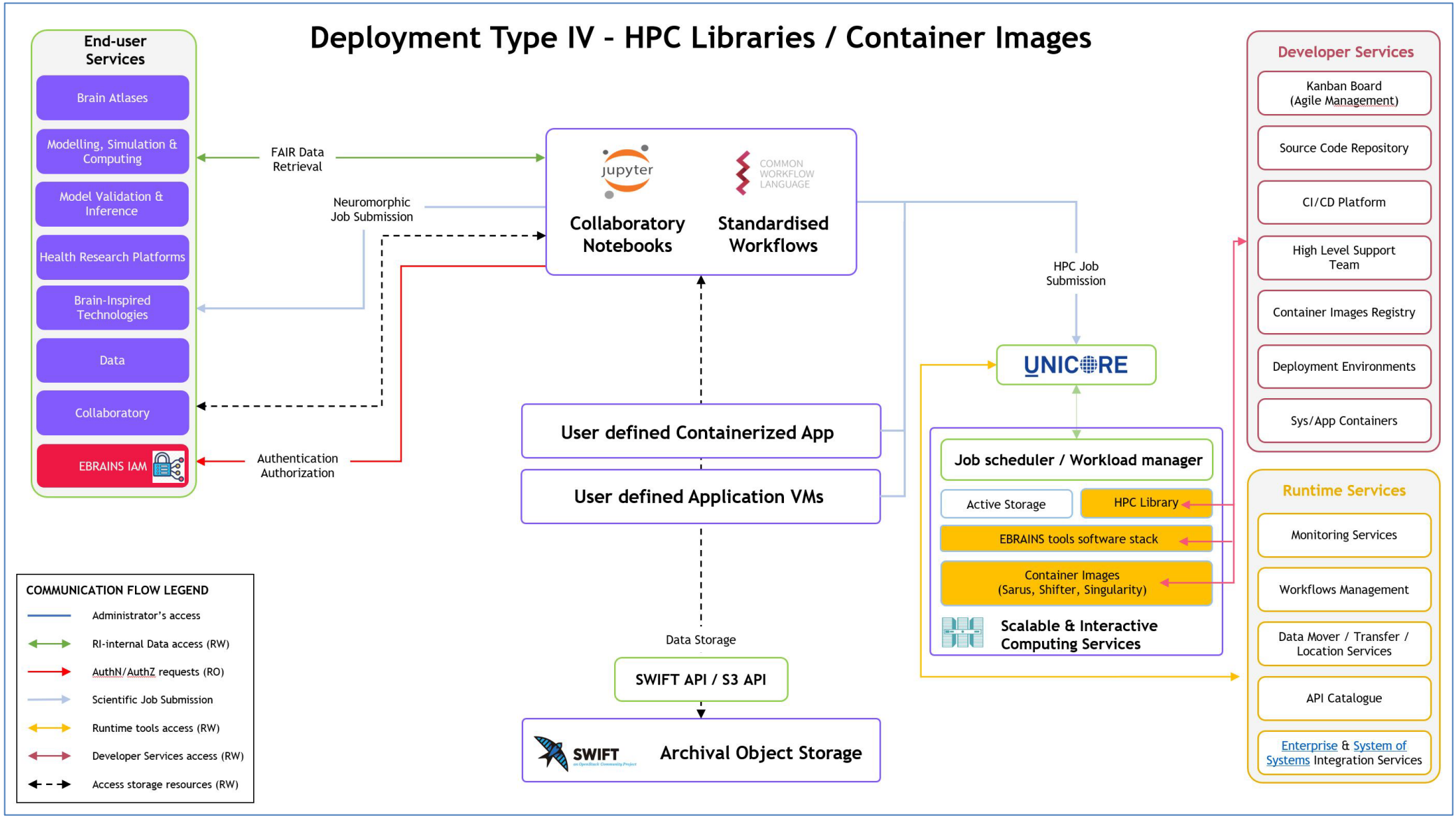

Figure 13: Deployment Type IV - HPC Libraries/Container Images

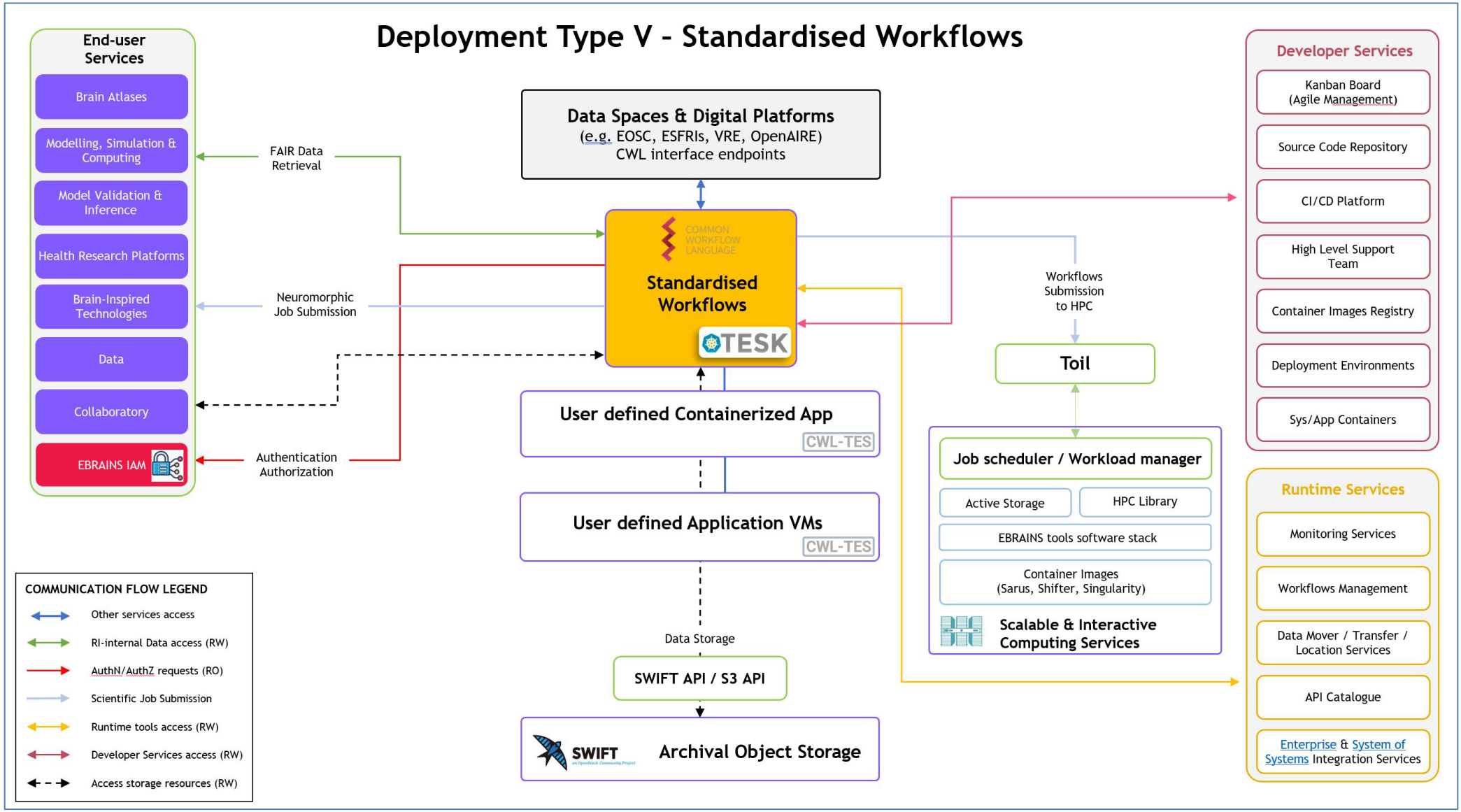

Figure 14: Deployment Type V - Standardised Workflows

3. EBRAINS Platform Services

This section presents the different tools of the EBRAINS platform services developed (middleware and front-facing) to (i) facilitate application/service developers in integrating their offerings into the EBRAINS RI, (ii) enable the detailed monitoring of the innerworkings and proper quality-assured operation of the EBRAINS RI, and (iii) enable users to harness standardized scientific computational workflows that use EBRAINS data, tools, models, and software.

3.1 Software Delivery and Deployment

One of the most important steps towards achieving the seamless integration of all components to the EBRAINS platform is the definition of a unified, consistent software environment containing all EBRAINS simulation engines and software tools as well as an efficient, optimised delivery strategy for deploying it to the different execution environments available to EBRAINS users.

Specifically, the end goal is to provide users with access to the diverse array of EBRAINS simulation engines and tools without requiring any actions to install, load or combine them, while at the same time streamlining the process for Component Owners (COs) and software developers to contribute and make their software accessible to the EBRAINS user base. In order to address the challenges introduced by the diverse range of tools and the increasing complexity of resolving their conflicting requirements, a delivery strategy leveraging Spack, a specialized package manager for scientific software, has been implemented during Phase 2, as presented in D5.4 (Section 3.1, [2]), which was further optimised and extended during Phases 3 and 4.

While this effort was at first solely focused on making EBRAINS software available in the Collaboratory Lab environment, the wide adoption of the standardised, version controlled official EBRAINS releases that were the outcome of this process, was embraced as an excellent opportunity to also extend this integrated software environment to HPC systems.

This section provides an overview of this delivery strategy, also focusing on the specific requirements of the delivery process within this context, offering insights into the fundamental design decisions that shaped the final solution. Furthermore, we provide information on the practical implementation of the delivery strategy within the EBRAINS RI, including a detailed breakdown of the deployment workflow, the automated testing procedures and release schedule introduced, as well as the progress made in making the EBRAINS software environment available on Fenix HPC systems.

The delivery strategy builds upon the foundational software delivery process for EBRAINS components initially outlined in Phase 1 (D5.3 [1], Section 4.2.3) and further refined during Phase 2 (D5.4 [2], Section 4.1) and Phase 3 (Section 4.1).

3.1.1 Requirements

The main requirements for all stakeholders (i.e. EBRAINS users, Developers/COs, EBRAINS DevOps/Maintainers, admins of Fenix ICEI HPC systems) stem from two primary objectives:

- Establishing a well-defined and scalable process for building, delivering, and activating EBRAINS software for direct use on the EBRAINS RI, and specifically on Jupyter Notebooks in the Collaboratory Lab environment.
- Making the same software environment also **available** in the **HPC systems** accessible to EBRAINS users through Fenix/ICEI, and **deployable** to other HPC target infrastructures in the future (e.g., PRACE, EuroHPC).

As explained in D5.4 [2], Section 3.1, the Collaboratory Lab environment did not initially have a defined methodology for adding or updating tools in the Docker image used to execute Jupyter notebooks, which posed significant challenges:

- any update of the software environment could break dependencies of all existing notebooks;

- library dependency conflicts were difficult to solve by builders of Docker images;
- tool installation/testing was often insufficiently documented to be performed by a central team.

At the same time, there was no defined process or central coordination of the deployment of EBRAINS simulation tools and software on FENIX ICEI HPC sites:

- a small number of tools were installed and made available on specific HPC systems by their development teams, but different sets of tools or versions were available on each system;
- there was no testing of the compatibility of the different EBRAINS tools available on the same system (e.g. conflicting dependencies);
- the complexity of certain EBRAINS tools posed significant challenges for users attempting to install them independently, especially in the case of multi-tool workflows.

For the purpose of addressing the constraints outlined above, the software delivery and deployment methodology was designed around the following principles:

- definition of a central modular software stack including all EBRAINS tools
  - version controlled, following a specific release schedule.
  - **well-tested**, with unit and integration tests, to ensure all tools function as expected and there are no **dependency conflicts** or incompatibilities between them.
  - **decoupled** from any specific system and installable on **different infrastructures.**
- **decentralised** software development: packaging and testing performed by tool developers.
- **small, easily maintained** Lab base container image, decoupled from EBRAINS software tools.
- **transparency** to the end-user: load and unload tools with ease

Based on the above, we can outline **the updated (since D5.4 [2]) main requirements** for the new approach in EBRAINS software delivery and deployment:

- requirements of **EBRAINS end users** (of the Lab and Fenix HPC systems):
  - out-of-the box solution (tested, seamless, no install/parameterisation needed)
  - integrated software environment, with all EBRAINS tools available and compatible
  - access to the same software in different systems (including the Lab environment and HPC systems) to ensure portability and reproducibility of experiments.
  - ensured future reproducibility of experiments/workflows (past versions of the software environment remain available)
- requirements of tool developers/component owners:
  - streamlined process for making their component/tool available.
  - software maintenance/update/testing in parallel
  - use of well-defined tests to ensure proper deployment of software and integration with other tools (interoperability)
  - recent versions of compilers (i.e. C, C++) and Python available
- requirements of EBRAINS platform DevOps/Maintainers:
  - distributed software development, testing and packaging (COs responsible)
  - centralized software deployment and distribution (Technical Coordination (TC) responsible, single integration point, central toolbox)
  - easier maintenance of Collaboratory Lab base container image (decoupled from software deployment)

- requirements of **administrators/maintainers of Fenix ICEI HPC systems** (or other external infrastructure):
  - well-defined, streamlined process for installing the EBRAINS releases on different systems/architectures, adaptable to each system's software management strategy.
  - sufficient, automated unit and integration tests to ensure the correct installation of software.
  - established central channels to facilitate ongoing communication with tool developers, to report and collaboratively solve issues that may arise concerning the installation of tools to new, more complex environments.

3.1.2 Implementation

With the purpose of meeting the above requirements and for the reasons explained in detail in D5.4 Section 3.1.2 [2], we opted for using Spack[9], an open-source, robust, and widely adopted solution designed for handling complex scientific software ecosystems, specialised in resolving conflicting dependencies by analysing requirements, creating isolated environments, and providing version control.

From an operational standpoint, our primary objective was to distribute the responsibilities of software development among Component Owners (COs), while concentrating the tasks of building, deploying, and distributing EBRAINS software under the oversight of TC. Simultaneously, we aimed to streamline the deployment process on external infrastructures, such as Fenix ICEI HPC systems, and potential future external providers (Trusted Research Environments, commercial cloud providers, etc.).

This approach established a single integration point, the TC-controlled EBRAINS Spack builds repository, that handles all test, build and deployment operations, and contains:

- the EBRAINS Spack repository: Spack package definitions of all EBRAINS tools, provided and maintained by the COs. Each Spack package encapsulates the build logic for different versions, compilers, options, platforms, and dependency combinations of an EBRAINS tool (Figure 15). External tools that are extensively used in the context of EBRAINS workflows can also be included in the EBRAINS Spack environment as dependencies of EBRAINS workflow "meta-packages".
- the EBRAINS Spack environment configuration: a well-defined list specifying the exact versions and configurations of the top-level EBRAINS tools that are part of the latest version of the EBRAINS software environment, as well as the site-specific configurations for its installation on the different sites (Figure 16).

[9] https://spack.io/

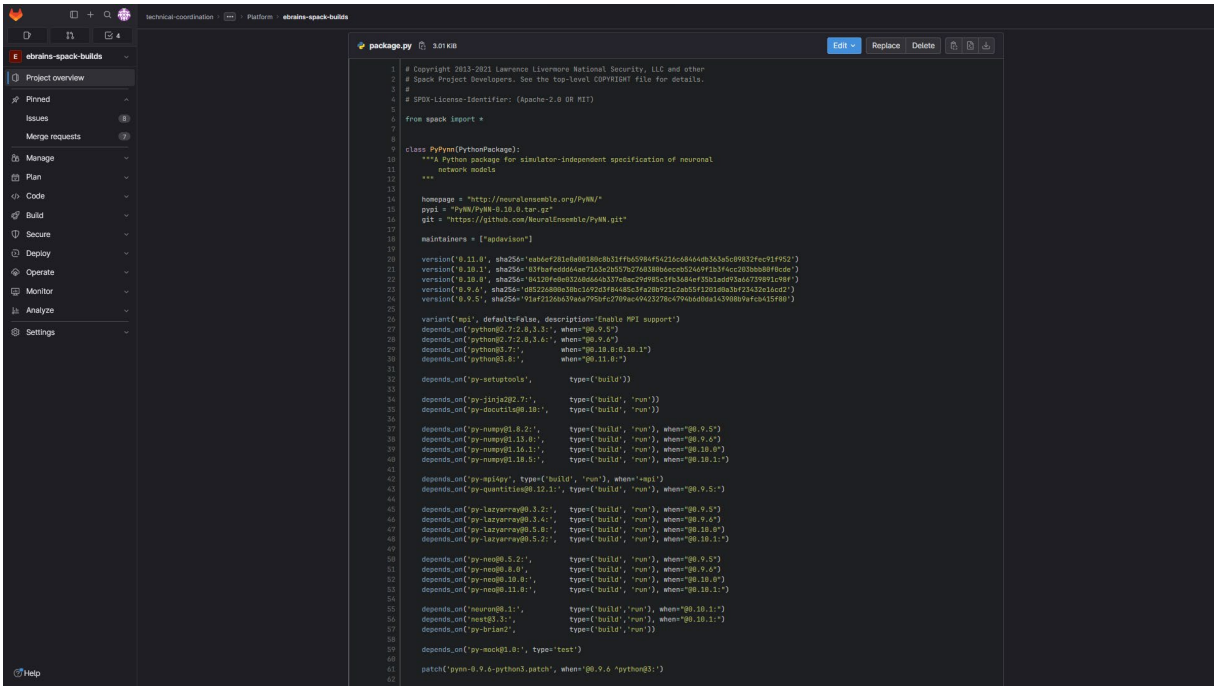

Figure 15: the EBRAINS Spack repository

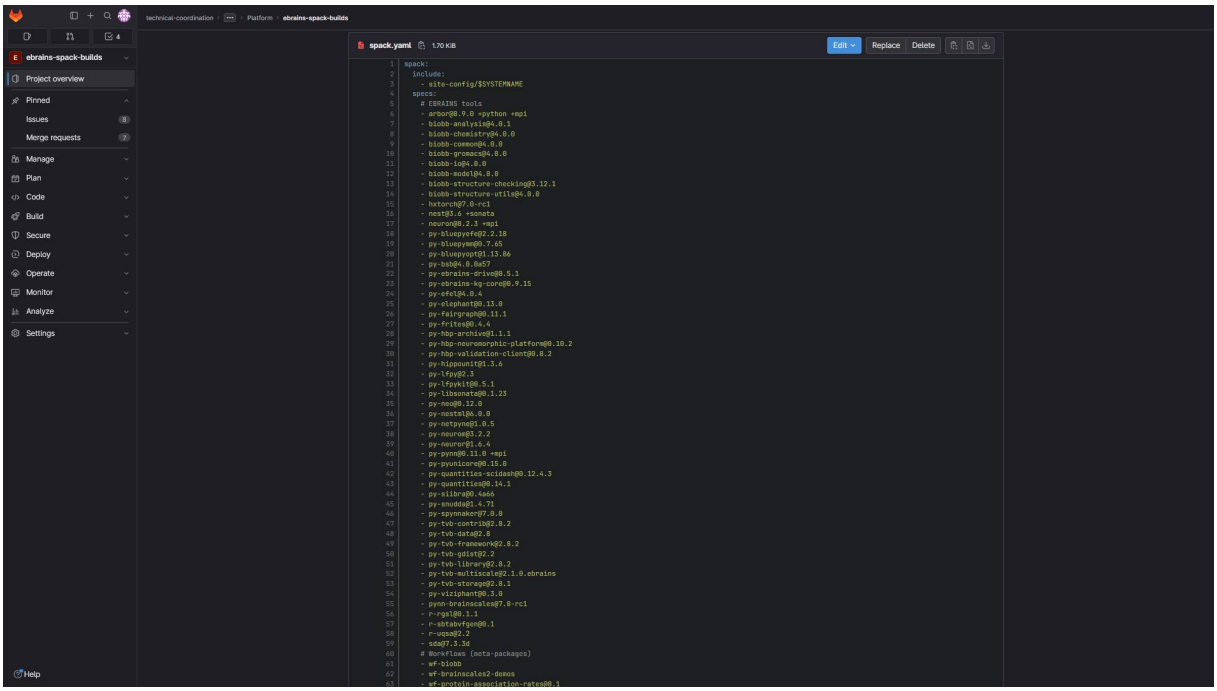

Figure 16: the EBRAINS Spack environment configuration

The implementation of the current solution can be abstracted in a simplified way from the diagram of Figure 17. In the next subsections, the different perspectives of this diagram will be explained, focusing on our main objectives regarding the EBRAINS common software environment:

- finalizing the process for building and deploying software within the EBRAINS Lab
- establishing a GitLab-based development environment with automated build testing for COs
- establishing a dedicated release schedule
- optimising the deployment process of the same software stack on HPC systems

Technical details and complete documentation for all relevant stakeholders (users, component owners etc.) can be found in Annex 8.4.1.

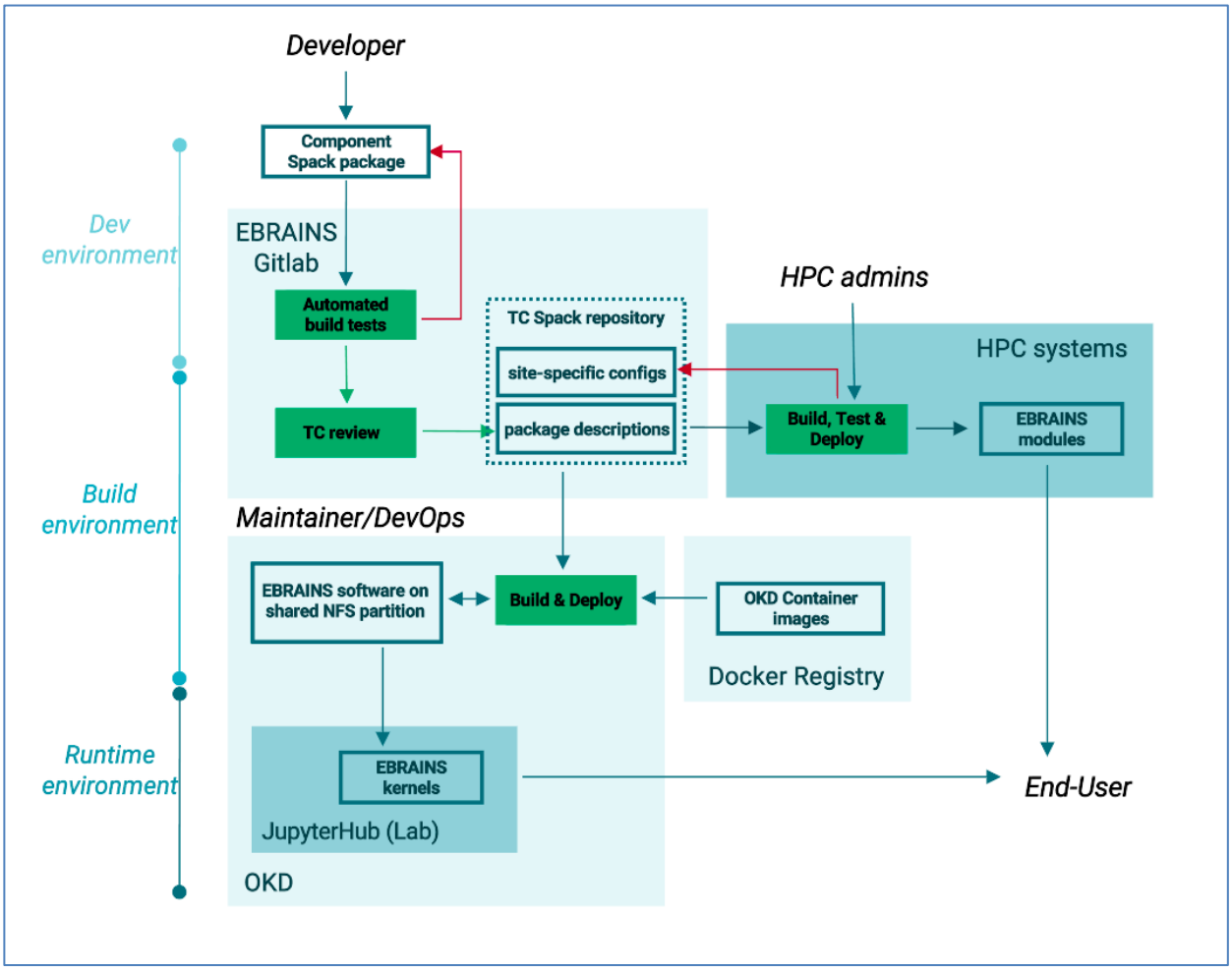

Figure 17: EBRAINS software delivery and deployment

3.1.2.1 Build and delivery process for the Lab environment

The process of building, delivering, and activating software for direct use in the Collaboratory Lab environment was initially formulated during Phase 2, as described in D5.4 [2], Section 3.1. This approach has been successfully implemented and operational within the EBRAINS Lab environment for the last two years, widely adopted by EBRAINS users and further optimized and extended during Phases 3 and 4.

Architecture-wise, the involved EBRAINS services are:

- **Gitlab** platform (and Gitlab CI) [build environment]: It holds the TC-controlled EBRAINS Spack builds repository where the Spack packages of the EBRAINS tools are stored, the site-specific configurations needed for the installation on different systems (see 3.1.2.4), as well as the CI configuration for all the test, build and deployment operations, that are coordinated via GitLab CI.
- **Docker registry** [build environment]: It holds the container images needed to run the build process on OKD, as well as the test build jobs on the GitLab CI runners.
- **OKD Container Platform** [build environment]: The build process for the Lab is executed through a Job on OKD. The build job is responsible for pulling the build image from the EBRAINS Docker Registry, transferring all necessary Spack packages, environment configuration files and scripts from the TC repository to the OKD pod that performs the build and installing (or updating) the Spack environment on an NFS partition that is mounted on users' Lab containers.
- **JupyterLab** (Notebooks environment) [runtime environment]: The running container mounts the shared NFS partition (read-only) and loads the Spack environment and all available tools. The

different releases of the EBRAINS software environment are available in different Python and R kernels on the Lab startup page (Figure 18).

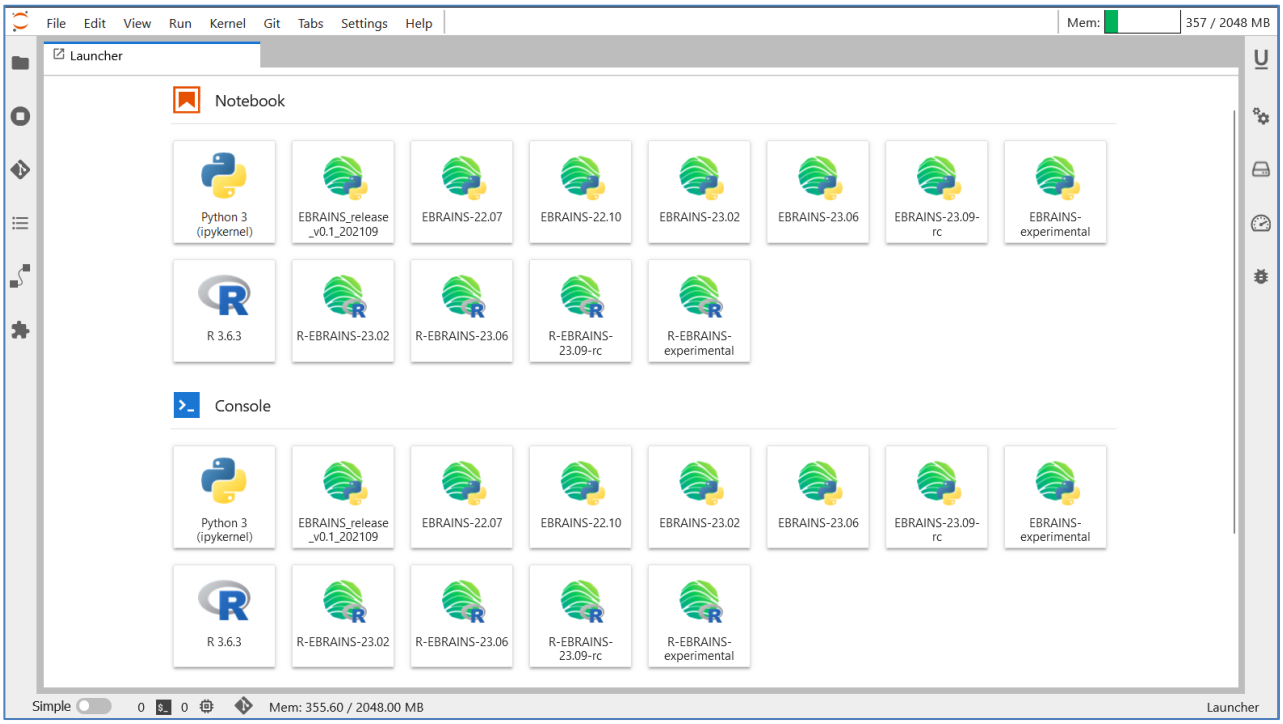

Figure 18: Collaboratory Lab Launcher page with all available EBRAINS kernels

Implementation-wise, and in direct correspondence with the EBRAINS CI/CD workflow established in D5.3 [1], we identified, designed, and implemented the integration steps for initiating the software release process:

- [Component's official code repository] **Development effort**
- [Gitlab] **Init step**
  - Creation of Spack package by COs/tool developers and Merge Request to include it to the official EBRAINS Spack repo.
- [Gitlab-OKD] Integration step
  - Automated build tests on top of the existing EBRAINS tools, triggered on MR (see 3.1.2.2)
  - (the proposed changes are merged after approval by TC)
  - Initiation and execution of OKD job that will perform the build on the Integration Lab
  - Propagation of the build logs from the OKD build job back to Gitlab.
- [Integration Lab] **Review step**
  - Automatic update of the test and experimental JupyterLab kernels on the Integration Lab
  - Validation by COs that everything is installed as expected.
- [Gitlab-OKD] **Delivery step**
  - Initiation and execution of OKD job that will perform the build on the production Lab.
  - Propagation of the build logs from the OKD build job back to Gitlab.
- [Manual step] Deployment step
  - Creation (or update) of the JupyterLab kernel
  - Update of the configuration of the Lab container to activate the new JupyterLab kernel, if necessary (in the case of a new release). This step does not require any downtime of the Lab, since it only affects the startup script of the containers.

3.1.2.2 Automated build tests and Continuous Integration

While, as described in the previous subsection, the delivery process itself needs to be performed centrally by the TC, it is also essential to provide COs with a development environment where they can work independently and in parallel, before finally submitting the Spack package for inclusion to the repository. Specifically, developers need to be able to update their packages, trigger test builds and have access to the build logs for debugging. However, testing each tool separately is not enough: to ensure the seamless integration of the various tools into one cohesive software ecosystem, it was necessary to implement a testing framework that guarantees their interoperability and compatibility.

To achieve this, an automated build testing process was introduced. When a component owner opens a merge request (MR) that adds or updates an EBRAINS tool, an automated build job is triggered. This build job attempts to install the new version on top of the existing EBRAINS software environment. This way, the integration of the new component with the existing toolset is tested, ensuring that it can function reliably within the common ecosystem.

Additionally, each component is required to define, as part of its Spack package description, a set of tests that sufficiently assess the software's functionality. Those tests are always executed at installation time for all the freshly installed tools and their dependents, to ensure that the changes introduced do not break the existing installation. This approach allows COs to identify and rectify integration issues early in the development cycle, while also providing a clear and standardised process for testing and validating changes.

To avoid long installation times and prevent unnecessary re-builds, a TC-owned persistent deployment of the EBRAINS software was installed on a read-only file system accessible from the GitLab runners. A shared NFS mount was set up, where only the TC controlled GitLab pipelines have read-write access. This always contains the latest deployment, so that developers can build and test their packages incrementally on top of it directly from GitLab, until all tests are successful.

In summary, the workflow described above effectively creates a complete Continuous Integration / Continuous Testing environment for the EBRAINS developer teams:

- developers can independently commit their changes to their own private forks and trigger build jobs without interaction with TC.
- the automated test jobs build any updated software on top of the latest version of the EBRAINS deployment, ensuring compatibility with the rest of the EBRAINS environment.
- Spack ensures that the necessary dependencies are resolved consistently (no conflicts)
- well-defined (by COs) tests run on every new build of each tool (or its dependencies) ensuring functionality.

This way, developers can confidently contribute to the EBRAINS software ecosystem, knowing that their changes will undergo sufficient testing and integration checks. The automated nature of the process reduces human error, accelerates development cycles, and enhances the overall quality of our software stack. Additionally, it encourages component owners to collaborate, share best practices, and maintain a high level of compatibility among tools, ultimately resulting in a more cohesive and efficient environment.

3.1.2.3 Release Cycle

The diagram in Figure 19 illustrates the various stages involved in developing and deploying the Spack-based versioned library for EBRAINS software. As detailed in the preceding section, the process of incorporating a new tool into the EBRAINS software environment starts with the submission of a Merge Request on the official EBRAINS Spack repository hosted on Gitlab by the Component Owner. These Merge Requests automatically trigger a predefined build test pipeline. Following the successful completion of the build tests and the approval of the changes by the Technical Coordination team, the new Spack packages are deployed to the Lab's integration environment, where COs are required to validate their performance to ensure they meet the expected behavior.

Figure 19: EBRAINS release cycle

New EBRAINS tools are deployed to production in two stages:

1) **Experimental Deployments (Weekly):** The EBRAINS TC team produces weekly experimental deployments of tools to the Lab environment. These deployments serve as a testing ground for new features but are not considered official EBRAINS releases. They are not subject to extensive verification or testing and are replaced by the subsequent Experimental Release. This implies that the tools available in an experimental deployment may be overwritten by the next one. Users are advised to use these experimental releases with caution and not depend on them for their production code, as they contain bleeding-edge delivery of new features in the tools.
2) **Official EBRAINS Releases (Quarterly):** On a quarterly basis, a new official EBRAINS Release is created. Each official release includes the tool binaries and a Python kernel that load all the new tool versions in the user's path within the Lab environment. These official releases are named following the format "EBRAINS_YY-MM" (e.g., "EBRAINS_22-07"), with the year (YY) and month (MM) denoting when the source code was finalized to create that particular release. Older releases are retained to ensure that users can reproduce their work with older notebooks in the future.

Preceding each new official production release of EBRAINS tools, a "development freeze" period is enacted, limiting the acceptance of Spack packages contributions. During this period, a "release candidate" is established, and EBRAINS tool developers and users are encouraged to test it and report any issues. Contributions that have successfully passed testing within two weeks of the release date are eligible for inclusion. Contributions received during these two weeks will be considered for the subsequent release. It is recommended that users avoid using release candidates in their production code.

Table 2 provides an overview of our quarterly official releases, showcasing the steady growth in the number of tools included in each release, alongside notable features that have been added to enhance our software ecosystem and delivery process over time, as covered in the preceding subsections. Information on which tools are part of the different releases is available in a dedicated Collab [6], the EBRAINS Spack builds repository as well as in Annex 8.4.2 of this report.

Table 2: Quarterly official releases

Release v0.1	EBRAINS-22.07	EBRAINS-22.10	EBRAINS-23.02	EBRAINS-23.06	EBRAINS-23.09
9 tools	21 tools	26 tools	36 tools	55 tools	59 tools
				deployed on Fenix HPC sites	
			available also in R kernels in EBRAINS Lab		
			automated deployment and testing process		
available in EBRAINS Lab (CSCS and JSC) in python kernels					
automated, centralized build & deployment process					

3.1.2.4 HPC deployments

The implementation of the approach described in the previous sections has proven highly successful within the EBRAINS Lab environment, enhancing efficiency and reproducibility in scientific workflows. As the advantages of having a central, version controlled, tested software environment were obvious, the need to expand this integrated software environment to other crucial infrastructures became clear. Specifically, making them available to all HPC systems accessible to EBRAINS users was identified as a crucial step in enabling researchers to harness the full potential of the integrated software ecosystem for their computational workflows.

In order to further enhance the usability of EBRAINS software on HPC resources and promote reproducibility in computational experiments and workflows, a robust mechanism was needed to make those EBRAINS software releases available on Fenix systems. To achieve this, a dedicated team collaborated closely with admins from the HPC sites (as reported in D6.4 [15]), with the goal of understanding the capabilities and constraints of Fenix systems and devising an optimised, streamlined deployment process tailored to the unique requirements of EBRAINS software.

As depicted in Figure 17, the TC-controlled EBRAINS Spack builds repository was once again used as a central integration point. In accordance with the requirements defined in Section 3.1.1, the installation process on HPC sites was based on the following principles:

- TC centrally defined the Spack-based EBRAINS software stack as described in the previous sections and handled the integration and testing process for the components.
- The different site software admins initiated the per-site installation of the EBRAINS software stack. The tests provided for each package (see 3.1.2.2) are executed to ensure the proper installation of the different tools.
- The requisite compilers, external packages, and specific configuration settings essential for the seamless and optimised operation were identified and documented in site-specific dedicated Spack yaml configuration files, that were then pushed to the repository, maintaining a single point of truth.
- Spack's environment definition syntax then allowed for these site-specific configuration files to then be effortlessly "included" into the defined EBRAINS Spack environment.
- A script that loads the installed Spack environment is exported and wrapped into a modulefile, so that the entire EBRAINS software environment can be loaded by loading an EBRAINS/YY.MM module (for each available EBRAINS release).
- For reporting and resolving build failures, effective communication channels were established within both the EBRAINS chat and the EBRAINS GitLab repository, accompanied by the creation of issue templates. This approach streamlined the process of reporting build failures and

facilitated their resolution through close collaboration with the software developers, thus ensuring that any issues that arose were promptly documented and contributing to the overall robustness and reliability of EBRAINS software across various computing environments.

The most up-to-date information on which releases are available on each site, as well as documentation on how to load them, are available in a dedicated Collab [6].

This approach ensures that EBRAINS software remains adaptable and accessible across diverse computing environments and allows users to take advantage of a versatile and consistent software ecosystem for their experiments and computational workflows, regardless of the computing environment they choose to leverage.

3.2 Standardised Workflows

3.2.1 Introduction

In D5.4 [2], we recognized the necessity of establishing clear definitions for terms associated with workflows, and specifically, we defined:

- *Standardised workflows:* series of non-interactive EBRAINS tools and services, linked together to create graphs, loops or branches for accomplishing scientific objectives related to brain neuroscience, defined as structured, common recipes, executed, and monitored via standardised workflow management systems and stored inside Knowledge Graph.
- *Non-interactive EBRAINS tools:* scientific data simulation or analysis command line tools bundled together with dependencies, binaries, and libraries, capable of resolving misconfigurations of software, as well as of reusing and versioning purposes.

As described in D5.4, establishing a standard and unified approach for defining workflows is of great importance for scientists and EBRAINS users in general. For that reason, we identified the Common Workflow Language (CWL) as the most suitable solution to meet our requirements. CWL is an open standard designed for defining analysis tools and workflows with specific inputs, parameters, and resources, executed portably and scalably across various computing environments (local, cloud, HPC).

In our context, Standardised workflows refer to a uniform and well-defined way of creating, sharing and executing CWL workflows. By adhering to CWL, developers can define workflows and tools in a consistent manner, making them easily accessible and executable by all EBRAINS users in different underlying infrastructures. Critical EBRAINS components adhered to and facilitated the wide introduction and support of Standardised workflows across EBRAINS. Namely, the Harbor EBRAINS Docker registry facilitates developers in creating projects and pushing docker images for each command line tool[10] used as workflow steps. Also, EBRAINS Knowledge Graph plays a crucial role as a centralized repository, serving as the single point of truth for users to discover, find and access CWL workflows and tools. EBRAINS KG offers the capability of associating CWL workflows, tools together with input and output data as well as with retrospective and prospective metadata (see Section 3.2.2).

The concept of Standardised workflows ensures that EBRAINS users and scientists can collaborate effectively, promote reproducibility, and streamline the workflows development creating an efficient environment for research and innovation. Thus, considering the importance of Standardised workflows to the EBRAINS community and in scientific communities in general, and taking into consideration the established methodologies applied in Scientific domains such as Life Sciences, the TC concentrated on **four action lines** for integrating Standardised workflows within the EBRAINS ecosystem. These actions have been briefly described in D5.4 (Section 3.2) and are provided in greater detail here for the reader's convenience.

[10] A command-line tool is defined as a non-interactive software designed to perform specific computational tasks.

1) A crucial first action line for scientists, component owners and developers is to logically divide their work into **building blocks** creating distinct non-interactive command line tools for each workflow step. As a subsequent step, they are required to bundle these non-interactive command line tools along with libraries, binaries and dependencies. Once this is completed, the next step involves creating the actual command line definition using CWL. To facilitate these steps and streamline the learning curve, TC has provided templates to assist tool owners in creating Docker containers as well as the non-interactive command line tool CWL definitions.

   For each non-interactive command-line tool, the following files must be generated following the template:

   - The tool's source code, forming the foundation of its functionality and operation.
   - A Dockerfile that contains instructions, including Base Image (Figure 20), Software Requirements (Figure 21), Tool installation and Docker command (Figure 22) for constructing the Docker image that encapsulates the tool's source code and its dependencies. This approach ensures seamless execution, detached from the specifics of the underlying infrastructure where the tool will run. Within the EBRAINS ecosystem, Docker images are pushed and stored in Harbor EBRAINS Docker registry.

```
# For more information please see:
# https://docs.docker.com/engine/reference/builder/
# https://docs.docker.com/develop/develop-images/dockerfile_best-practices/



# ------------------------------------------------------ BASE IMAGE ------------------------------------------------------

# Firstly, a base image needs to be defined. This image will be used as a base for the new, isolated system, where all
# the necessary software for the tool to run will be installed.

# For tools written in python, one of the following python Docker images should be used:
# - the defacto python image:
FROM python:3.8
# - the slim python image only contains the minimal packages needed to run python:
FROM python:3.8-slim
# - the alpine python image is based on Alpine Linux, and thus is much smaller:
FROM python:3.8-alpine
# Similarly, there are also images created for Java and most other languages.

# If the tool has no such requirement, a Linux distro image should be sufficient:
# - lightweight minimal Docker image based on Alpine Linux:
FROM alpine:3.14
# - Ubuntu Docker image, much larger than alpine:
FROM ubuntu:20.04

# For tools that need to use the GPUs of the host machine, several images are available in the nvidia/cuda dockerhub repository,
# providing an easy-to-use distribution for CUDA supported platforms and architectures. Three flavors of images are provided:
# - base, that includes the CUDA runtime
# - runtime, that also includes the CUDA math libraries, NCCL and cuDNN
# - devel, that also includes headers and development tools for building CUDA images
# for different OS, architectures and CUDA and CudNN versions. All available tags, and more information can be found
# here: https://hub.docker.com/r/nvidia/cuda
ARG UBUNTU_VERSION=18.04
ARG ARCH=
ARG CUDA=11.2
FROM nvidia/cuda${ARCH:+-$ARCH}:${CUDA}.1-base-ubuntu${UBUNTU_VERSION} as base

```

Figure 20: Template provided by TC team for creating Dockerfile; Base Image

```
# ------------------------------------------------------------ NOTES ------------------------------------------------------------

# It is important to keep in mind that the docker build command examines the lines in the Dockerfile one by one, and
# creates an intermediate image for each of those steps, by adding all the new files from that step as another layer on top
# of the previous image. Docker will use these intermediate images as image cache, and future builds will be much faster for
# the Dockerfile steps that are not modified. Consequently, in order to reduce image build time, it is important to
# put in the beginning of the Dockerfile the lines that are least likely to change while the tool undergoes further development.
# That's why essential tools needed to build the application must be installed first (e.g. curl, git, virtualenv etc), then
# tool-specific dependencies, and finally the application itself.

# It is also useful to try minimizing the number of steps in the Dockerfile, by combining several steps, to reduce the number
# of intermediate images produced.



# ------------------------------------------------------ SOFTWARE REQUIREMENTS ------------------------------------------------------

# In this section, all the software dependencies necessary to run the tool must be installed in the docker image.
# Widely-used package managers are supported, such as pip, apk, apt etc.
# All commands should start with RUN. It is recommended to use 'no-cache' flags when possible to reduce image size.

# examples:

# install using apt:
# especially for apt-get commands, it is recommended to structure them as follows
# (the explaination can be found at https://docs.docker.com/develop/develop-images/dockerfile_best-practices/)
RUN apt-get update && apt-get install -y \
    package-bar \
    package-baz \
    package-foo  \
    && rm -rf /var/lib/apt/lists/*

# install using apk
RUN apk add --no-cache package-bar package-foo

# install from source
RUN mkdir -p /opt/.foo \
    && wget https://foo.com/download/foo.tar.gz -O foo.tar.gz \
    && tar -xf foo.tar.gz -C /opt/.foo \
    && rm foo.tar.gz

# install requirements using pip
COPY requirements.txt .
RUN pip install --no-cache-dir --upgrade pip && pip install --no-cache-dir -r requirements.txt
```

Figure 21: Template provided by TC team for creating Dockerfile; Software Requirements

```
# ------------------------------------------------------ TOOL INSTALLATION ------------------------------------------------------

# The next step is to install the actual tool in the new, isolated environment. That means that the source code of the tool
# must be made available inside the docker image.
COPY . .

# The tool installation is a different process for each tool, so no specific instructions can be provided.
# If a setup.py file is provided, for example, it can be as simple as
RUN pip install .

# In case the tool is not packaged so that it can be installed, for example if it is a single script, it should be made
# executable:
RUN chmod +x /path/to/tool/preprocess.py
# - and added to the docker image path, so that it can be executed anywhere
ENV PATH="/path/to/tool/:$PATH"



# ------------------------------------------------------ DOCKER COMMAND ------------------------------------------------------

# No changes should be made in this step.
CMD [ "/bin/bash" ]
```

Figure 22: Template provided by TC team for creating Dockerfile; Tool installation and Docker command

- A CWL tool description template, depicted in Figure 23, Figure 24 and Figure 25, describes the semantics, syntax and execution details of the tool. These CWL descriptions will later be utilized as workflow steps when creating CWL workflows.

```yaml
# This file contains the command line tool CWL description.
# For more information please see:
# https://www.commonwl.org/v1.2/CommandLineTool.html

# ------------------------------------------------- COMMAND LINE TOOL INFO -------------------------------------------------

# CWL version can be v1.0, v1.1 or v1.2.
# v1.0 is recommended, since it is the most compatible with all workflow engines.
cwlVersion: v1.0

# A CWL description file can be either a Workflow or a CommandLineTool.
# for command line tools:
class: CommandLineTool


# ------------------------------------------------------ BASE COMMAND ------------------------------------------------------

# This is the base of the command line tool, meaning the program to be executed, without any inputs or parameters.

baseCommand: preprocess.py

# If the command line tool has any mandatory command line arguments, the baseCommand should be an array, whose first element
# is the command to execute, and subsequent elements are the mandatory command line arguments, e.g.:
baseCommand: ['preprocess.py', '--method1']


# ------------------------------------------------- HINTS/REQUIREMENTS -------------------------------------------------

# This section is used to introduce requirements and hints concerning the execution environment of the tool. It is
# possible to define either soft or hard runtime requirements. A requirement modifies the semantics or runtime
# environment of a process. If an implementation cannot satisfy all requirements, or a requirement is listed which is
# not recognized by the implementation, it is a fatal error and the implementation must not attempt to run the process,
# unless overridden at user option. A hint is similar to a requirement; however, it's not an error if an implementation
# cannot satisfy all hints. The implementation may report a warning if a hint cannot be satisfied.

hints:    # (or requirements)
  # The DockerRequirement tag is used to define the Docker image that is necessary for running the specific tool. The
  # image will be pulled from the URL specified by the DockerPull tag. For EBRAINS tools, images are stored at
  # docker-registry.ebrains.eu
  DockerRequirement:
    dockerPull: docker-registry.ebrains.eu/project/tool_image:latest
  # The ResourceRequirement tag is used to specify basic hardware resource requirements. It can be used for any of the
  # resouces listed at https://www.commonwl.org/v1.1/CommandLineTool.html#ResourceRequirement, as long as the system
  # supports them
  ResourceRequirement:
    ramMin: 2048
    outdirMin: 4096
  # ...
```

Figure 23: Template for creating CWL tool definitions; Information, Base Commands, Hints/Requirements

```yaml
# ------------------------------------------------------- TOOL INPUTS -------------------------------------------------------

# All the inputs with their types and input binding are defined here. InputBinding field describes how to turn the input
# parameters of a program into command line arguments (prefix defines the prefix used and position the position in the command.
# (when the parameter type ends with a question mark "?" it indicates that the parameter is optional)

# for example, the inputs shown below are for a command line tool with the following syntax:
# preprocess.py doi path/to/data/ --keep_logs --n_steps 4 --channels [1,2,3,4,5]
inputs:

# These inputs can be:
# strings, e.g.:
  doi:
    type: string
    inputBinding:
      position: 1
# file or directory objects, e.g.:
  data:
    type: Directory   # (or File)
    inputBinding:
      position: 2
# boolean binary variables (true/false), e.g.:
  keep_log:
    type: boolean?
    inputBinding:
      position: 3
      prefix: --keep_logs
# numbers (int, long, float, double), e.g.:
  n_steps:
    type: int?        # (or long, float, double)
    inputBinding:
      position: 4
      prefix: --n_steps
# arrays (of any of the above), e.g.:
  channels:
    type: int[]?
    inputBinding:
      position: 5
      prefix: --channels
  # ...
```

Figure 24: Template provided by TC team for creating CWL tool definitions; Tool inputs

```
92
93  # ------------------------------------------------------- TOOL OUTPUTS -----------------------------------------------------
94
95  # All the outputs with their types and output binding are defined here. OutputBinding field describes how to generate this
96  # output object based on the files produced by a CommandLineTool. The glob field consists of the name of a file in the output
97  # directory. If you don't know name of the file in advance, you can use a wildcard pattern like glob: '*.txt'.
98
99  outputs:
100   job:
101     type: File
102     outputBinding:
103       glob: out.json
104
105 # To capture stdout (same for stderr) the stdout field has to be added, as follows:
106 # stdout: output.txt
107 # outputs:
108 #   example_out:
109 #     type: stdout
110 # The above redirects the stdout in a file named output.txt and then considers that the output of the tool.
111
```

Figure 25: Template provided by TC team for creating CWL tool definitions; Tool output

- Optionally, one or more .yml files can be included, containing specific inputs used for different executions of the CWL command line tool.

2) Our next action line is dedicated to the **formulation of CWL workflows**, which are framed as structured recipes, defined in a common and standard way. To facilitate a smoother transition for developers embarking on the adoption of Standardised workflows approach, TC team has thoughtfully provided a workflow template that offers a straightforward file structure for creating CWL workflows without requiring an in-depth comprehension of CWL descriptions. The template structure includes the following:

   - A directory that contains the CWL descriptions of all the command line tools that will be used as workflow steps (one .cwl file per workflow step).
   - A CWL workflow template description depicted in Figure 26 and Figure 27,documents all aspects of the workflow, including all its steps, inputs and outputs files (serving as the "workflow recipe").

```
1   # This file contains the workflow "recipe" (workflow description).
2   # For more information please see:
3   # https://www.commonwl.org/v1.2/Workflow.html
4
5   # ------------------------------------------------------- WORKFLOW INFO ----------------------------------------------------
6
7   # CWL version can be v1.0, v1.1 or v1.2.
8   # v1.0 is recommended, since it is the most compatible with all workflow engines.
9   cwlVersion: v1.0
10
11  # A CWL description file can be either a Workflow or a CommandLineTool.
12  # for workflows:
13  class: Workflow
14
15
16  # ------------------------------------------------------- WORKFLOW INPUTS ---------------------------------------------------
17
18  # All the inputs with their types are defined here.
19  # (when the parameter type ends with a question mark "?" it indicates that the parameter is optional)
20  inputs:
21
22  # These inputs can be:
23  # strings, e.g.:
24    doi: string
25    output_filename: string
26  # boolean binary variables (true/false), e.g.:
27    keep_log: boolean?
28  # numbers, e.g.:
29    int_input: int          # (32-bit signed)
30    long_input: long        # (64-bit signed)
31    float_input: float      # (single precision)
32    double_input: double    # (double precision)
33  # file objects, e.g.:
34    input_file: File
35  # directory objects, e.g.:
36    input_dir: Directory
37  # arrays (of any of the above), e.g.:
38    inputs: string[]?
39  # ...
40
41
42  # ------------------------------------------------------- WORKFLOW OUTPUTS --------------------------------------------------
43
44  # All the outputs with their types are defined here.
45  # Apart from the type of each output (which can be any of the types mentioned above),
46  # the outputSource must be set to define whick step produces the output.
47  outputs:
48    output_file:
49      type: File
50      outputSource: visualization/output_file
51
```

Figure 26: Template provided by TC team for creating CWL workflow definitions; Info, Inputs, Outputs

```
# ------------------------------------------------------ WORKFLOW STEPS ------------------------------------------------------

# Finally, all the workflow steps are defined here.
# For each step, the user must define:
# - the path to the .cwl definition of the CommandLineTool
# - the way that workflow inputs (or intermediate results) are mapped to the tool inputs defined in the CWL tool description.
#   (on the left there is the tool input name, and on the right the workflow input name)
# - the step output, that can then be used as input of other steps or workflow output
# The relations between the various steps that are defined here are used to create the DAG (Directed Acyclic Graph) that
# determines the order of step extecution.
steps:
  preprocessing:
    run: steps/preprocessing_tool.cwl
    in:
      doi: doi
      data: input_dir
      keep_log: keep_log
    out: [preprocessing_out]  # note that this is used as input of the next step

  analysis:
    run: steps/analysis_tool.cwl
    in:
      data: preprocessing/preprocessing_out
      keep_log_flag: keep_log
    out: [analysis_out]       # note that this is used as input of the next step

  visualization:
    run: steps/visualization_tool.cwl
    in:
      data: analysis/analysis_out
      figure_name: output_filename
    out: [output_file]        # this is the workflow output defined above

  # ...
```

Figure 27: Template provided by TC team for creating CWL tool definitions; Workflow steps

- One or more .yml files containing specific inputs used for different executions of the workflow. These files serve as a guide for executing the recipe with pre-defined inputs.

After the completion of defining the CWL workflow and tools, the next action line involves adding the CWL workflow and tool definitions to the **EBRAINS Knowledge Graph (KG).** KG plays a crucial and important role in enhancing shareability, findability and completeness of standardised workflows. Each CWL workflow and tool description are assigned with a unique Digital Object Identifier (DOI) for future reference and seamless association with scientist's research efforts. In parallel, KG also assumes a central role in orchestrating the handling of outputs, inputs as well as intermediate results generated from executed workflows, each identified by specific DOIs. By adopting this approach, the neuro – scientific fields are advancing towards a FAIR and innovative era. The whole process includes manual curation to ensure important fields are filled out, covering critical information such as a description of each step, of the workflow, the input parameters as well as input and output data is carefully taken care of, is not missing and is properly integrated into EBRAINS KG. Users who execute CWL workflows may choose to associate retrospective metadata, collected during runtime, with the prospective metadata of Standardised workflows in EBRAINS KG. This linking process associates all retrospective and prospective metadata with specific digital objects, including CWL workflow and tool recipes as well as input and output data.

Finally, we identified that a dedicated and integrated interface for users to execute, view and monitor their Standardised workflows was of high importance. This was manifested in a dedicated section of the EBRAINS Dashboard (the **Workflows Dashboard**, see Section 3.5), in which numerous features are available, from execution, monitoring logs and editing workflows, to tracking the status for each workflow step as well as the workflow itself and finally fetching any outputs that are produced.

3.2.2 Provenance

As mentioned in D5.4 Section 3.6.1, [2], a provenance API was under development incorporating the openMinds schema inside EBRAINS KG. After careful consideration, it was decided that provenance in the context of EBRAINS would encompass four sub-components, three of which are already in development or pre-production stages, with one planned for the future.

- First, the openMINDS computational module metadata schemas expand upon the openMINDS core by providing schema-templates for adding metadata related to simulation, data analysis, and visualization to the EBRAINS Knowledge Graph (KG).

- The second sub-component, the EBRAINS Provenance API, is accessible to all EBRAINS users at https://prov.brainsimulation.eu/docs#/. This API offers various endpoints, including their parameters and request/response examples, to support a wide range of functions. These functions encompass managing workflow recipes, workflow executions, simulations, data analysis, visualization, optimization and data transfer. Developers can perform operations such as GET, POST and DELETE through this API to interact with these components effectively.
- The third sub-component, the provenance visualization app, is under development to help users visualize provenance information effectively.
- Finally, there are plans for the fourth sub-component, a Python library for capturing prospective and retrospective provenance information, transforming provenance records from other tools (such as cwl runner and Datalad metadata schema), into the openMINDS format, and sending this captured metadata to the KG via the Prov API.

In the context of Standardised workflows, provenance metadata serves both retrospective and prospective purposes. Retrospective metadata includes information generated during the execution of a CWL workflow, such as logs, input parameters, and execution history. This retrospective data allows users to review past workflow executions, identify errors, and refine their workflows for improved efficiency and accuracy. On the other hand, prospective metadata involves details provided before a workflow's execution, including input parameters, expected outputs, and workflow descriptions. Prospective metadata aids in planning and documenting the workflow's intended purpose, making it easier for users to understand and reproduce the workflow in the future. Together, these metadata components enhance the traceability, reproducibility, and overall management of CWL workflows.

In the context of EBRAINS, developers are provided with essential guidelines to ensure the inclusion of prospective metadata in their Common Workflow Language (CWL) workflows. Annotating CWL files is a crucial step, simplifying the subsequent manual curation process performed by workflow curators which usually takes a couple of days. Moreover, best practices dictate storing CWL files for all workflow steps, as well as workflow recipes, within a version control repository. Developers are strongly encouraged to create either commit IDs or tags to precisely identify the versions of tools, workflow steps, or recipes used in their workflows. Additionally, adding labels to CWL files enhances the comprehensibility of provenance records. To maintain consistency, it is advisable to use a single keyword as a label for each CWL file. Leveraging the openMinds schema adopted by EBRAINS KG, keywords are available to specify the type of computation performed in each workflow step, ranging from visualization and data analysis to simulation, optimization, and more. This meticulous metadata management not only ensures traceability but also enables scientists to associate all aspects related to their workflows with their research, publications, and scientific contributions. This metadata not only facilitates sharing workflows with others while adhering to FAIR principles but also integrates seamlessly with EBRAINS Knowledge Graph (KG).

Integration with EBRAINS KG is already in place, where workflows and command-line tools defined via CWL, which are used as workflow steps, can be found within EBRAINS KG. Unique Identifiers (UUIDs) for CWL tools and workflows are available, facilitating the interconnection of various digital objects. Currently, two out of three Showcases that have adopted the Standardised approach have embraced best practices and guidelines for capturing prospective metadata, defining workflow recipes to pass the curation process, and better integrate with EBRAINS KG. Showcase 3[11] and Showcase 4[12] are publicly available under EBRAINS KG. Different versions of the workflows (v1.0 and v1.1) are maintained along with annotations, labels and keywords added during definition phase. All users, both inside and outside of EBRAINS can review the visual representation of the workflows, together with their input and output parameters.

Exemplary Use cases

The concept, further development, and operationalization of Standardized workflows facilities in the EBRAINS RI has been pursued by three EBRAINS Showcases (*Brain Complexity and Consciousness*

[11] https://search.kg.ebrains.eu/instances/bd71c7c0-c2bb-454a-8d18-12ef96f45cdd
[12] https://search.kg.ebrains.eu/instances/86efc59a-b4d1-443d-a470-dd27ddc4465b

- ***Showcase 3***, *Object Perception and Memory* - ***Showcase 4*** and *Dexterous manipulation* - ***Showcase 5***) and one EBRAINS component (CobraWap). This resulted in four EBRAINS exemplary use cases which (i) explore and demonstrate the full breadth and depth of Standardized Workflows facilities in the EBRAINS RI, and (ii) serve as blueprints for the wide adoption of CWL workflows by end users. An additional external use case originating from BioExcel, was also provided to demonstrate the de facto portability of CWL workflows, as well as the available reuse pathways between neighbouring scientific communities. Overall, these are available as 'Feature Workflows' within the EBRAINS Workflows Dashboard, enabling EBRAINS users to seamlessly retrieve the workflow in their own personal space, alter the workflow recipe by adding or removing workflow steps for their research or execute it in its original form. EBRAINS users are encouraged to execute the workflow with different input parameters as they best fit their research requirements and specific needs.

Detailed information regarding these five exemplary use cases is presented in the sections that follow, as well as in the corresponding Annex.

Brain Complexity and Consciousness – Showcase 3

A summary of the scientific part of Showcase 3 is presented next, with additional information available in [9].

A brain-based quantification of the levels of consciousness is of the utmost importance because, each year, intensive care medicine is called upon to treat millions of patients whose level of consciousness is difficult to assess due to severe brain injuries and disconnections. Detecting the fundamental mechanisms of consciousness is crucial, not only for better diagnosis, but also to guide recovery in an optimal manner. It is also critical to provide tools - such as eye tracking or brain computer interfaces - to provide communication channels for patients who have recovered consciousness but remain disconnected (e.g. locked-in patients). Another relevant requirement comes from the field of anesthesiology - pharmacologically induced alterations of consciousness – which is used in millions of patients every year. The effectiveness of this approach is limited by a lack of systematic understanding of the underlying circuit mechanisms and a lack of reliable brain-based measures of anesthesia depth. Therefore, deeper understanding of consciousness also paves the way to engineering novel methods of tracking the results of pharmacological interventions, as well as engineering next-generation, non-pharmaceutical, direct methods for inducing states of non-responsiveness, with potentially fewer side effects and dangers.

In Figure 28, we observe a demonstration of the implementation of AdEx mean-field models into The Virtual Brain (TVB), which leads to a framework where one can evaluate the effect of "microscopic" parameters, such as spike-frequency adaptation, on the "macroscopic" behavior at the level of the whole brain, such as the emergence of asynchronous dynamics and synchronized slow waves [10].

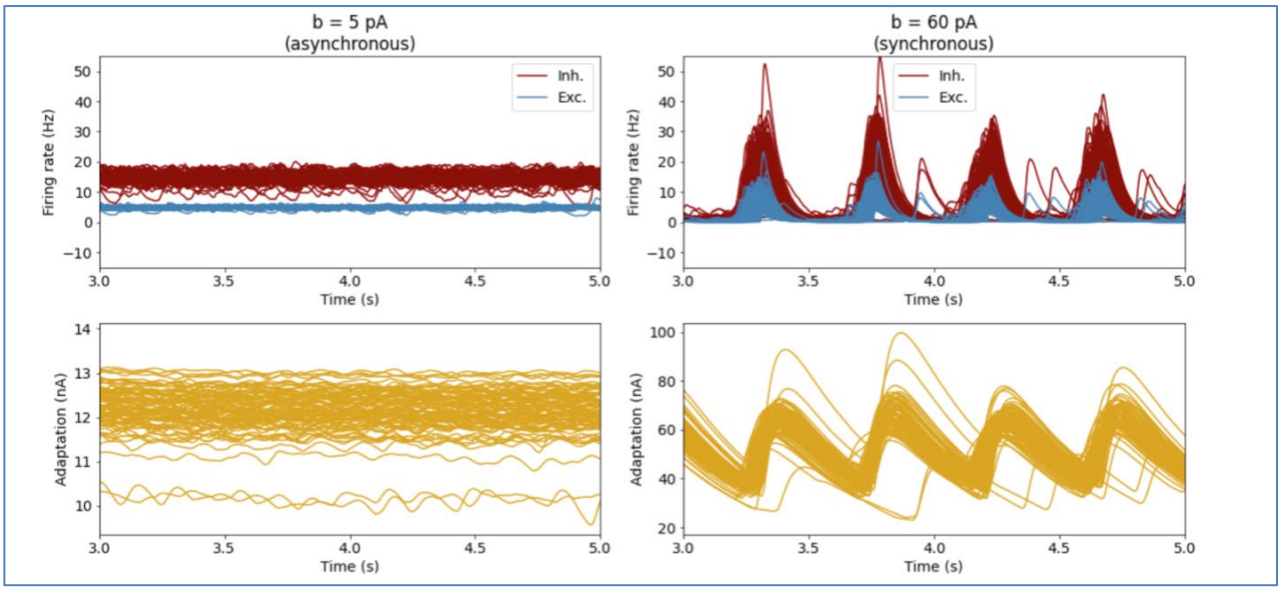

Figure 28: Whole-brain-scale simulation of connected AdEx mean-field models.

Showcase 3 marked the initial adoption of the Standardised Workflows approach with EBRAINS. The first pivotal phase involved the division of the previously existing pipeline, originally accessible

through a Jupyter notebook, into discrete building blocks. Towards this, the authors created distinct non-interactive command line tools for each of these building blocks, ensuring they were encapsulated with all required dependencies, libraries and binaries through the Docker containerization method and pushed into the Harbor EBRAINS Docker registry. The next step involved crafting the CWL definitions for each of the tools that will be utilized as workflow steps during the definition of the CWL workflow.

As far as EBRAINS Knowledge Graph is concerned, it served a dual purpose in the context of Showcase 3. First, it was employed for data retrieval, primarily to source input data used as inputs for the CWL workflow. Second, it served as a repository for prospective metadata13 for Showcase 3. This encompassed various elements, such as a preview and a summary of the CWL workflow, a preview and a summary for all input and output data types, with ability for interacting with different workflow steps to gain detailed information.

Regarding the Showcase 3 CWL workflow[14], it consists of three steps. In the first step (simulation), the Mean-Field AdEx model is executed based on the input parameters given. The second step (visualization) gathers the output of the previous step and displays two plots. The first simulates traces for all nodes under given conditions and the second contains traces of the simulation for all nodes under given conditions. At the third step, the output is pushed into S3 buckets to be stored for future reference.

The default input parameters can be summarized in Table 3.

Table 3: Showcase3 input & output parameters

ID	Type	Document	Data Type
• *bvals:*	Input file	adaptation values, the ones that make you obtain different activity regimes (b=0 for asynchronous and b=60 for synchronous dynamics)	*Int[]*
• *simname*	Input file	label for each of the bvals to be added in the figure	*String[]*
• *output_dir_name:*	Input file	name of the output directory where to store the results	*string*
• *run_sim:*	Input file	total time of the simulation (in ms)	*int*
• *cut_transient:*	Input file	transient time (in ms) to cut at the beginning of the simulation	*int*
• *token:*	Input file	user's token for the data proxy (obtained via clb_oauth.get_token())	*string*
• *bucket_id:*	Input file	a bucket id with read and write access to	*string*
• *folder_root and synch_root:*	Input file	auxiliar folders, they should not be changed and should be left as 'result' and 'synch' respectively	*string*
• *fig_PSD.png*	Output file	final plot containing the PSD of the simulated traces for all nodes under given conditions	*File*
• *Fig_traces.png*	Output file	final plot containing the traces of the simulation for all nodes under given conditions	*File*

3.2.2.1 Object Perception and Memory – Showcase 4

A summary of the scientific part of Showcase 4 is presented next, with additional information available in [10].

Predictive coding (PC) is an influential theory in neuroscience, which suggests the existence of a cortical architecture that is constantly generating and updating predictive representations of sensory inputs. Owing to its hierarchical and generative nature, PC has inspired many computational models of perception in literature. However, the biological plausibility of existing models has not been sufficiently explored due to their use of artificial neural network features such as a non-linear,

[13] https://search.kg.ebrains.eu/instances/bd71c7c0-c2bb-454a-8d18-12ef96f45cdd
[14] https://gitlab.ebrains.eu/workflows/sc3_cwl

continuous, and clock-driven function approximator as basic unit of computation. Therefore, Showcase 4 develops a spiking neural network for predictive coding (SNN-PC), in which neurons communicate using event-driven and asynchronous spikes. While adopting the hierarchical structure and Hebbian learning algorithms from previous PC neural network models, SNN-PC introduces two novel features. The first one involves a fast feedforward sweep from the input to higher areas, which generates a spatially reduced and abstract representation of input (i.e., a neural code for the gist of a scene) and provides a neurobiological alternative to an arbitrary choice of priors. The second one involves a separation of positive and negative error-computing neurons, which counters the biological implausibility of a bi-directional error neuron with a very high basal firing rate. After training with the MNIST handwritten digit dataset, SNN-PC developed hierarchical internal representations and was able to reconstruct samples it had not seen during training. SNN-PC suggests biologically plausible mechanisms by which the brain may perform perceptual inference and learning in an unsupervised manner. In addition, it may be used in neuromorphic applications that can utilize its energy-efficient, event-driven, local learning, and parallel information processing nature.

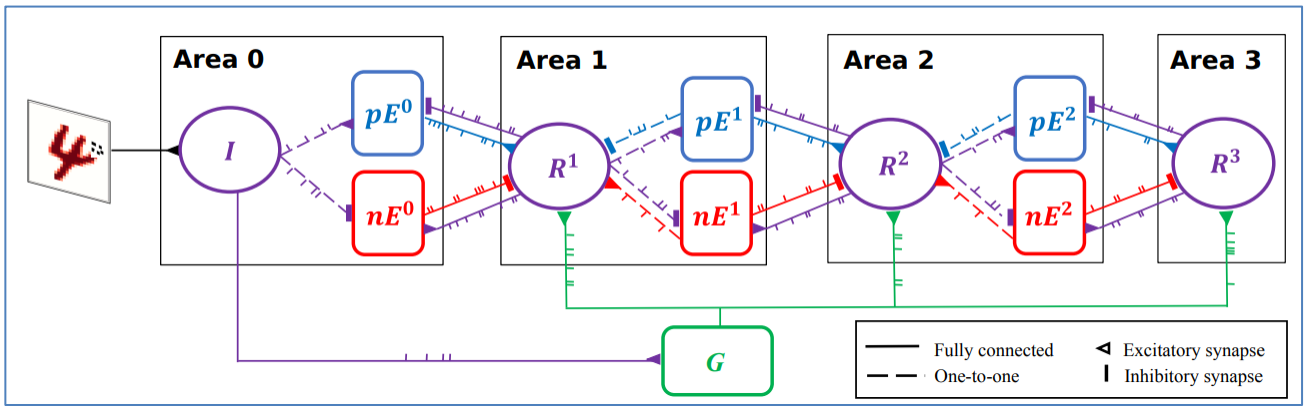

Figure 29: Spiking neural network for predictive coding (SNN-PC) [10]

The authors of Showcase 4 successfully transitioned from their initial approach to Standardised workflows. The first step involved the division of the previously existing pipeline, originally accessible through a Jupyter notebook, into discrete building blocks. In this regard, the authors created distinct non-interactive command line tools for each of these building blocks, ensuring they were encapsulated with all required dependencies, libraries and binaries through the Docker containerization method and pushed into the Harbor EBRAINS Docker registry. As a next step, the authors crafted the CWL definitions for each of the tools that will be utilized as workflow steps during the definition of the CWL workflow.

EBRAINS Knowledge Graph served a dual purpose in the context of Showcase 4. First, EBRAINS KG was used for data retrieval, primarily for source input data utilized as inputs for the Showcase 4 CWL workflow. Second, it served as a repository for prospective metadata [15] for Showcase4. This encompassed various elements, including a preview and a summary of the CWL workflow, as well as a preview and a summary for all input and output data types. Users can interact with different workflow steps and input and output data types to access detailed information.

The Showcase 4 CWL workflow[16] consists of two steps. In the first step (simulation), the workflow demonstrates the performance of a 3-layer spiking neural network for predictive coding. A user can select a MNIST sample which the network has never encountered before. However, the network can still infer based on other MNIST samples on which it has been trained. The second (visualization) displays a figure of the chosen sample. Figure 29 displays mean synaptic current points, each corresponding to mean firing rates.

The default input parameters can be summarized in Table 4.

[15] https://search.kg.ebrains.eu/instances/86efc59a-b4d1-443d-a470-dd27ddc4465b
[16] https://gitlab.ebrains.eu/workflows/sc4_cwl

Table 4: Showcase4 input & output parameters

ID	Type	Document	Data Type
• *folder_root:*	Input file	results folder	*string*
• *plot_dir:*	Input file	directory where outputs will be stored	*string*
• *digit:*	Input file	digit of the MNIST dataset	*int*
• *sample:*	Input file	sample of the digit of the MNIST dataset	*int*
• *curr_t:*	Input file	inference progress at time curr_t	*int*
• *fig_traces_digit_{digit }sample{sample}.png*	Output file	final plot containing the inference progress of digit, sample and curr_t	*File*

3.2.2.2 Dexterous manipulation - how the brain coordinates hand movement - Showcase 5

A summary of the scientific part of Showcase 5 is presented next, with additional information available in [11].

Goal-driven deep learning is being used more frequently in computational neuroscience to complement traditional modeling methods. Deep neural networks excel at autonomously learning complex connectivity patterns needed for real-world tasks, reducing the reliance on manually engineered connections. This approach generates hypotheses about how the brain processes information. While goal-driven modeling is common in perception neuroscience, it's challenging for sensorimotor control due to the complexity of training models involving closed sensation-action loops. To address this, AngoraPy, a modeling library, has been introduced. It equips researchers with tools to train intricate recurrent convolutional neural networks for modeling sensorimotor systems. This Showcase employs deep learning to train a recurrent convolution neural network (RCNN) for controlling an anthropomorphic robotic hand during complex in-hand object manipulation tasks. These tasks demand substantial computational power, which is provided through HPC resources within the EBRAINS infrastructure.

The training data is acquired through the agent's experiences in a simulated environment. Following the successful training of the RCNN, its representation transformations is analyzed using standard tools from neuroscience, such as representational similarity analysis. Additionally, deep learning techniques are applied, including feature visualization, and explore concepts from dynamical systems theory, such as stability analysis, to gain insights.

Showcase 5 utilizes an asynchronous algorithm that enables multiple workers to independently generate experiences. Parallel data gathering and optimization are facilitated through MPI (Message Passing Interface). Agents automatically distribute their workers evenly across available CPU cores, while optimization tasks are distributed across all available GPUs (Graphics Processing Units). If no GPUs are available, optimization is shared among all available CPUs. To utilize MPI, an active MPI implementation like OpenMPI is necessary. Furthermore, to leverage the HPC resources of the EBRAINS infrastructure, distributed training on SLURM-based HPC systems is supported.

Despite the maturity of Showcase 5, the process of transitioning certain parts of the work, which heavily relied on MPI and GPUs, into a CWL-based workflow posed some challenges. At the same time, the CWL community was actively exploring ways to incorporate these features into both the definition and execution of CWL workflows. In more detail:

- CWL Specifications: In its earlier versions, CWL specifications did not explicitly define standard ways to support GPU and MPI for parallelism. The specifications were first focused on defining workflow steps with inputs and outputs. This made GPU-related configurations and MPI-based parallelism difficult to express as CWL workflow.
- Support for GPU and MPI: The CWL specification itself is not an execution engine; it relies on execution engines that interpret and execute CWL workflows. In the past, many of these

execution engines did not have built-in support for GPUs, MPI, or integration with job schedulers like Slurm. Developers had to create custom solutions to handle these features. Although some workflow engines, such as Toil, made some preliminary steps in executing part of CWL workflows that support GPU configurations, the MPI parallelization remains a challenge.

Over time, as the CWL community continues to grow, develop and improve the specification, tools, and workflow engines, support for features like GPUs, MPI, and integration with systems like Slurm has become more feasible and standardized. Execution engines have been developed or enhanced to handle these features more effectively. Additionally, community-driven efforts have resulted in best practices and tools for integrating these technologies into CWL workflows, making it easier for researchers to leverage high-performance computing resources while using CWL for workflow description and execution.

3.2.2.3 Collaborative Brain Wave Analysis Pipeline (Cobrawap)

A summary of the scientific part of Collaborative Brain Wave Analysis Pipeline is presented next, with additional information available at [11].

The current variety of data from neuronal recordings paves the way to complementary approaches in the long-term goal of understanding brain activity. However, this poses the challenge to consistently compare data across experiments, species, and spatio-temporal scales, promoting the integration of multiple analysis techniques from different neuroscience sub-domains. Also, experimental data are essential to benchmark theoretical models, meanwhile providing insights for further developments. These requirements can be fulfilled by defining standardized, but still to some extent customizable, analysis workflows.

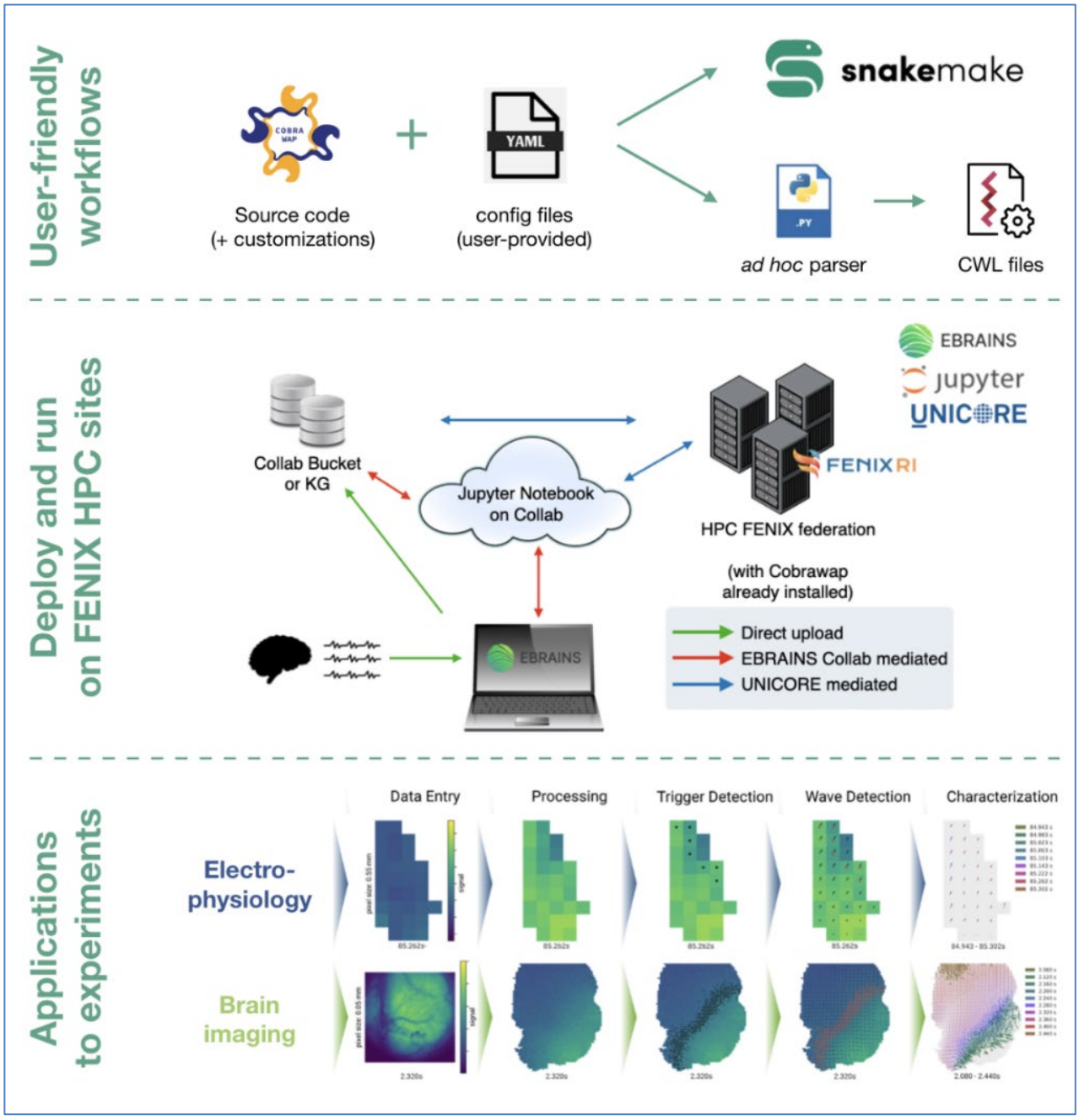

Figure 30: User-Friendly workflows.

Two different ways of defining Cobrawap pipelines via Snakemake and CWL workflows via ad hoc parsers; Deploy and run on FENIX HPC sites: Integration with various EBRAINS components, including Collab buckets via Data proxy, Jupyter notebooks and UNICORE to execute pipeline in the underlying infrastructure; Applications to experiments: Intermediate results of two datasets covering most of the right hemisphere of mouse brain

In the context of brain waves, Cobrawap [12] (Collaborative Brain Wave Analysis Pipeline) is developed and maintained by INFN, Jülich FZ and other European partners as an open-source software 17 within a reproducible and cooperative framework, adhering to FAIR principles. Cobrawap's source code is hierarchically structured as a collection of modular building blocks that implement various data processing algorithms and analysis methods, such as signal detrending, spatial down sampling and filtering. These components are organized into sequential stages, each corresponding to a different step in the analysis of wave analytics. The same data format for inputs and outputs is adopted throughout the entire pipeline. This design allows for flexible addition, removal, or changes of blocks within the respective stages. The final output of Cobrawap describes the cortical wave activity in a standardized manner, through quantitative observables. Recently,

[17] https://cobrawap.readthedocs.io

Cobrawap has been successfully applied to analyze and compare ECoG[18] and calcium-imaging[19] data from mice, and for model validation and calibration [13]. Latest efforts are focusing on the optimization of Cobrawap for the analysis of a novel dataset of imaging recordings from mice under isoflurane anesthesia [20], characterized by an unprecedented spatial resolution, through the development of a recursive algorithm able to identify the optimal data processing that allows for improving signal-to-noise ratio while preserving the most of the information from the high number of fluorescence signal sources on a cortical recording, at the same time taking into account the underlying biological brain structure and activity. Cobrawap is integrated with the Snakemake workflow manager which is based on Python syntax. In fact, the modular structure of the pipeline, organized into stages and blocks, has been designed to align with Snakemake components represented as files and rules.

A general drawback of complex analysis tools is the technical effort required for their initial setup and configuration. Additionally, there is a notable demand for computational resources when executing these tools on local machines. Also, it is crucial that proposed analysis procedures ensure reproducibility of the data analysis results, aligning with the FAIR principles. These requirements led to Cobrawap adopting the CWL approach, maintaining the same hierarchic organization. It consists of a CWL Workflow file[21] for each stage and a CWL command line tool file[22] for each block within a stage (Figure 28). Eventually, to execute the entire 5-stage analysis pipeline, an additional layer can be introduced, which includes another CWL workflow referencing the 5 stage-specific workflow files. This facilitates the expansion of the pipeline with additional stages, to address additional research objectives or for creating parallel pipelines that share some of their stages. To retain the same degree of block-level modularity flexibility as offered by Snakemake and given that CWL is not designed for interactive command-line tools that require user interaction during runtime, a set of Python scripts has been developed. These scripts are executed before the initiation of the entire workflow and parse the Python script responsible for individual blocks, compare their list of command-line arguments with the content of configuration files, which contain parameter assignments and information about the designated order of block execution, and adjust the default CWL workflow files accordingly.

3.2.2.4 Protein-ligand Docking

This use case is an example of an external workflow described via CWL which originates from a public workflows' repository (WorkflowHub).

The workflow focuses on illustrating the process of protein-ligand docking, using the BioExcel Building Blocks library (biobb). The example used is based on the "Mitogen-activated protein kinase 14 (p38-α) protein (PDB code 3HEC[23]), a well-known Protein Kinase enzyme, in complex with the FDA-approved Imatinib (PDB Ligand code STI[24], DrugBank Ligand Code DB00619[25]) and Dasatinib (PDB Ligand code 1N1[26], DrugBank Ligand Code DB01254[27]), small kinase inhibitors molecules used to treat certain types of cancer."[28]

[18] https://search.kg.ebrains.eu/instances/Dataset/2ead029b-bba5-4611-b957-bb6feb631396
[19] https://search.kg.ebrains.eu/instances/Dataset/cc72a888-4c93-464d-8a48-405d8beeee5e
[20] https://search.kg.ebrains.eu/instances/Dataset/5fbaeea7-fac2-472e-aefc-556c2cde1218
[21] https://github.com/APE-group/wavescalephant/blob/CWL_integration/pipeline/stage02_processing/workflow.cwl
[22] https://github.com/APE-group/wavescalephant/blob/CWL_integration/pipeline/stage02_processing/cwl_steps/spatial_downsampling.cwl
[23] https://www.rcsb.org/structure/3HEC
[24] https://www.rcsb.org/ligand/STI
[25] https://go.drugbank.com/drugs/DB00619
[26] https://www.rcsb.org/ligand/1N1
[27] https://go.drugbank.com/drugs/DB01254
[28] https://workflowhub.eu/workflows/259

The Common Workflow Language definition of this workflow has been developed "in the MMB group[29] at the BSC[30] & IRB[31] for the European BioExcel[32], funded by the European Commission (EU H2020 823830, EU H2020 675728)."

The default input and output parameters of the CWL workflow can be summarized in Table 5.

Table 5: Protein - ligand Docking input & output parameters

ID	Type	Document	Data Type
• *step1_fpocket_select_input_pockets_zip*	Input file	Path to the pockets found by fpocket.	*File*
• *step1_fpocket_select_output_pocket_pdb*	Output file	Path to the PDB file with the cavity found by fpocket.	*string*
• *step1_fpocket_select_output_pocket_pqr*	Output file	Path to the PQR file with the pocket found by fpocket.	*string*
• *step1_fpocket_select_config*	Config file	Configuration file for biobb_vs.fpocket_select tool.	*string*
• *step2_box_output_pdb_path*	Output file	PDB including the annotation of the box center and size as REMARKs.	*string*
• *step2_box_config*	Config file	Configuration file for biobb_vs.box tool.	*string*
• *step3_babel_convert_prep_lig_input_path*	Input file	Path to the input file.	*File*
• *step3_babel_convert_prep_lig_output_path*	Output file	Path to the output file.	*string*
• *step3_babel_convert_prep_lig_config*	Config file	Configuration file for biobb_chemistry.babel_convert tool.	*string*
• *step4_str_check_add_hydrogens_input_structure_path*	Input file	Input structure file path.	*File*
• *step4_str_check_add_hydrogens_output_structure_path*	Output file	Output structure file path.	*string*
• *step4_str_check_add_hydrogens_config*	Config file	Configuration file for biobb_structure_utils.str_check_add_hydrogens tool.	*string*
• *step5_autodock_vina_run_output_pdbqt_path*	Output file	Path to the output PDBQT file.	*string*
• *step5_autodock_vina_run_output_log_path*	Output file	Path to the log file.	*string*
• *step6_babel_convert_pose_pdb_output_path*	Output file	Path to the output file.	*string*
• *step6_babel_convert_pose_pdb_config*	Config file	Configuration file for biobb_chemistry.babel_convert tool.	*string*

3.3 Automated testing of Jupyter Notebooks

In the ongoing pursuit of elevating the quality and reliability of Jupyter notebooks within EBRAINS ecosystem, enhancements have been implemented within the automated headless browser testing service (D5.4 [2], Section 3.3). As described in D5.4, the service played a pivotal role in ensuring the quality and reliability of publicly accessible Jupyter notebooks by verifying the availability of

[29] http://mmb.irbbarcelona.org/
[30] http://www.bsc.es/
[31] https://www.irbbarcelona.org/
[32] http://bioexcel.eu/

essential resources. The current Section clarifies the new features incorporated into this service, building upon the groundwork established in the preceding phase.

- **Enhanced Notebook Management**: Notebook owners were previously required to communicate with the TC team for notebook unregistration or metadata updates. This requirement has been dropped via an enhanced service interface, enabling owners to unregister and update their notebooks directly.
- **Customizable Execution Frequencies**: Acknowledging the heterogeneity in notebook execution requirements, a feature has been introduced that permits notebook owners to define the execution frequency for their notebooks. This facility allows notebooks to be executed at varying intervals based on the predetermined frequency, thus achieving greater flexibility and resource optimization.
- **Integration with Centralized Monitoring and Visualization**: To enhance the monitoring capabilities, an integration has been established between the service and the centralized monitoring service (Section 3.4) of the EBRAINS RI. After each notebook execution, the outcomes are sent to the centralized monitoring service, being securely archived and readily accessible. This integration allows administrators to identify any anomalies and, hence, increase the overall reliability and performance of Jupyter notebooks. Further, notebook owners can now access testing results through the provided visualizations within the monitoring service, in addition to the existing email notifications providing links to the GitLab CI artifacts folder.

3.4 Monitoring services

The EBRAINS RI monitoring services have been fully implemented with an architecture designed for scalable and efficient monitoring of all applicable software components across the platform, primarily tailored for services, APIs, and web applications. The objective was to create a centralized monitoring service embodying the "all in one place" capability, facilitating not only data collection and analysis into one centralized location, but also delivering valuable insights to both admin and component owner levels.

The architecture of the monitoring system (Figure 31) is based on the Elastic Stack[33], comprising Elasticsearch, Logstash, Kibana, and Beats. This stack allows for an extensive data collection and analysis system, where metrics are collected from each component using Beats agents or custom scripts.

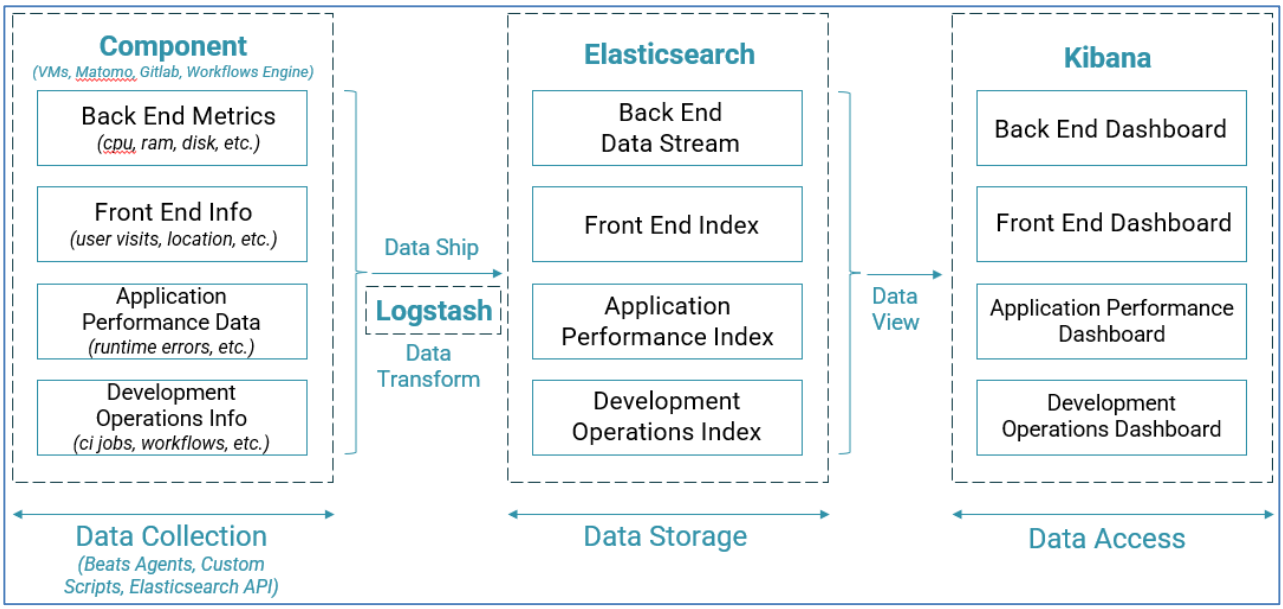

Figure 31: Monitoring Architecture and Data Pipeline

[33] https://www.elastic.co/elastic-stack

3.4.1 Data Collection and Transformation

We focus on a multi-level global monitoring view, collecting data via Beats agents or custom scripts across four levels: Back-End, Front-End, Application Performance, and Development Operations for comprehensive insight into component performance.

- **Back-End System Metrics**: Utilizing Metricbeat agents on each component's VMs, we collect key metrics like RAM and CPU usage, shipping them to Elasticsearch.
- **Front-End UI Metrics**: Custom scripts are utilized to extract data from the EBRAINS Matomo platform, capturing user visits on each web application component.
- **Application Performance Metrics**: Component owners send logs via the Elasticsearch API based on their specific needs.
- **Development Operations Metrics**: Custom scripts are utilized to collect workflow, notebook, and CI job execution results. Through this data, the performance, stability, and reliability of platform operations are enhanced.

3.4.2 Data Storage

After data collection, the data is stored into Elasticsearch. During the onboarding process for every component, the admin team creates all necessary data structures, with each data category specific to a component having its own dedicated structures, named index or data stream within Elasticsearch. The main differences between indices and data streams, along with when and which type is used are described in Annex 8.2.1. This setup not only streamlines the process of data storage and retrieval but also ensures efficient and accurate querying of the data when required. Moreover, this arrangement guarantees that each component owner will have access only to indices related to their components, maintaining data integrity and access control. More details for the onboarding process can be found in Annex 8.2.2. Lastly, backup policies have been implemented for data backup and retention to ensure data durability and availability over time. Those policies are described in more detail in Annex 8.2.3.

3.4.3 Data Access and Visualization

The final part of our monitoring system implementation involves data access and visualization, accomplished via Kibana. This is where component owners can interact with their data.

Kibana provides a user-friendly interface to visualize the data stored in Elasticsearch in a way that's easy to understand and interpret. Component owners are provided with their credentials by the Kibana administrator, who controls the permissions for each Kibana space. This controlled access mechanism ensures the confidentiality and integrity of the data, allowing only authorized users to view specific data and dashboards.

Each component has a dedicated Kibana space (Figure 32), which can be understood as an independent work area where dashboards and visualizations can be stored. Each space is customized to show only the data relevant to the corresponding component.

Inside their Kibana space, component owners access tailored Dashboards per monitoring level of their component. The Back-End Metrics Dashboard provides insights into VM performance and ongoing processes. The Front-End Visits Dashboard, resembling an EBRAINS Matomo subset, offers user interaction insights on the component's webpage. The Application Performance Dashboard, customized to component requirements, sheds light on application-level performance metrics. And, the Development Operations Dashboard displays a snapshot of CI job metrics, simplifying pipeline health assessment, aligning with the central monitoring objective.

Finally, alongside component-specific dashboards, administrative-level dashboards have been introduced to provide a comprehensive view of all components and EBRAINS infrastructure not tied to specific components. The Overall System Performance Dashboard provides a snapshot of system-

wide performance metrics for all registered components. The EBRAINS Website Visits Dashboard focuses on the website traffic and engagement, the Workflows Executions Dashboard provides insights into workflow performance and resource usage and the Test Executions Dashboard offers a detailed overview of test performances within the notebooks system, including test count, chronological executions, and operational mode divisions, broadening insights into infrastructure operations beyond individual components.

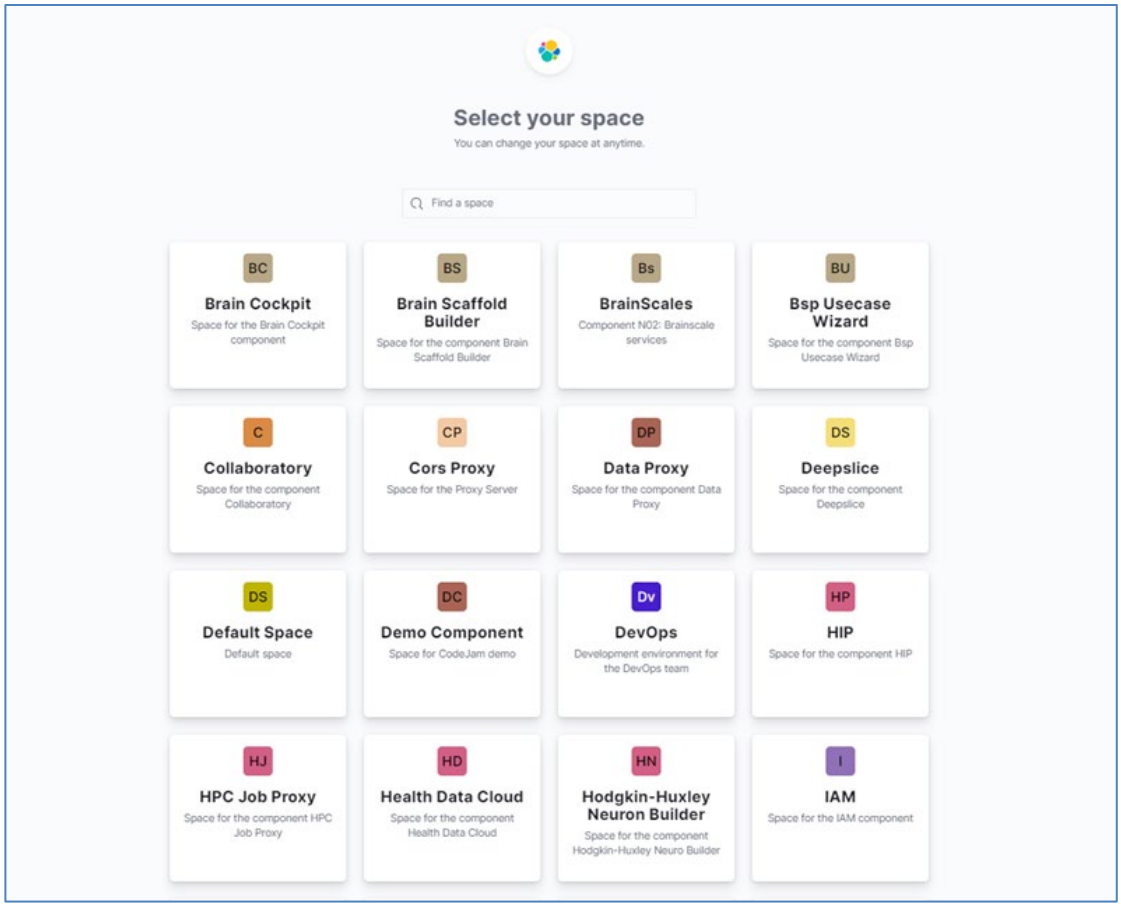

Figure 32: Kibana Spaces

Detailed description for all mentioned dashboards can be found in Annex 8.2.4.

3.4.4 OpenSearch migration

The current approach to data collection and analytics is built upon the Elastic Stack. The setup involves collecting data relevant to each component through the utilization of Beats agents or custom scripts. This data is then transformed or combined if needed, before being loaded into Elasticsearch (Figure 33). Despite the significant benefits of this technical approach, the Elastic Stack has recently received critical backlash from the community for the transition of several key features behind a paywall and a commercial license. Two such features critical also for the EBRAINS RI are: Single Sign-On (SSO) and Document Level Security (DLS).

- The monitoring service needs to align with the standard SSO approach in which users are authenticated through their EBRAINS credentials. Implementing SSO, however, currently requires a license under Elastic Stack.
- Instead of duplicating indices, data views, and dashboards across components, Document Level Security (DLS) controls access to individual documents within an index and dashboard. Thus, the data is stored in a single index, visualized on a unified dashboard, with DLS ensuring each component owner has access only to their data based on their roles. Much like SSO, implementing DLS within the Elastic Stack framework currently requires a license.

In our pursuit to meet these requirements, we turned our attention towards OpenSearch[34], a community-driven, open-source fork of Elasticsearch and Kibana. OpenSearch was created by Amazon Web Services in response to Elastic altering the licensing terms for Elasticsearch and Kibana from the open-source Apache 2.0 to the Server Side Public License (SSPL). OpenSearch, as an open-source platform, offers a publicly visible roadmap, providing transparency and planning security.

OpenSearch and OpenSearch-Dashboards, as successors to Elasticsearch and Kibana, maintain much of the older software's functionality while operating under the Apache 2.0 license. This license promotes free use, modification, and distribution of the software. Importantly, OpenSearch supports implementing Single Sign-On and Document Level Security, addressing the limitations of the initial Elastic Stack approach. Additionally, it is compatible with Beats agents and Logstash, which are key elements in our data architecture.

Given these advantages and capabilities, we decided to migrate the EBRAINS Monitoring Services from Elasticsearch to OpenSearch. All back-office facilities, agents, pipelines and dashboards are already in place, with the actual production transition taking place during the planned migration presented in Section 6.4. This enables us to ensure the uninterrupted operation of the Monitoring Services and transition into OpenSearch with no loss of data or functionality.

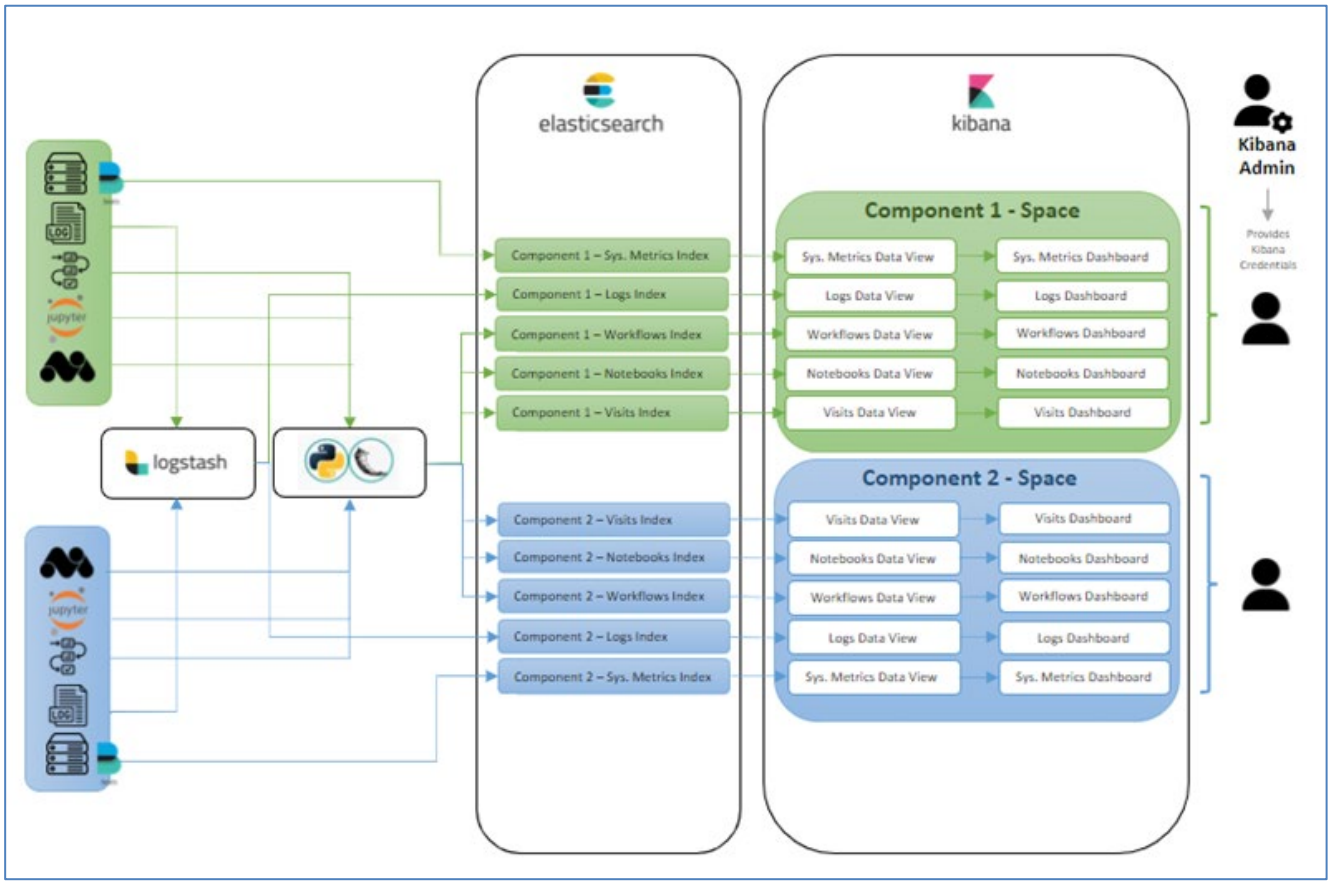

Figure 33: Monitoring Service Architecture based on ELK Stack

Once we migrate to OpenSearch, the data will be collected for each component using Beats agents or custom scripts. This data will also be transformed or combined if needed and then loaded into OpenSearch, storing data from common categories in shared indices across components (Figure 34).

[34] https://opensearch.org/

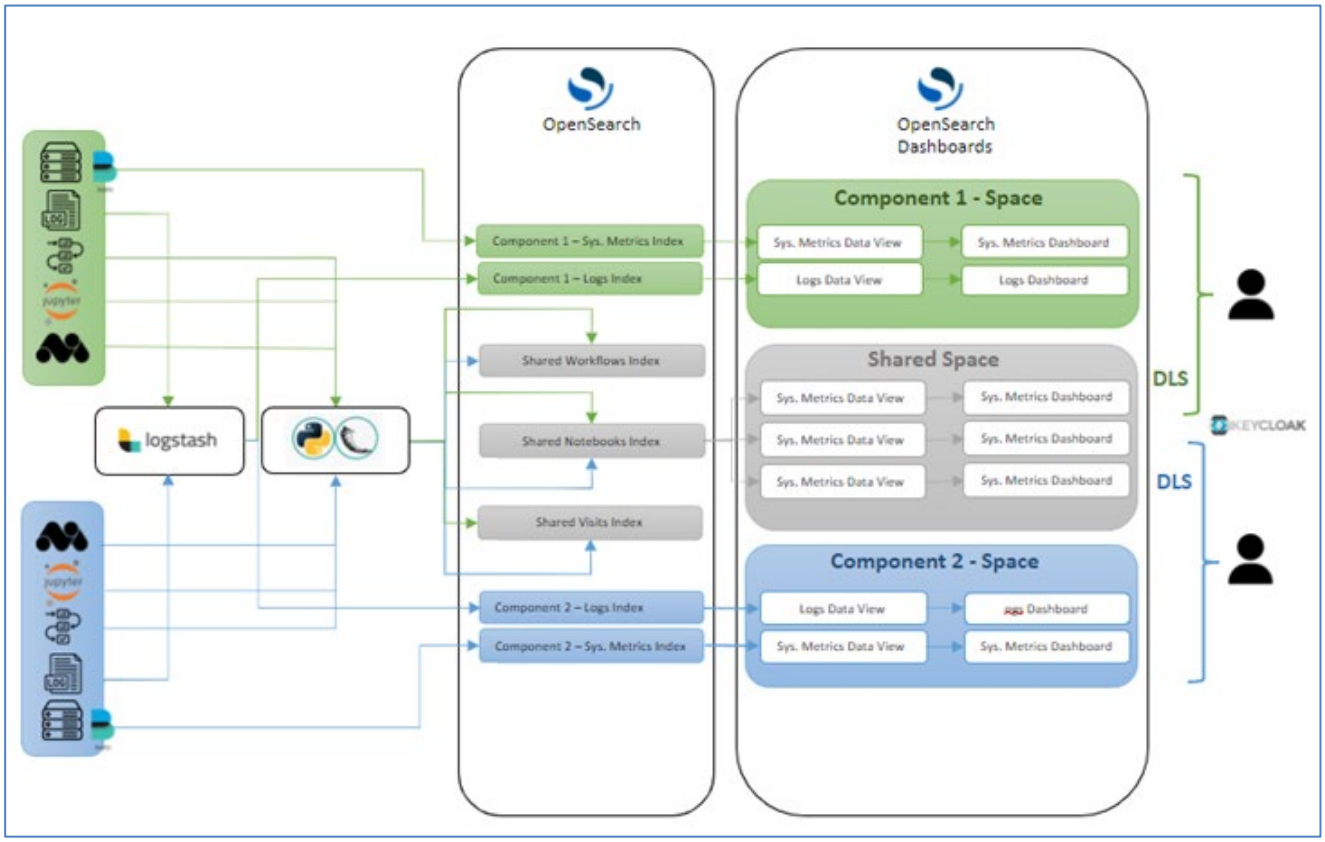

Figure 34: Monitoring Service Architecture based on OpenSearch

Component owners will access their data and dashboards through OpenSearch Dashboards, authenticated via their EBRAINS credentials through Keycloak. Through the implementation of DLS, owners will be granted access to their private and shared spaces where they can only view their data and dashboards. Finally, OpenSearch comes with built-in support for alerting, and enhancing the capability to monitor and respond to system incidents.

3.5 Dashboard

The EBRAINS Dashboard serves as a user-centered interface, offering a wide range of capabilities and providing an overview of various features such as standardised workflows, EBRAINS news, user settings and integration with EFSS (Enterprise File Sync and Share). Initially, the primary focus concentrated on Standardised workflows, which prompted the development of the Workflows Dashboard. However, it has since evolved to incorporate a range of additional features that provide even more utility to users.

3.5.1 User Interface & Functionalities

EBRAINS Dashboard concentrates functionalities in a single web page. It provides efficient control over operations and seamless user experience. The User Interface together with its functionalities are further explained in the upcoming sections.

3.5.1.1 User Data & Sharing

The Dashboard lets EBRAINS users store their data using either Amazon S3 Buckets or in the integrated NextCloud instance. Below, these options are explained in detail:

- **Buckets**: These are Amazon S3 Buckets integrated using the EBRAINS Data Proxy API[35]. EBRAINS users are able to navigate and access only the buckets and objects they have permissions for. They can perform actions such as Downloading, Deleting, as well as Searching for specific buckets. Buckets also serve as the primary storage for CWL workflows executed through the Workflows Section Dashboard, where intermediate results and final outputs are stored.
- **Files**: Every EBRAINS user has access to a File Storage system which is implemented using NextCloud [36] platform. They have access to their files and can obtain information about file/folder size and ownership.

3.5.1.2 Workflows

Upon entering the EBRAINS Workflows Dashboard, EBRAINS users gain access to their personal workspace, which contains a list of CWL workflows. Each workflow in the list is tagged with important metadata, including its creation date and the date of last execution. The listed workflows can either be uploaded by the users themselves or fetched from 'Featured workflows' section and saved in the personal workspace. This provides a cohesive hub for workflow management offering users a range of options:

1) **"View"** action consists of:
    a. **"Run"**: Users may choose to execute the workflow using its pre-defined parameters, or they can customize the parameters to suit their specific requirements. Dashboard automatically detects if a field seems to be a token; in that case, it prefills the field in the back-end with the user's access token, without exposing it to the UI. However, the user still has the option to overwrite the field with any value.
    b. **"INFO"**: This section provides a concise summary of the workflow, introducing its (scientific) purpose, input parameter types, and a visual representation, or else a preview, of the standardised workflow including all input and output parameters and workflow steps. Data dependencies of different workflow steps are highlighted upon user interaction, enhancing their understanding of data flow within the workflow.
    c. **"EXECUTIONS"**: Users can access a list of previous executions of every workflow, each tagged with essential metadata such as infrastructure, start time, duration, and execution status, spanning from success, failure or running. By expanding single executions of the list, users can gain more detailed information for each one of those:
        i. INPUT: This includes input parameters provided during the runtime.
        ii. TASKS: Users may check the status of different workflow steps, spanning from completed to failed
        iii. LOG: Logs are available for all workflow executions. In cases where a workflow encounters execution errors, users may redirect themselves to logs for deeper understanding of the issues. They can alleviate problems by editing the workflow and / or workflow steps.
        iv. OUTPUT: When workflows executions are successful, users can easily access each workflow executions' outputs. In cases where the output contains images, an embedded viewer is integrated offering an immediate visual outcome of the workflows. There is also the possibility for users to download the workflow outputs to their local devices.

[35] https://data-proxy.ebrains.eu/
[36] https://nextcloud.com/

d. "**RECIPE**": EBRAINS users can view and dynamically alter the workflow recipes through an embedded viewer. They can add or remove workflow steps to create their own unique workflows.

2) "**History**": This option redirects users to the EXECUTIONS section page for each workflow.

3) "**Mark as Featured**": Workflows can be shared with the rest of the EBRAINS users by proposing them as Featured Workflows

4) "**Delete**": This option allows users to remove a workflow from the EBRAINS Workflows Section Dashboard.

Some of the available functionalities within Workflows Section Dashboard are:

1) 'Featured Workflows': Presently, the EBRAINS Workflows Dashboard hosts a collection of three (3) Featured Workflows. As elaborated in Section 3.2.1, two of these originate from EBRAINS Showcases (3 and 4) while the third one extends its origins beyond EBRAINS and specifically from BioExcel's ecosystem. For each 'Featured Workflow', EBRAINS users gain access to a comprehensive repository of informative details. These include a concise summary of the workflow introducing the (scientific) purpose of the workflow, input parameter types, and a visual representation, or else a preview of the standardised workflow. They can explore the workflow recipes and the different tools that were used as workflow steps to construct the workflow. EBRAINS users can seamlessly fetch and save these workflows into their personal workspace in case they find a particular workflow which aligns with their research objectives. By fetching these into their own workspace, they can execute them in their original form with already available predefined input parameters or modify the workflow recipe by adding or removing steps to suit their needs and create different workflows by containing the core functionalities. EBRAINS users could mark this new version of the workflow as "Featured Workflow", expanding its accessibility within the EBRAINS consortium. An administrative team is responsible for accepting or rejecting the newly marked Featured Workflow. If accepted, all EBRAINS users can then access the new Featured Workflow and decide if they wish to execute it in its original form or alter it to create their own. This approach to workflow development embraces the essence of reusability, which either way is strictly related to standardized workflows, creating a new era within EBRAINS ecosystem for research and innovation.

2) 'Create Workflows': EBRAINS users are also able to create their own standardised workflows in their personal workspace at the Workflows Section Dashboard by submitting the workflow files' recipes, workflow steps and workflow input parameters. The process can be concluded in two ways:

   a. Uploading zip files: EBRAINS users can upload a "zip" file containing the workflow recipe file together with every workflow step and the workflow input parameters for providing a pre-defined way of executing the workflow.

   b. Providing a public GitLab repository link: EBRAINS users can also fill in an external URL that points to a public "git" repository that contains the above files.

When creating a workflow using either of the two ways, users are requested to import essential details of the workflow, such as a workflow summary together with workflow parameters information. When the workflow is uploaded, a validation check is implemented by the back end, checking if required files are missing. If the workflow is valid, a graphical representation, or a workflow preview, is automatically generated. Each EBRAINS user can propose their workflows as 'Featured Workflows' in cases where workflows should be publicly available allowing others to access, modify and execute them. After a review process which takes place by Workflow Dashboard administrators, the proposal may be rejected if essential details are missing, or accepted if the proposed workflow is accepting the criteria. By proposing workflows to be Featured ones, they are made public to all EBRAINS users, who can view the details of each workflow (such as Summary, Parameters and the visual representation) as well as fetch and save them in their user-specific workspace under their workflows list. A notification system is in place to notify EBRAINS users when new Featured Workflows become available within the EBRAINS Dashboard or in case that has been

rejected. In addition, the same system notifies the admins about new requests by the workflow owners.

3.5.1.3 User settings

EBRAINS users can add and customize information related to their profile. From the "Settings" section, users can edit information that is related to the Profile image (user can upload an image from his computer), *First name, Last name, Email, ORCID, Affiliation, Organization department, Personal site and Areas of interest*, which users can select as predefined tags.

3.5.2 Technical Specifications

Delving into the elaboration of the Workflows Section Dashboard, it is essential to comprehend its three fundamental components: the front-end, backend and middleware. Each plays a distinct but interconnected role in the seamless functioning and user experience of the platform.

3.5.2.1 Frontend

EBRAINS's Dashboard user interface (UI) is developed using Vue.js[37] which is a modern JavaScript framework for creating Single-Page Applications (SPAs). It builds on top of standard HTML, CSS, and JavaScript and provides a declarative and component-based programming model. Vue.js 3 is compatible with TypeScript and Sass which are described below. Most of the UI components are developed using TypeScript, a superset of JavaScript that transpiles to pure JavaScript. It is a free and open-source high-level programming language developed by Microsoft that adds static typing with optional type annotations to JavaScript. It is designed for the development of large applications, giving better tooling at any scale. The styling of components is written in Syntactically Awesome StyleSheet (Sass), a stylesheet language that's compiled to CSS. It allows you to use variables, nested rules, mixins, functions, and more, all with a fully CSS-compatible syntax. Sass helps keep large stylesheets well-organized and makes it easy to share design within and across projects. For state management, the Pinia library is used. Pinia is currently the official state management library for Vue.js. It uses declarative syntax and offers its own state management API.

3.5.2.2 Backend

The backend consists of the following two components:

- A REST API that is used by the dashboard UI to execute all the required CRUD operations.
- A client tool for running workflows in a remote server and monitoring its progress. The client tool is running as a sub-process of the REST API.

The Restful API is written in Node.js, which is an open-source, cross-platform JavaScript runtime environment. It is an asynchronous event-driven JavaScript runtime, designed to build scalable network applications.

The REST API app uses Express.js, a minimal and flexible Node.js web application micro-framework. It is the most popular among all Node.js frameworks. It provides a set of tools for web applications, HTTP requests and responses, routing, and middleware for building and deploying large-scale, enterprise-ready applications.

Code is written in Typescript, to take advantage of the benefits that are mentioned in the previous sub-section.

The REST API uses Object-Relational Mapping (ORM) to add an abstraction layer between Node.js and the Relational Database (RDBMS). Specifically, it uses TypeORM which is an ORM that has native

[37] https://vuejs.org/

support for TypeScript and is designed to develop high-quality, loosely coupled, scalable, maintainable applications in the most productive way.

Data is stored in a PostgreSQL Relational Database (RDBMS). PostgreSQL is a powerful, open-source database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads. PostgreSQL has been proven to be highly scalable both in the sheer quantity of data it can manage and in the number of concurrent users it can accommodate. There are active PostgreSQL clusters in production environments that manage many terabytes of data, and specialized systems that manage petabytes.

3.5.2.3 Middleware

To implement the middleware of the Workflows Section Dashboard, we relied on the TES (Task Execution Service) Standard38. This involves an API with a standardised schema for describing and executing batch execution tasks, along with a TES server. In our Workflows Section Dashboard implementation, we have deployed cwl-tes39 client which is responsible for orchestrating workflow submissions. The client submits workflows to TES server and translates them to TES tasks. For each step within a workflow, a corresponding task is created. We also employ TESK40 as the TES server, which stands as one the two existing (2) backend components. The TESK server is based on the TES Standard and operates within the OKD distribution of Kubernetes. We operate two instances of the TESK server, with one deployed on the OKD platform at CSCS and the other one OKD at JSC. Its architecture consists of a running pod, which acts as the web server that consumes TES requests, validates them and translates them into Kubernetes pods. When feasible, TESK also handles task-level parallelization. After a workflow is submitted via the cwl-tes client to the TESK server, the web server creates one task controller or a taskmaster for each task. These task controllers then create filler pods, which are responsible for initiating Persistent Volumes Claims (PVC) and creating all the necessary executor pods for task execution. Once the final executor completes its task, the filler pod takes over. It processes the task's outputs and ensures they are pushed to remote storage locations.

For the storage of intermediate and final outputs of the submitted workflows, we utilize Archival Data Repositories, infrastructure resources provided by both CSCS and BSC FENIX ICEI sites, as remote storage solutions. The EBRAINS Data Proxy API is responsible for retrieving the results from these locations back to the Buckets section inside Dashboard. For EBRAINS users, the entire mechanism of workflows submissions, executions, possible parallelization of tasks and final outputs handling occurs seamlessly behind the scenes, ensuring a streamlined and user-friendly experience.

[38] https://github.com/ga4gh/task-execution-schemas
[39] https://github.com/ohsu-comp-bio/cwl-tes
[40] https://github.com/elixir-cloud-aai/TESK

4. Implementation of the Integration process (Delivery and Software Quality)

4.1 Software Delivery

In this Section, the main developments and changes during Phases 3 & 4 relevant to the EBRAINS Software Delivery strategy are outlined. Software Delivery evolved greatly during the project by constantly adapting to the evolution of the architecture and to the continuously increasing number and type of components that needed to be integrated. EBRAINS RI is a complex e-infrastructure relying upon federated, multi-site, base computing resources provided by the Fenix RI and the Software Delivery strategy ought to mitigate various relevant challenges regarding the delivery process and the successful utilisation/operation of several service environments. Case-dependent deployments (both single and multi-site) only added to the challenges that need to be addressed.

As documented in the Software Delivery section of the Technical Coordination Guidelines (D5.3, Section 4.2.3, [1]), Phase 1 concluded that individual (component specific) deployments are efficient (fast, bottom-up) but difficult to manage at scale. To sustainably operate EBRAINS, we needed to evolve towards a centrally coordinated and horizontal deployment process.

During Phase 2, as documented in the Interim EBRAINS Infrastructure Implementation Report (D5.4, Section 4.1, [2]), the delivery strategy was further improved and became more implementation-specific. New concepts were introduced and applied with a view towards cost-effective, scalable and sustainable operation of the EBRAINS RI. In essence, the main strived-for axes of Software Delivery in EBRAINS were detailed. These axes (which were extensively documented in D5.4) formed the base for solidifying software delivery within EBRAINS during Phases 3 & 4 and are outlined below for reference:

- **Axis 1**: The concept of **top-level automated (code) monorepos** as a fundamental integration point for all components was introduced instead of having a single platform-wide monorepo. A decision was reached to maintain independent automated monorepos (with repos included as git-submodules) per deployment-type (e.g. VMs/VNets, process-containers/pods, HPC environment-modules, notebooks).
- **Axis 2**: The notion of **Namespaced integration subdirectories** by way of standardised base-directory names in component repositories was encouraged for all deployment types according to the implementation for software delivery and deployment for the Collaboratory Lab (see Section 3.1).
- It was identified that such "namespaced" directories (contrary to "namespaced" git-tags which was also considered) facilitate easy maintenance of a dedicated "EBRAINS" branch, primarily for delivery/deployment, with easy & regular merging of the upstream branch onto it. As long as the upstream branch avoids any identically-named base-directories this can be done purely with automated fast-forward merges, avoiding history-rewriting (rebasing or merge-conflict resolution). Its pipelines would use already-built artifacts from the main repository where possible (from public or EBRAINS registries if already uploaded, or at least delegating the build to the repository's main build configuration), only initiating context-specific building for EBRAINS when required. This provides a clear delineation-point between each component's (A) independent build/unit-test logic, and (B) EBRAINS-specific system-test/delivery/deployment logic.
- Components needing simpler onboarding could initially retain oversight of EBRAINS-specific "continuous delivery" by keeping the "EBRAINS" branch in their repository - but with EBRAINS DevOps coordinating, manually auditing, continuously delivering and manually deploying to production via an internal mirror, not "continuous deployment" from the external repository. Some central components could retain control of the "EBRAINS" branch in their repository longer term (but still with EBRAINS DevOps coordination and oversight), which would be discussed and agreed on a case-by-case basis to facilitate the integration process and alleviate some operational overhead from the EBRAINS DevOps team.

- **Axis 3**: The approach of **Everything as a trackable component** which can be described as facilitating the integration process by handling (almost) everything within the EBRAINS RI as an organisationally trackable component with a responsible “component owner”. This means that every "component owner" is required to create & maintain thorough hierarchical "resource maps" (tree of dependencies of all shared resources) for that component, right down to "external resources" which are those beyond EBRAINS TC/AISBL administrative control as far as the platform is concerned (those provided by Fenix RI, UMAN, UHEI, external service-providers, etc). Even material developed by TC would be "just be another component". It is worth reminding, from D5.4, that the types of components (in ascending order of maintenance-burden for TC/AISBL) could be:
    - Open/free components with upstream maintainers (e.g. OKD, Spack, Nginx)
    - Community "contrib" (unsupported) components, provided by EBRAINS users, when platform operational status will allow for such “plugin” contributions.
    - Officially supported components contributed by HBP consortium maintainers (e.g. TVB, NEST, KG, Cosim framework, Provenance API)
    - Base functionality and middleware components contributed by EBRAINS DevOps (e.g. Collaboratory, Datamover, Container Orchestration Engine, API gateway)
- **Axis 4**: The technique **Test-Driven Integation (TDI)** analogous to test-driven development which was applied to TC’s internal work and was assessed for possible external use. TDI has proven effective for integrating disparate components from separate teams in as "lean" and "agile" a manner as possible. This technique ties with the top-level automated monorepos per deployment type concept to streamline the integration process.

During Phases 3 & 4, software delivery leveraged the conclusions of Phase 1 and Phase 2 towards a balanced combination of the two approaches. This was necessary to effectively facilitate the vast number of components participating in the integration process in a timely manner. In essence, software delivery revolved around the 4 axes established during Phase 2 but capitalized on the swiftness of individual (component specific) deployments that occurred during Phase 1 wherever necessary. Out of the 4 axes, it gravitated towards some of them as a result of the development effort of the various component teams, the inclusion of a large number of entirely new components to the integration process during Phase 3, and the various particularities of the EBRAINS operating environment as a consequence of the challenges posed by the need to deploy on top of federated base infrastructure resources.

Therefore, during Phases 3 & 4, software delivery gravitated towards “Everything as a trackable component” and “Namespaced integration subdirectories” for most of the EBRAINS components. “TDI” and “top level automated code monorepos” were mostly followed in TC internal work. Some components also followed individual deployments as discussed and agreed on a case-by-case basis to facilitate the integration process. It needs to be mentioned that even if some components followed individual deployments as agreed, they had to comply responsibly with Axis 3 to maintain the coherence of the software delivery strategy.

The integration requirements that all components were required to fulfil to be signed-off as fully technically integrated to the EBRAINS RI reflected the developments and changes in the Software Delivery Strategy. For completeness, the integration requirements relevant to Software Delivery are presented below:

- All components need to have their official code repository mirrored at EBRAINS Gitlab [Axes: 3,2]
- All components need to have their code automatically tested via dedicated CI pipelines [Axes: 4,1]
- Conformance to the available deployment options
    - VM, Containerized, HPC, and Standardised workflows deployments were eventually handled by the respective component teams [Axis 3, Individual component specific deployments]

- EBRAINS tools software stack which involves the availability of EBRAINS software libraries in the Collaboratory Notebooks and HPC systems is handled by the EBRAINS DevOps team centrally. [Axes: 2,4]

- Container images that are involved in the component's deployment(s) on EBRAINS should be pushed to EBRAINS image registry [Axis 2, individual component specific deployments]
- Solid deployment strategy with appropriate deployment environments as prescribed in the "Software Delivery" guidelines [At least comply with Axis 3]

4.2 Software Quality

Software quality (SQ) standards for all EBRAINS software components have been established and are maintained by the EBRAINS Software Quality Working Group (ESQ-WG). As detailed in D5.4 [2], Section 4.2, the group has conducted monthly meetings and maintained close collaboration with the TC team to create a comprehensive documentation (the "guidelines document") that encompasses requirements and guidelines for all EBRAINS developers. The documentation is accompanied by a checklist that summarizes the core points of the document's chapters. It includes guidelines that were weighted based on three priority keywords:

- '**Must**': Requirements - essential elements that are mandatory and must be adhered to by component owners.
- '**Should**': Recommendations - practices that are highly recommended for the efficient operation and maintenance of the EBRAINS RI and the development and maintenance of its components.
- '**May**': Best Practices - practices that, while not mandatory, are extremely beneficial and enhance the overall quality and performance of the EBRAINS software components, making them more robust and efficient.

During the last year of the project, through collaborative efforts, multiple sections of the document were revised, resulting in an enriched and refined second version. The guidelines document can be found in the folder of D5.7 material[41]. The checklist has also undergone a transformation, transitioning from its original format in the Excel sheet to an interactive self-assessment PDF, available in the same folder[42]. This new format empowers component owners with the ability to easily mark and unmark requirements and guidelines within the checklist, offering a dynamic and user-friendly approach to self-assessment. This self-assessment tool has been seamlessly incorporated into the final integration phase of EBRAINS Components. As part of this integration, curators from the TC team assess the self-assessment checklist results and categorize them into one of three distinct categories:

- '**Passing**': the component meets the essential requirements.
- '**Silver**': the component not only meets requirements but also excels in implementing recommendations.
- '**Gold** ': The component not only fulfills all requirements and recommendations, but also exemplifies effective software quality practices.

4.3 Sensitive data and integration

EBRAINS, taking into consideration the possibilities and regulations[43] within the Fenix RIfor storing and processing personal data, has offered several pathways to EBRAINS users for accessing and analyzing sensitive human data. The Health Data Cloud (HDC)[44], which is built upon an existing

[41] https://drive.ebrains.eu/d/6061531326d048308823/
[42] https://drive.ebrains.eu/d/6061531326d048308823/
[43] https://fenix-ri.eu/content/processing-personal-data
[44] https://hdc.humanbrainproject.eu/login

GDPR-compliant and EBRAINS interoperable Virtual Research Environment (VRE)[45] located at Charité, came to a production-ready state. VRE provide a secure and scalable data platform enabling multi-institutional research teams to store, share and analyze complex multi-modal health datasets. The HDC service offers EBRAINS users the capability to seamlessly use the VRE service by logging in with their EBRAINS account in order to access and use sensitive data in their research. For more details regarding the HDC service, readers are redirected to D6.7 [16].

The precedent of HDC service was the Human Data Gateway (HDG), which served as a temporary solution until HDC reached a production-ready state. Through HDG, EBRAINS users were able to request access to datasets under controlled access directly from the dataset cards inside EBRAINS KG. Given the sensitive nature of the data, the process involved manual steps where users had to request access, receive an email containing a Data Use Agreement (DUA), click the link on that email and accept the DUA. With the DUA acceptance, the user gained access to a data-proxy UI, allowing them to inspect the objects of the controlled access dataset. Additionally, there was a programmatic method for third-party services accessing a controlled access dataset on behalf of the user. The same process involved back-and-forth email exchanges with the user interested in the controlled access dataset to ensure compliance and authorization. HDG was part of the EBRAINS Data proxy service with the latter being fully integrated with the EBRAINS RI ecosystem.

Two more EBRAINS services, the Human Intracerebral EEG Platform (HIP) and Medical Informatics Platform (MIP), complete the picture of services dealing with sensitive data within the EBRAINS ecosystem. HIP is a dedicated platform designed for the collection, storage, sharing and analysis of Human intracerebral EEG data recorder by patients with drug resistance in epilepsy undergoing pre-surgical evaluation using stereo-EEG (SEEG). HIP offers unique capabilities, such as facilitating transfers of data from hospitals to the platform in a secure way, converting EEG files into BIDS-SEEG format, providing workflows for precise localization of SEEG contacts on patient's MRI scans, and running advanced SEEG analytical tools. These tools provide value in the understanding of the human brain, offering spatial and temporal resolution, enabling the recording of single neuron and multi-unit activity, and completing existing data sources [17].

MIP[46], focuses on creating virtual databases that interconnect data from diverse sources into a common data schema. This data federation allows neuroscientists to access and analysed data from various diseases (neurological and psychiatric) without moving it from one location (hospital) to another (including hospitals, cloud) ensuring data privacy. MIP serves as an innovative data analysis system which offers interfaces to analyse and access neuroimaging, clinical, -omics and medical records data. Users can customize workflows with existing algorithms, explore various pathologies[47] (including but not limited to Dementia, Epilepsy, Traumatic Brain Injury etc.), compare datasets and run experiments helping clinicians and clinical researchers to promote neuroscience research using hospital medical data [18].

As described in D5.15 [8], "while HIP, MIP, HDC appear to EBRAINS users as functionally integrated and with a seamless user experience including login in with EBRAINS IAM, accessing Drive data and making use of JupyterLab, they are nevertheless deployed, managed and provisioned as independent and sandboxed full stack services at all their levels (IaaS cluster, network policies, internal user/group policies and data ingress/egress)". The TC team placed particular emphasis in the deployment, integration, and security of these services, given the rising interest in sensitive data management and processing via Trusted Research Environments (TREs) and their five (5) Safes (Safe people, Safe data, Safe Projects, Safe Outputs, Safe Projects and Safe Settings) [19].

[45] https://vre.charite.de/vre/
[46] https://mip.ebrains.eu/
[47] https://mip.ebrains.eu/#Federations

5. Technical Coordination

EBRAINS **Technical Coordination** (TC) is responsible for the planning, design, integration, and delivery of the EBRAINS RI. It encompasses activities such as requirements elicitation, functional and operational specification definition, architecture design, and validation that the delivered infrastructure addresses the needs of the offerings.

The current section presents the update of the **KPIs** for monitoring and assuring the quality of the EBRAINS RI, specifically for the integration process, along with the achieved values for the whole duration of the project as evaluated by the TC.

Further, we present the status of the development/implementation phases that have been established to keep track of the evolution and progress during SGA3 and the status at the end of the project, along with the risks we have faced. All four Phases of Integration are presented in detail.

5.1 EBRAINS service evaluation methodology

To assess the quality of the EBRAINS Infrastructure integration process, the EBRAINS Technical Coordination team introduced a set of Key Performance Indicators (KPIs). The purpose of the KPIs were: (a) to facilitate the components alignment with the EBRAINS architecture and integration guidelines in a quantitative manner, and (b) providing a holistic view of the overall performance and operational status of the entire research infrastructure. The evaluation process utilizes metrics to measure the collaboration among teams, documentation completeness, EBRAINS API adoption, component maturity, and responsiveness/readiness in issue resolution.

5.1.1 KPIs

The KPIs are categorized into two main categories: **Administration and Management** and **Development and Operations**. The former category focused on efficient project management and resource allocation while the latter on tracking the progress in development and integration efforts and assessing the operational efficiency and adherence to guidelines. An outline of the KPIs list follows, including sub-categories as presently decided.

- Administration and Management
  - Collaboration indicators: measuring the level of collaboration among teams and evaluating the effectiveness of teamwork towards project goals.
- Development and Operations
  - Documentation: emphasizing the importance of comprehensive documentation.
  - Service Operations: tracking adoptions of EBRAINS APIs and physical deployment and ensuring conformity to integration/delivery guidelines.
  - Component maturity: measuring the number of components using EBRAINS API and monitoring progress in VM-based and container-based services deployment.
  - Responsiveness/Readiness: utilizing TC GitLab metrics and ensuring timely resolution of issues.

Please note that the KPIs are intended for goal setting and intention-signalling and do not reflect, replace, nor overlap with the KPIs officially defined in the framework of SGA3. Further, they should be applied to portray the integration and delivery progress for the entire EBRAINS RI, and not to assess the performance or support the progress analysis of a particular component.

All KPIs are described in detail in Table 6 and Table 7, including specifying which are "point in time", and timespans for those which are "averages". Descriptions also indicate which are extracted manually, semi-automatically, and automatically and the calculation procedures where appropriate.

Table 6: KPIs - Administration and Management

Administration and Management	KPI name	KPI description	Target value at M42
Collaboration indicators	Participation in TC Weekly meetings	3-month average percentage of EBRAINS components with representatives in the TC Weekly meetings. Calculated manually from collected participation information from the meetings' minutes.	>70%
	Wiki pages update response time	3-month average number of weeks component owners take to update the component's wiki page at TC Collab upon receiving request from TC. Calculated manually.	<3 weeks

Table 7: KPIs - Development and Operation

Development and Operations	KPI name	KPI description	Target value at M42
Documentation	Developer Documentation	Point in time KPI. Percentage of components which have Developer documentation available online in a user-friendly format. Calculated manually every 3 months.	>80%
	Maintainer/ Operator Documentation	Point in time KPI. Percentage of components which have Maintainer/Operator documentation available online in a user-friendly format. Calculated manually every 3 months.	>60%
	End-User documentation	Point in time KPI. Percentage of components which have End-User documentation available online in a user-friendly format. Calculated manually every 3 months.	>60%
Service Operations	Deployment to testing	3-month average percentage of time target component-version is successfully deploying/deployable in testing (passed linting, unit-testing, component-e2e-testing, etc). Extracted semi-automatically from the CI system.	-
	Deployment to staging	3-month average percentage of time target component-version is successfully deployed in staging (also passed system-testing, solution-testing, performance/stability-testing, e2e-testing acceptance-testing, etc). Extracted semi-automatically from the CI system.	-
	Deployment to production	Deployment to production: 3-month average percentage of time target component-version is successfully deployed/deployable in production (also passed manual system-e2e-testing, user-acceptance-testing, beta-programme, etc - and packaged for install). Extracted semi-automatically from the CI system.	-
Component maturity	EBRAINS APIs adoption	Point in time KPI. Percentage of components (only where adoption is relevant) that are using the EBRAINS APIs. Calculated every 3 months.	>50%
	VMs Deployment	Point in time KPI. Percentage of VM-deployed components that have achieved physical deployment planning goals out of those prescribed. Used for tracking physical deployment progress for VM-based services. Calculated semi-automatically every 3 months.	>50%

Development and Operations	KPI name	KPI description	Target value at M42
	Containers Deployment	Point in time KPI. Percentage of container-deployed components that have achieved physical deployment planning goals out of those prescribed. Used for tracking physical deployment progress for container-based services. Calculated semi-automatically every 3 months.	>50%
Responsiveness/Readiness	Turnaround of TC Gitlab active issues.	3-month average percentage of issues which have been active more than 3 weeks. Calculated semi-automatically from Gitlab.	<70%
	TC Gitlab active issues	3-month average number of active issues. Calculated semi-automatically from Gitlab.	<75
	Turnaround of TC Gitlab active SC prioritization issues.	3-month average percentage of SC prioritization issues which have been active more than 3 weeks. Calculated semi-automatically from Gitlab.	<70%
	TC Gitlab active SC prioritization issues.	3-month average number of active SC prioritization. Calculated semi-automatically from Gitlab.	<35
	TC Gitlab open issues	3-month average number of open issues. Calculated semi-automatically from Gitlab.	<20
	TC Gitlab backlog issues	3-month average number of backlog issues. Calculated semi-automatically from Gitlab.	<25
	TC Gitlab in-progress issues	3-month average number of in-progress issues. Calculated semi-automatically from Gitlab. (the target values are just an estimation based on the current status)	<30
	TC Gitlab open SC prioritization issues	3-month average number of open SC prioritisation issues. Calculated semi-automatically from Gitlab.	<10
	TC Gitlab backlog SC prioritisation issues	3-month average number of backlog SC prioritisation issues. Calculated semi-automatically from Gitlab.	<10
	TC Gitlab in-progress SC prioritisation issues	3-month average number of in-progress SC prioritisation issues. Calculated semi-automatically from Gitlab. (the target values are just an estimation based on the current status)	<15
	Automated tests	3-month average percentage of components having automated their testing procedures out of those prescribed. Calculated semi-automatically from the CI system.	>60%

In Table 8 and Table 9, as well as in Figure 35 and Figure 36, we present the values of the defined KPIs for all EBRAINS components across all phases from M27 to M42. Each table consists of the main category, the KPI name, and finally five reference months. The chosen months hold equal significance, carefully selected to provide a precise view and state of each component separately.

- M27 represents the midpoint of Phase 3.
- M33 marks the official Phase 3 conclusion. We need to note, however, that Phase 3 activities were unofficially extended until M39 to perform thorough assessment on the integration progress of all the components and allow the proper integration of several new components that were introduced.
- M36 represents the critical and complicated assessment period where the TC team undertook a vital assessment evaluating 73 components against the technical requirements and shared the feedback with the owners.

- M38 provides an accurate estimate of the results right before the conclusion of Phase 4. From M36 to M38, an additional communication campaign has been implemented, with more than 170 messages exchanged between the TC team and the component owners. As measured, this campaign resulted in a substantial 70% decrease in the response time from the component owners.
- M42 marks the conclusion of Phase 4. We expected no major changes from M39 to M42 as many of the components had already been integrated right before the end of M38.

Table 8: KPIs - Administration and Management-All Components M27-M42

Administration and Management	KPI name	M27	M33	M36	M38	M42
Collaboration indicators	Participation in TC Weekly meetings	61.9	62.3	62.3	71.2	75.3
	Wiki pages update response time (weeks)	8.8	8.8	4.4	2.7	2.2

Table 9: KPIs - Development and Operation - All Components Values

Development and Operations	KPI name	M27	M33	M36	M38	M42
Documentation	Developer Documentation	67.2	72.5	75.4	83.6	83.6
	Maintainer/Operator Documentation	25.4	24.6	26.1	30.1	30.1
	End-User documentation	85.1	94.2	95.7	97.3	98.6
Service Operations	Deployment to testing	49.2	60.0	63.1	70.6	75.0
	Deployment to staging	19.0	27.3	31.8	39.1	39.1
	Deployment to production	56.3	76.9	81.5	92.6	92.6
Component maturity	EBRAINS APIs adoption	70.6	82.9	82.9	81.1	81.1
	VMs Deployment	45.5	52.9	55.9	55.6	55.6
	Containers Deployment	63.6	64.7	64.7	69.4	69.4
Responsiveness/ Readiness	Automated tests	53.7	56.5	69.6	80.8	80.8
	Turnaround of TC Gitlab active issues.		65		61	
	TC Gitlab active issues		54		27	
	Turnaround of TC Gitlab active SC prioritization issues.		27		22	

Development and Operations	KPI name	M27	M33	M36	M38	M42
	TC Gitlab active SC prioritization issues.		15		2	
	TC Gitlab open issues		15		7	
	TC Gitlab backlog issues		6		9	
	TC Gitlab in-progress issues		2		4	
	TC Gitlab open SC prioritization issues		4		0	
	TC Gitlab backlog SC prioritisation issues		1		0	
	TC Gitlab in-progress SC prioritisation issues		1		0	

Since the beginning of Phase 3 in M21, the Component Owners Participation in TC Weekly meetings values experienced a 30% decrease compared to M21 (see D5.4). As seen in the visualization of the numbers in the left chart of Figure 35, it becomes evident that not all owners were equally engaged in participating in the event. However, this ratio was expected due to the mass introduction of the 47 components, which brought new tools and owners into the integration journey and therefore resulted in a drop in the overall participation numbers. Upon reaching the end of Phase 3 (M33), our assessment showed that the participation rate remained nearly the same as it was in M27, suggesting the need for improved engagement strategies within the project. The communication campaign in M34 (see 5.3.2) resulted in a 15% increase in the component owner's engagement with the Technical Coordination team and the overall collaboration. As seen in the right chart of Figure 35, from M34 to M40, the TC team measured a remarkable 70% decrease in the Wiki page update response time upon receiving request. In M33, for instance, it took 8.8 weeks for a request to be addressed while in M38 it was only 2.7 weeks. It's important to note that the introduction of that new communication campaign resulted in a substantial improvement of all KPIs values.

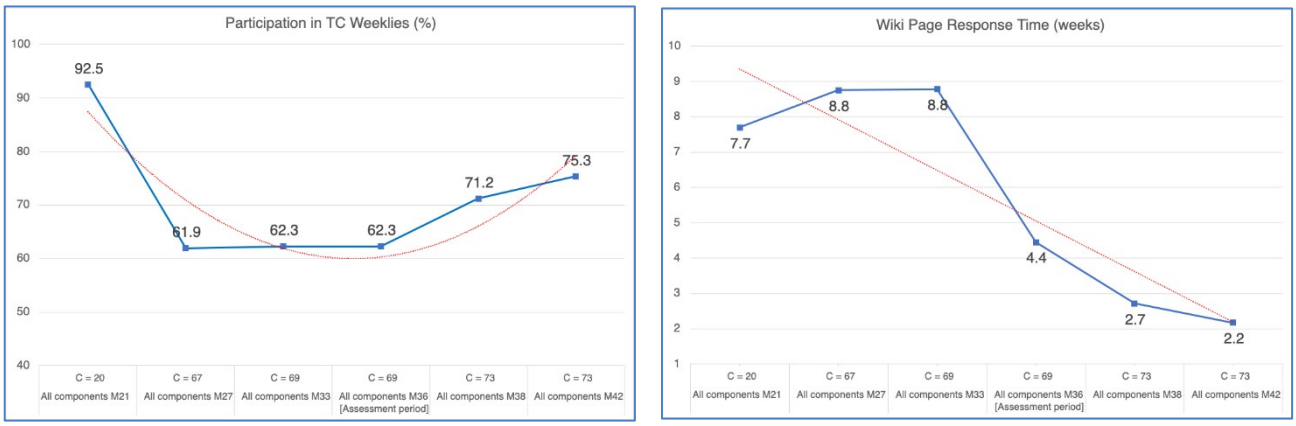

Figure 35: Collaboration: Components progress in reference months

Overall, as seen in Figure 36, roughly all components managed to deliver end-user and developer documentation while one-third of them managed to deliver maintainer and contributor

documentation. It has been determined, however, that most of the end-user documentation includes detailed information for maintainers as well. From M27 to M42, there was a significant increase of 60% in deployments to production. Additionally, it appears that users bypassed the staging environment and proceeded directly from testing to production. Another important insight is that all components managed to deploy their application to either OpenShift OKD, VMs, or both.

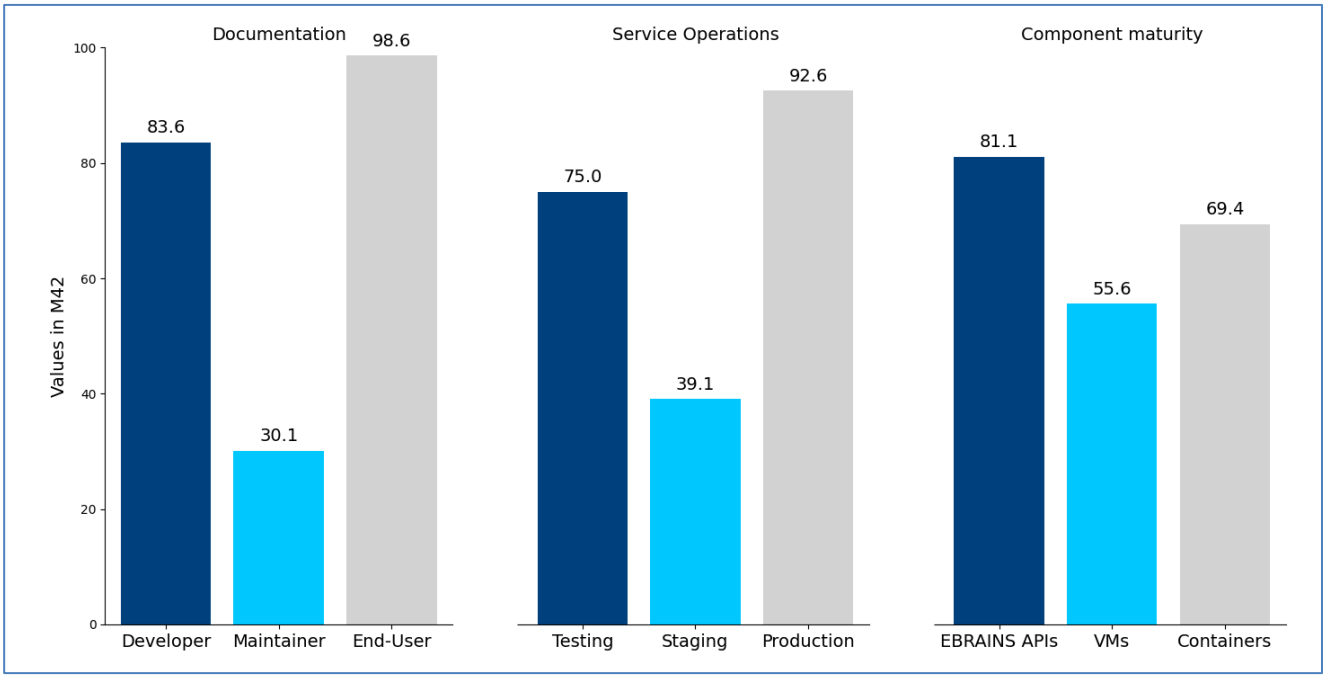

Figure 36: Development and Operations: Components progress in M42

5.1.2 Integration and Collaboration

At the beginning of Phase 4 (M34) and considering our goal for the timely delivery of the final EBRAINS RI, the TC conducted a thorough evaluation of the existing communication and feedback iterations with component owners. The evaluation revealed that since the beginning of Phase 3 (M21), component owners took an average of 8.8 weeks to respond to TC calls, highlighting the need for a more efficient approach. In response, the TC team introduced at the same time: (i) the Technical Integration Score (TIS) and (ii) a revised communication strategy with multiple internal campaigns, spanning from M34 to M40.

As components continued their integration path, the TIS offered a numerical snapshot of their progress – quantifying the extent to which their integration obligations had been met. Importantly, the TIS is determined by calculating a weighted average of all requirements, meaning that it considers the significance or importance of each individual requirement in assessing the overall integration performance. Right after, the TC Communication Strategy was formulated. The TIS score was integrated into the Communication Strategy, adding a different dimension to the process. The strategy included distinct stages such as Stage 1 (M38), Stage 2 (Early M39), Stage 3 (Late M39), Stage 4 (Early M40), Stage 5 (Mid M40), and Stage 6 (Late M40). The strategy aimed to enhance communication with component owners and optimize the integration process.

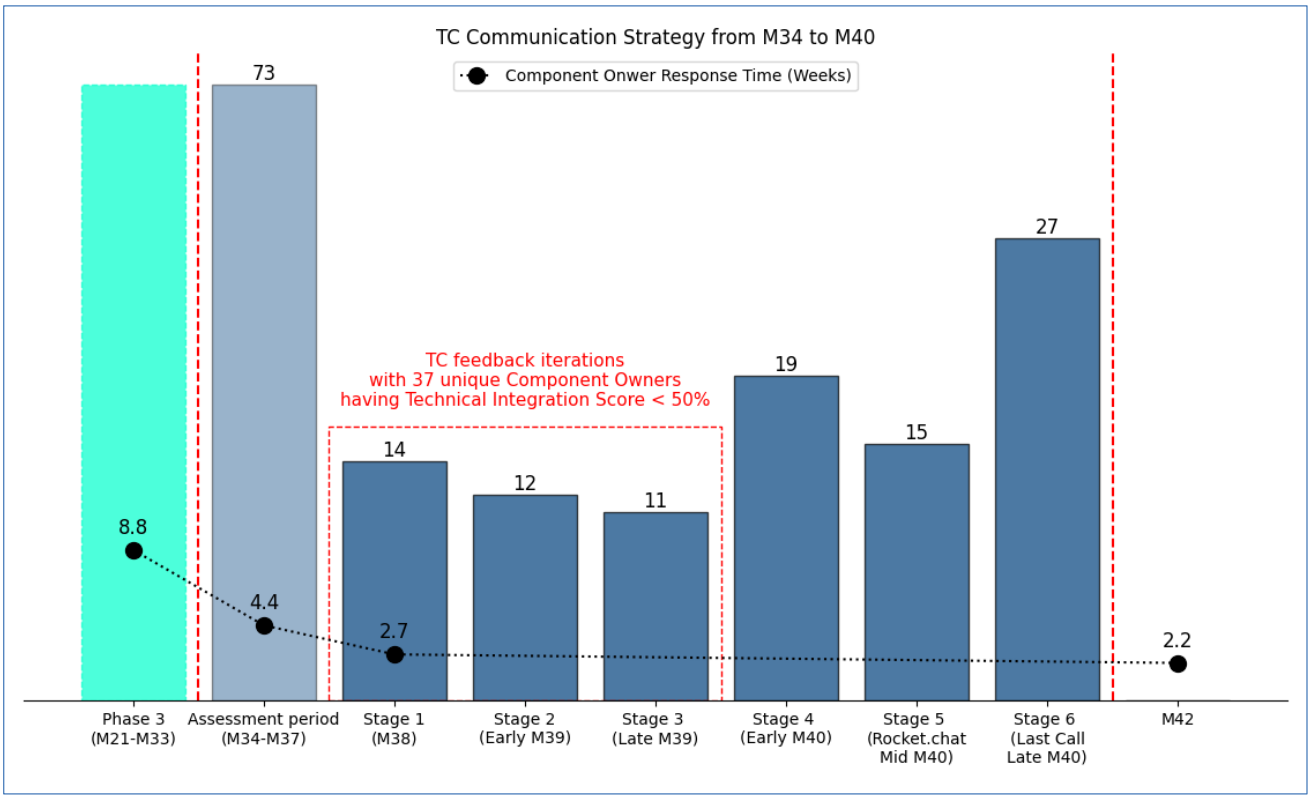

Figure 37: TC Communication Strategy

Figure 37 illustrates the actual implementation of the strategy. It displays information about the number of emails sent during different stages and the component owners' response time upon receiving request from the TC team. The second bar marking the Assessment period, during which all components participated, resulted in the exchange of seventy-three (73) emails. These emails were used to evaluate the integration status of the components involved in earlier phases (20) and to assess the rest fifty-three (53). The red dashed-bordered rectangle groups together certain stages: 1,2, and 3. In these stages, the TC team instantiate one more round of communication with thirty-seven (37) components that had a TIS of less than 50% at that time, during which thirty-seven (37) emails were exchanged, meaning the owners of these components had twice as many requirements to complete compared to the components in the later stages. In Stage 4, the last sixteen (16) components with TIS of more than 50% joined the campaign. Stage 5 marks the integration acceptance stage, indicating that all components had reached an accepted TIS of 80% by this point, where TC continues to engage actively with the owners. As seen in Figure 37, during Stage 4 and 5, thirty-four (34) emails were exchanged between the TC team and the component owners. In Stage 6, the Technical Coordination team continued the feedback iteration process to ensure the completion of the remaining integration requirements. This stage focused on fine tuning and addressing any remaining needs to achieve component level goals, during which twenty-seven (27) emails were exchanged.

In terms of results, the TC Communication Strategy marked positive outcomes. At the end of the assessment period, the response time depicted using dotted markers was measured at 4.5 weeks. At the end of M38 was measured at 2.7 weeks, with the final response time measurement taken at the beginning of M41 being 2.2 weeks which remained stable.

5.2 Phases

The EBRAINS RI has experienced remarkable growth and evolution through its integration journey across four distinct phases during SGA3 (see D5.3, Annex II: TC planning), enabling the successful integration of a diverse range of components. Each phase played a crucial role in forming the basis, validating the integration processes, and gradually scaling up the integration activities. The four established phases (see Figure 38) cover the entire duration of SGA3.

The process of integration was in full motion from day one, involving in total 73 active components, striving to fulfil the horizontal integration requirements that had been formulated by the TC team in order to streamline the integration process. These requirements (see Annex 8.1) handled the integration process at two levels, namely, at the organizational level and at the technical level. At the organizational level, they established two mandatory points of presence in the EBRAINS RI for discoverability. At the technical level, they introduced 13 mandatory points to which all components needed to comply to be considered technically integrated and another set of 8 extra points pertinent to components that were applications/services/Web APIs as far as their type was concerned. In total, 23 integration requirements govern the integration process covering organizational, technical, and security aspects.

Every component went periodically through a careful evaluation by the TC team, to assess the satisfaction of integration requirements and the respective progress. An evaluation report was sent to component owners who had not fulfilled the requirements yet. For the components that had met their obligations, the TC team took a step back, letting things progress naturally. But for those still on path, a new communication strategy was introduced. This approach involved emails that provided support and assurance, ensuring component owners had what they needed and stayed on track.

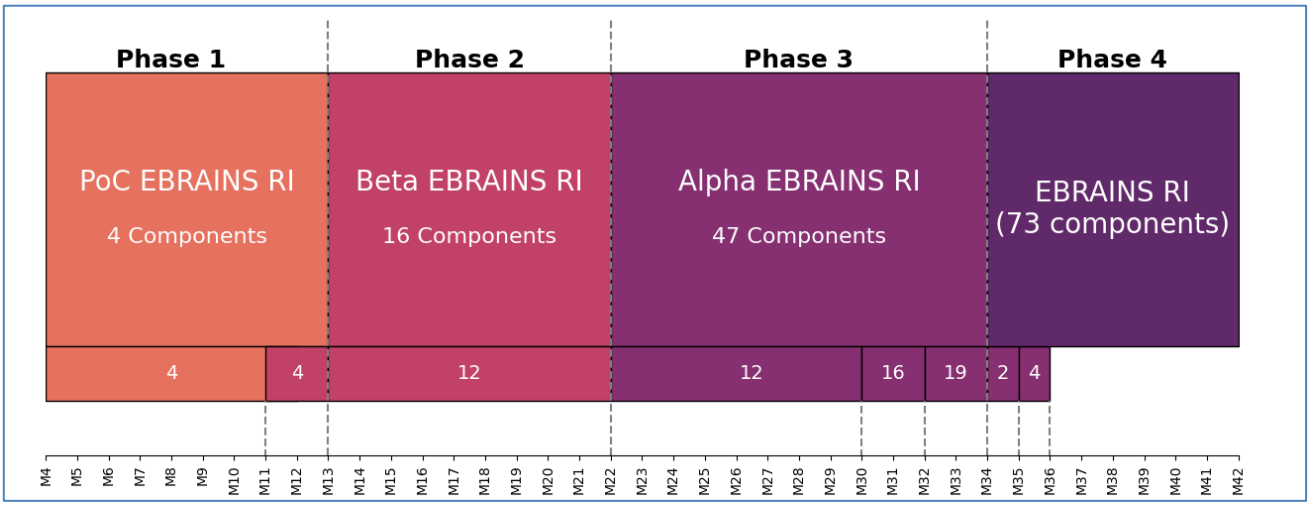

Figure 38: EBRAINS RI Components Integration Timeline

During Phase 1, which spanned from M4 to M12, the Technical Coordination processes were established, and the architectural requirements for each component were introduced. This initial phase set the groundwork for the integration process with a selected number (4) of components participating from the beginning (see D5.3) and for scaling out the activities to all SGA3 components in the next phases.

During Phase 2, which covers the period from M13 to M21, the EBRAINS components participating in the integration process extended significantly. From Phase 1 to Phase 2, the number of components entering the integration process increased by 300%. Almost one-third of the total number of components reviewed by the TC team and delivered tested, documented software adhering to the Software Quality and Delivery guidelines (see D5.3).

Phase 3, spanning from M22 to M33 and with the EBRAINS RI growing, forty-seven (47) components joined the integration journey. To maintain collaboration and feedback iterations with the component owners and ensure the efficiency of the process, the TC team periodically onboarded components in the integration process (see Figure 38).

The final Phase 4, which lasted from M34 to M42, marked the milestone in the integration process and brought the total number of integrated components to seventy-three (73).

While the integration of the components from Phase 1 to Phase 4 has experienced noteworthy progress and success, the transition from one phase to the next revealed drawbacks and delays like interdependencies between components that required additional time and effort, diverse technical issues to ensure compatibility, communication, and collaboration. However, with the thorough assessment of the components before integration, by setting sensible timelines, and through

effective planning, the TC team managed to minimize disruptions and ensure smoother transitions from one phase to another.

In the following sections, a summary of the activities and progress that were performed in each Phase are briefly presented. For Phases 1 and 2 detailed descriptions can be found at both D5.3 and D5.4.

5.2.1 Phase 1 (Duration: M4-M12; Output: MS5.1 Proof of Concept EBRAINS RI)

EBRAINS Phase 1 established the foundation for the intensive integration efforts of SGA3, ultimately leading to the delivery of EBRAINS. Its objectives included: (a) assessing and standardizing the Fenix-RI powered services, such as containerization, for deployment and availability across all sites during Phase 2, (b) establishing TC-related processes and tools for monitoring software quality and delivery, (c) transitioning select software components from project-centered deployment and management mode to an EBRAINS-centered one, (d) evaluating and proposing necessary middleware components, and (e) conducting a series of Proofs of Concept (PoCs) to assess different architectural and operational decisions before their wider adoption in subsequent phases.

5.2.2 Phase 2 (Duration: M13-M21; Output: MS5.2 Beta EBRAINS RI)

In Phase 2, the aim was to integrate nearly 50% of the components listed at that time while ensuring that integration practices were in full compliance with D5.3 EBRAINS Technical Coordination Guidelines. A total of 20 out of the 51 listed EBRAINS components participated in the integration process. It is worth mentioning that four of the Phase 2 components had started their integration effort during Phase 1, while the rest onboarded at the beginning of Phase 2.

A key action item during Phase 2 was the expansion of the EBRAINS stack of services to fully utilize at least two of the Fenix-RI sites (CSCS, JSC). The research was carried out to identify services that could be deployed in more than one site, prepare the onboarding of the remaining three Fenix-RI sites and identify site specific considerations according to the EBRAINS Architecture.

EBRAINS RI Monitoring and Observability services moved forward greatly during Phase 2 and extended to a considerable number of components. The processes that would leverage the operation of these services and would lead to the achievement of EBRAINS observability goals were continuously improved. With the beginning of Phase 3, where the vast majority of the components joined the integration effort, metrics and monitoring information from all the EBRAINS applications, services, and APIs were feeding the EBRAINS Web Analytics and backend monitoring services and thus critical insights on the EBRAINS RI usage and utilization of the resources were produced.

Apart from observability goals, progress in the EBRAINS RI moved forward in several fronts as documented in detail in D5.4. These fronts included the strengthening of the Container Orchestration with OKD as far as the deployment strategy was concerned, the introduction of the “Standardised Workflows”, the extension of the interactive workflows, the introduction of the “EBRAINS tools software stack” for the Collaboratory Lab, progress with the efforts in provenance tracking, and finally, the further refinement of the Component Integration Requirements that guided the integration process.

5.2.3 Phase 3 (Duration: M22-M33; Output: RC EBRAINS RI)

In Phase 3, and considering that the core infrastructure, EBRAINS middleware services, and all the appropriate processes were already in full operation, our goal was to integrate 100% of EBRAINS components into the EBRAINS RI and ensure integration practices’ full conformance with both D5.3 ‘EBRAINS Technical Coordination Guidelines’ and D5.4 ‘Interim EBRAINS Infrastructure Implementation Report’ that revised some aspects of the Technical Coordination Guidelines.

To facilitate a smooth and efficient EBRAINS RI integration, the remaining forty-seven (47) components that had not joined the integration process, were divided into three-time intervals (M21, M30, M32). The selection of these specific time intervals and components list was based on an assessment report, considering the maturity level of each component, its type (e.g., service, library, desktop app), and estimated time and effort required for timely delivery. Twelve (12) components joined at the beginning of Phase 3, to be followed by sixteen (16) at M30, and by adding the remaining nineteen (19) components at M32, the official completion of Phase 3 was marked. This approach allowed the TC team to monitor progress over time and manage collaboration and communication with the component owners. Additionally, at the end of Phase 3, six (6) additional components expressed their interest in joining the integration process. The incorporation of these components took place in months M34 and M36, respectively.

It is worth mentioning that during the initial stages of Phase 3 **the total -eligible for integration- components number skyrocketed from 51 to 73** (7 became inactive at the latest stages of Phase 3 resulting in 67 active components in total at the end of this Phase). Such an increase in the number of components posed a challenge as it was largely unexpected, since the integration process was adequately and repeatedly communicated across the consortium partners from the beginning of the project. Regardless, new components emerged at an advanced stage of SGA3 that needed to be accommodated, facilitated, and ultimately integrated into the EBRAINS RI. These activities were successfully accomplished during Phase3. The impact of this unexpected increase in the number of components on the estimated progress of the integration process is presented in Figure 39. This figure highlights the great effort the TC team had to invest to accommodate the components through the integration path against the initial planning. The venture was successful owing to the robustness and effectiveness of the devised integration process.

Apart from the advancements and challenges in the integration process, progress was achieved in other areas as well. The monitoring and observability services were extended to the majority of the components, the “Standardized Workflows” were further extended, the “EBRAINS tools software stack” was extended and made available to the HPC facilities, and the Component Integration Requirements that guided the integration progress were further improved.

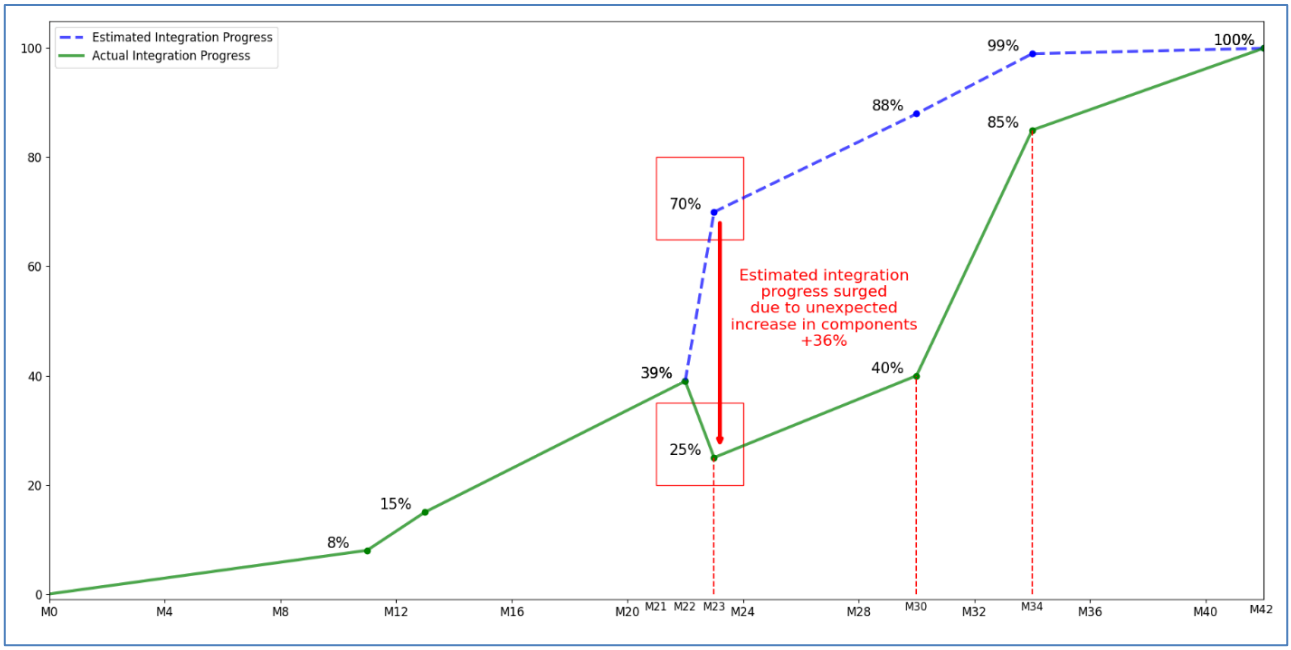

Figure 39: EBRAINS RI Estimated and Actual Integration Progress

5.2.4 Phase 4 (Duration: M34-M42; Output: EBRAINS RI)

In the previous phases, significant efforts were made to lay the foundation, integrate components, and ensure seamless operations within the EBRAINS RI. During Phase 4, the focus was on the delivery

of the EBRAINS RI in full production operation and the handover of its operations to the EBRAINS AISBL. In the initial stages of Phase 4, this entailed the finalization of the architecture, completion of the EBRAINS middleware services, monitoring the integrated components to ensure that the established practices in delivery of services were followed, extending the monitoring and observability services in all integrated components (where applicable), and finally, ensuring the overall Quality of Service of the RI. In the final stages of Phase 4, the efforts focused on the successful handover of the RI's operations to the EBRAINS AISBL, ensuring its continued functionality and sustainability under the governance of the organization.

It needs to be mentioned that the last round of Phase 3 was extended and run in parallel with the initial stages of Phase 4 due to the need to integrate the remaining components and mitigate the unexpected increase in the components number (as detailed in Section 5.2.3). Moreover, as seen in Figure 38, another 6 new components requested to join the integration process at this late stage, and they had to be facilitated as well. This resulted in the total number of integrated components reaching a peak of 73 active components.

The TC team managed to accommodate not only the integration of all known components but also components that emerged later in time, effectively, as well. This demonstrates that the integration process that was devised and coordinated by the TC team was robust, efficient, and streamlined as it successfully brought the different component development efforts under the same roof and homogenized at least the core development practices and outputs for a maintainable and sustainable RI. This outcome was one of the core mandates of the TC team since the beginning of SGA3 and was successfully fulfilled.

During Phase 4, the integration procedure that each new component needs to follow to be considered fully technically integrated in the EBRAINS RI was officially standardized and the individual steps were also formalised. The integration procedure flow is presented in Figure 40. During the “Initial Steps” a standardized document aims to collect basic component information, necessary to assess several key aspects of the candidate component. This document is included in Annex 8.3. The integration requirements were finalized at the end of Phase 4 and are included in Annex 8.1. Finally, to facilitate the assessment of such a vast number of components, an assessment template was drafted based on the finalized integration requirements, to enable the TC team to track the individual component’s progress against every integration requirement relevant to each component. The integration assessment template is included also in Annex 8.3.

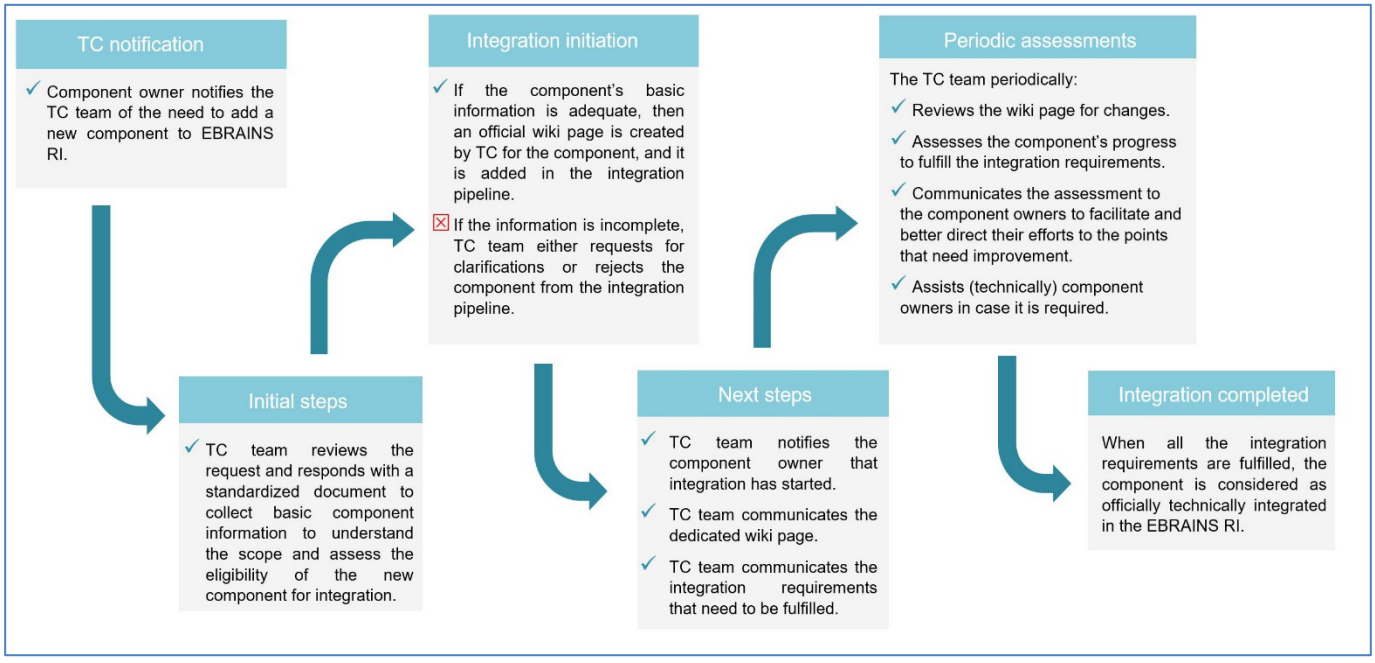

Figure 40: Flowchart of the standardised component integration procedure

5.3 Current status

At the end of SGA3, all seventy-three (73) components have been successfully integrated into the EBRAINS Research Infrastructure, marking a significant milestone in the project's journey. This achievement comes after a challenging integration period from Phase 1 to Phase 4, during which various application types with varying complexity were integrated. This success underscores the dedication of the component owners, who remained committed to the projects' completion. While some components faced initial hurdles, including late-comers and those with limited response to TC calls, the collective effort has led to a stable and fully integrated system.

Moreover, it's worth mentioning that all components have effectively mirrored their source repositories in gitlab.ebrains.eu, ensuring transparency and collaboration in our development efforts. Additionally, every component fulfilled crucial requirements by establishing a presence on the EBRAINS website (ebrains.eu) and securing an entry in the EBRAINS Knowledge Graph.

6. Looking forward

6.1 EBRAINS Architecture White Paper

The evolution of the EBRAINS RI Architecture has been imprinted into a White Paper since M26 and has been revised continuously since, to leverage all lessons learnt in SGA3, provide a foundation and 'true north' for the RI's future, as well as a roadmap for technical developments.

The latest version of the proposed evolution of the EBRAINS Architecture has been presented and discussed in D5.15, as one of its key aspects has been the increase in security and resilience. Further, it has been formally adopted in the future EBRAINS roadmap in the EBRAINS 2.0 proposal (INFRA-SERV), serving as the technological and operational framework for EBRAINS moving forward.

Overall, the proposed EBRAINS architecture aims to repurpose, adapt, revise, and extend the current EBRAINS RI to (a) increase user satisfaction, scalability, and cost-effectiveness, (b) lower operating expenses while guaranteeing a high level of service to its users, (c) ensure prolonged sustainability and multi-faceted FAIRness, (d) address emerging user needs and technology outputs, and (e) flexibly accommodate the business, operational, and synergetic scientific and industrial value co-creation models. Towards this, we leverage lessons learnt from the development of EBRAINS during HBP SGA3 and introduce a series of technical and organizational interventions to proactively address all identified technology, integration, and service provision risks/challenges. Among others, these relate to the (a) decoupling, restructuring, and extension of core infrastructure software components, (b) clarity of the business models, access policies, and governance of EBRAINS, and (c) flexible and cost-effective exploitation of heterogeneous base infrastructure resources.

The reader is invited to review Section 4.1 of Deliverable D5.15 for more details. Next, we provide a summary of key demarcation points compared to the current Architecture of the EBRAINS RI.

6.1.1 Scope

EBRAINS is a federated e-Infrastructure providing data, models, and services to its users directly (EBRAINS AISBL node) or via its National Nodes (federated nodes), harnessing base infrastructure compute and storage resources in a cost-effective, scalable and dependable manner. EBRAINS abstracts and hides complexity related to the deployment, integration, operation, security and availability of its myriad software and hardware resources to provide its users one environment where they can perform science. The key features of EBRAINS are:

- **FAIR by design:** All types of scientific output in EBRAINS are FAIR by design and EOSC-compatible, including data, publications, models, notebooks, workflows, and source code.
- **High-level service provision**: The software, computing, and storage offerings of EBRAINS are managed, deployed, provisioned, maintained, and supported in a guaranteed manner.
- **Multi-level infrastructure**: EBRAINS provides IaaS, PaaS, and SaaS offerings to its users depending on the specific type of software, service, operation, need, knowhow, and SLA.
- **Base infrastructure**: EBRAINS flexibly harnesses computing and storage resources from HPC/exascale (EuroHPC/PRACE, FENIX) and cloud providers (commercial, private, hybrid) in a manner that ensures scalability, high-availability, provider-independence, and a uniform experience for its users.
- **Secure by design**: EBRAINS implements a strict security framework for the handling of sensitive health data, enabling their legal and effective management, sharing, and scientific use.
- **Uniform user experience**: EBRAINS users have access to a single web-based environment enabling them to discover, reuse, learn, create, share, collaborate and publish their scientific output, harnessing all EBRAINS offerings and resources with a single account.
- **Open-ended offerings**: The software and service offerings of EBRAINS are constantly enriched and extended to serve scientists. New offerings (libraries, services, web/desktop applications)

are deployed centrally in EBRAINS under excellence and cost-benefit criteria, while users can independently provide their output to others in the form of workflows and notebooks executed within EBRAINS SaaS offerings.

- **Open by design**: EBRAINS exclusively adopts, extends and provides open-source software, open data-formats, and open APIs. EBRAINS offerings are available well outside the neuroscience community, welcoming cross-disciplinary collaboration, and inviting scientists from all fields to harness its data, computing, and service offerings.
- **Federated services**: EBRAINS services can additionally be provided directly by national competence centers/nodes in a federated manner, serving theme-, country-, or institution-specific research and innovation priorities.
- **Uniform and granular access policy**: Access to EBRAINS is open to all researchers (basic tier) and EBRAINS AISBL-affiliated researchers (advanced tiers), with additional services/resources provided against well-defined excellence-based or commercial criteria.

6.1.2 *Base Infrastructure*

EBRAINS flexibly harnesses computing and storage resources from HPC/exascale (EuroHPC/PRACE, FENIX) and cloud providers (commercial, private, hybrid) in a manner that ensures scalability, high-availability, provider-independence, and a uniform experience for its users. The key considerations, based on the learnings of SGA3, which influenced our design decisions are:

- **Low cost**: The total cost for procuring/purchasing, provisioning, and operating base infrastructure resources should be minimized as much as possible, as it comprises a sizeable cost center, and thus risk, for the sustainable operation of EBRAINS. Where possible and relevant, we favor low-cost and commoditized resources and services, with high scalability characteristics, and exploit them to scale up/down EBRAINS (and its costs) according to user demand, service workloads, utilization, and strategic priorities. Towards this, we will explicitly utilize commercial EU-based cloud offerings, where legally, ethically, and technically sound.
- **Standardization**: In SGA3, the first version of EBRAINS and its base infrastructure layer (FENIX) were developed and matured in parallel, which resulted in unexpected complexity, delays, blocking points, and unmet requirements. The know-how and clarity gained in terms of EBRAINS requirements allow us to define and standardize the technical requirements from the base infrastructure in a manner that covers current and future needs in a cost-effective manner. This also enables EBRAINS to satisfy its user demands, negating the observed mismatch regarding the type/availability of computing resources.
- **Vendor-independence**: The long-term lifecycle of EBRAINS, during which base infrastructure providers, their technology offerings, and associated cost centers may change, require a high-level of flexibility in terms of where EBRAINS is deployed and how efficient the re-deployment and allocation of new base infrastructure resources is. This is expressed as increased operational sovereignty and portability from EBRAINS by minimizing and/or standardizing dependence on external resource providers for core internal functions (e.g., access policies, resource allocation, open/de facto standards).

6.1.3 Services

EBRAINS services to its users are delineated into Platform, Science, and Federation services.

- **Platform:** These are horizontal service offerings provided to all EBRAINS users, offering core access, discovery, management, processing, and collaboration facilities. They include the following: Dashboard (the main entry point for all registered users from which they can search, discover and access EBRAINS services and computing resources), Catalogue (aka Knowledge Graph, enables users to find, discover and deposit scientific output - e.g. data, models, publications, notebooks, workflows), Drive (personal and shared cloud-based file/object storage integrated across platform and science services), Documents (web-based viewer/editor for wikis

and documents), Lab (enables users to edit and execute Jupyter notebooks over a variety of processing environments), Workflows (visual/CLI editor and engine for creating and executing CWL workflows over HPC/cloud resources), Virtual Machines (ad hoc provision of VMs).

- **Science**: These are horizontal service offerings provided to all EBRAINS users, offering access to a variety of loosely integrated scientific applications, services, visualization engines, and processing environments. A science service can be deployed into EBRAINS if it satisfies the minimum integration interoperability requirements (see previous point) and can be available as a: library (e.g. available in Lab notebooks and/or HPC), API (e.g. processing, simulation, visualization), web application (e.g. self-contained thematic application), or a remote desktop/kiosk application (e.g. preconfigured VM with remote access for interactive visualization). The list of current EBRAINS science services is available in the Annex of this document.
- **Federation**: These services are provided independently by the national nodes and competence centers and are discoverable via the Dashboard (authorization and access policies are defined by each node). The minimum technical requirements for a federation service are: (a) EBRAINS AAI integration (authentication services), (b) Service description metadata in the Catalogue (discovery services), (c) automated provision to EBRAINS of utilization statistics/KPIs (monitoring services). Any additional integration with EBRAINS platform/science services is examined on an ad hoc basis depending on the nature of the federation service.

EBRAINS Services are provisioned under well-defined technical, operational, scientific, and sustainability criteria throughout their lifecycle. Specifically:

- **Platform**: The type, nature, features, and evolution of Platform services are (a) defined by the EBRAINS CIO with feedback from the EBRAINS Product Owner/Manager and Ops Teams, and (b) documented in the Platform Roadmap. As these services are integral to the operation of the infrastructure and all other offerings, changes should be kept to a minimum with emphasis on security, scalability, fault tolerance and high availability.
- **Science**: The collection and features of Science services are (a) defined by the EBRAINS CEO with feedback from the EBRAINS Science Panel and EBRAINS CIO, and (b) documented in the Services Roadmap. Since Science services by definition must support state of the art research and maintain the competitive advantage/USP of EBRAINS, they are considered as evolving, with new features and services introduced at a high frequency. For each service, lifecycle decisions (onboarding, features, resource allocation, tiers, sunsetting) consider at least the following criteria: size/significance of user base (current or potential), state-of-the-art (ahead of the curve, uniqueness, scientific excellence), technical maturity & sustainability (size/activity/responsiveness of development team, technology framework/dependencies, licensing), technical conformance & integration (satisfaction of EBRAINS interoperability & integration framework), provision effort/costs (EBRAINS Ops Team, Helpdesk), network effects (added-value of other services, ecosystem), legal framework (private/sensitive data, veto/access exceptions), strategy (EBRAINS AISBL, ESFRI, EOSC, competitors/partners).
- **Federation**: The type, nature, features, and evolution of Federation services are the sole responsibility of the national competence centers/nodes. The requirements set by EBRAINS are kept to a minimum to promote diversified offerings (see previous point on EBRAINS integration and monitoring), but also safeguard the EBRAINS brand (i.e.an actual service with a meaningful functionality and level of support). A federation service may utilize EBRAINS Platform/Science services and/or computing/storage resources (handled by ad hoc requests) or graduate to a science service (i.e. be provisioned/managed by the Ops Team). Conversely, EBRAINS Platform & Science services do not hard-depend on Federation services (which are outside of central control for guaranteeing SLAs), and Federation services depending on other Federation-member's services do so purely at the responsibility and administrative cost of the two member centres.

*The above mentioned **criteria** will be combined with the criteria defined by the **EBRAINS PREP** project[48] (**D2.2** and **D2.5** of EBRAINS PREP).*

6.1.4 Sustainable Resource Allocation

The allocation of EBRAINS resources (e.g., storage, computing, service quota) is managed by the AISBL in accordance with the EBRAINS Access Policies, and fully integrated within the EBRAINS infrastructure in a transparent, fair, pragmatic, cost-effective, and sustainable manner. Scaling up/down to accommodate changes in user demand, strategic priorities, and synergies, is integrated within the architecture and operations of the infrastructure, further enhancing its sustainability. Specifically:

- EBRAINS groups the underlying federated base infrastructure resources and provisions them via resource 'pools' (allocation may dynamically alter between them): platform pool (EBRAINS platform services), science pool (EBRAINS science services), IaaS pool (direct IaaS provision), spare pool (scaling).
- EBRAINS establishes clear access tiers with integrated quota for its users (storage, computing, services). These are: available to all visitors (unregistered), registered access, advanced access (AISBL-affiliated), on demand access/add-ons (permanent temporary additional resources/access), federated access (access to node services), developer access (IaaS/APIs).
- Access authorization, monitoring, and enforcement of policies across all tiers is performed via a dedicated software component of EBRAINS collecting information across the entire infrastructure (from federated base infrastructure up to science services). Users have access to tier information, quota utilization, and means to request additional resources integrated within their Dashboard.
- All EBRAINS Services are designed and provisioned to minimize the static resources consumed and maximize the cost-effective allocation of ephemeral resources, while maintaining a high level of service, and incentivizing revenue-generating opportunities. The application of these principles differs depending on the type and nature of EBRAINS resource requested. What follows is a non-exhaustive list:
    - ***Storage:*** All users are allocated a fixed amount of cloud storage with their subscription tier with quota enforced *across* EBRAINS Platform and Science services. A user may request additional storage (type, duration) via the Dashboard. The request is processed and accepted automatically if it falls within a specific limit (access-policy based and global workload-aware). Requests above this limit are assessed by the EBRAINS Allocation Committee, with all interactions (request, assessment, rebuttal) tracked, communicated, and monitored within EBRAINS. The exception to this process relates to research output (e.g., data, models, software) *deposited* to KG, with storage and human effort required (curation) managed by the EBRAINS Curation team under a given annual budget.
    - ***Notebooks***: Users have access to different base images, processing backends, and quota depending on their tiers. For example, registered users have access to a selection of base images, small/medium computing environments, and processing/keep-alive time for the notebooks. Higher tiers have access to increased base images, environments, and quota. When quota for all tiers are exhausted or additional short-term resources are required, users can similarly request and receive them via their Dashboard (see previous point). The execution of the notebooks is opaque to the users (unless required for data locality reasons), with EBRAINS load-balancing effort across its federated infrastructure. For notebooks serving as entry points to exascale computing resources, the quota for these applies (see below).
    - ***CWL***: Users can discover, create, and execute CWL workflows via the CLI and web editor, across numerous processing back-ends selected in an opaque manner and automatically from the EBRAINS infrastructure. The request is processed and accepted automatically if it falls

[48] https://cordis.europa.eu/project/id/101079717

within a specific limit (access-policy based and global workload-aware; see previous points). Quotas for the execution of CWL workflows consider the processing time, I/O, and storage (interim), and are expressed as credits, with higher-tier users enjoying more credits. The execution of CWL workflows is handled by the workflow engine using a combination of priority queues, thus selecting the appropriate processing environment and response depending on the user's tier. For workflows being executed over exascale computing resources, the quota for these applies (see below).

- ***Exascale computing***: EBRAINS manages and allocates exascale computing resources via *service accounts* provided by HPC sites; these non-dedicated resources serve the exascale computing needs of EBRAINS users via Platform and Science services in a *shared* manner. When *dedicated* exascale computing resources are required, users interact with the HPC site directly (process mediated by EBRAINS) to request and receive personal allocation, which they can consume individually. All users have access to exascale resources, with higher tiers having access to higher quota and/or higher priority processing queues.
- ***Virtual Desktop/kiosk***: Users can access specific EBRAINS Services virtual desktop sessions resembling access to a dedicated desktop environment, over multiple different backends (*e.g., memory size, number of cores, attached GPUs*). The type of EBRAINS applications, the available backends, duration per session, total access duration (e.g., per month), and priority queue (*i.e. how long till the session starts*) depends on the tier of the user, with higher tier users generally having access to more and readily available resources.
- **IaaS**: Users can request dedicated IaaS resources for a given period directly within their Dashboard. The request is processed and accepted automatically if it falls within a specific limit (duration, type/size); otherwise, it is assessed by the EBRAINS Allocation Committee, with interactions (request, assessment, rebuttal) tracked, communicated, and monitored within EBRAINS.
- **EBRAINS Science Services**: The number of EBRAINS Science services a user can access depends on her tier, with higher tier users generally having access to more services (e.g. early access, experimental, specialized). Depending on the type/nature of the application and its respective workload, users consume quota available to them from all other categories (e.g. storage, exascale computing).

6.1.5 Operations

The operations of EBRAINS are handled by a single dedicated cross-disciplinary Ops Team, in charge of all aspects related to the continued, uninterrupted, and high-availability operation of EBRAINS. Members of the Ops Team may be affiliated with the EBRAINS AISBL, EBRAINS AISBL members, competence centres, ad hoc project partners, or even external organizations and sub-contractors. Regardless of their affiliation their assigned responsibilities as members of the Ops Team are managed, directed, and overseen exclusively by the EBRAINS CIO.

In more detail, the responsibilities of the Ops Team include at least the following:

- Deploy, maintain, and manage EBRAINS over the available base infrastructure, allocating computing and storage resources in accordance with EBRAINS requirements and access policies.
- Design, implement, enforce, and manage policies, instruments and interventions related to EBRAINS deployment, availability, monitoring, redundancy, and security.
- Design, develop/create, maintain, deprecate, and sunset EBRAINS platform software components.
- Onboard/offboard EBRAINS services according to the established integration, interoperability, and business policies.
- Provide technical support and guidance to national nodes/competence centers regarding the satisfaction of the EBRAINS federation technical guidelines and requirements.

- Assemble, assess, and provide to EBRAINS AISBL quantitative information regarding the utilization, availability, and SLA performance of EBRAINS.

The Ops Team is not responsible for:

- Developing new EBRAINS services and offerings.
- Providing training services and first-level support to EBRAINS users.
- Managing the operation of federated EBRAINS services.

6.2 Migration post-SGA3

The currently allocated Fenix-provided SGA3 base infrastructure resources will not be available after the end of SGA3, and specifically by the end of 2023. This gap in the availability of base infrastructure resources has been a direct consequence of the rejection of the initial INFRA-SERV proposal submitted by EBRAINS (Feb 2023) and had been methodically mitigated by the EBRAINS leadership since to ensure the sustained and uninterrupted operation of the RI. Towards this:

- JSC and CINECA have committed to providing base infrastructure resources for the operation of the EBRAINS RI, located under the JSC Cloud and ADA Cloud (February 2023, June 2023 respectively).
- The WP4, WP5, and WP6 leadership has initiated the preparation of a comprehensive and complete migration of the EBRAINS RI to these available resources (April 2023).
- A Migration Task Force (MTF) under the leadership of the Athena RC (TC), with the participation of AISBL, JSC, and CINECA, has been formed (May 2023), engaging all EBRAINS stakeholders, and tasked with preparing, planning, supervising, and performing all required actions to ensure a successful migration with negligible impact to the operation of the RI and the satisfaction of its users.

It is important to emphasize that the migration process consists of different stages, most of which have already been implemented. The actual migration of the EBRAINS RI is scheduled to occur after the upcoming Final Review of SGA3 and before the end of the year. This timing has been chosen to ensure minimal disruption to any pending SGA3 activities. However, COs are encouraged to deploy (not migrate) their components as soon as possible.

The migration process is underway, with implemented and planned actions detailed next.

6.2.1 Scope and Responsibilities

The migration encompasses the EBRAINS RI in its entirety, from back-office systems and science services, up to user files. The actual migration will be performed as follows:

- Component owners (COs) are responsible for the migration of their components, except for components that only consist of software libraries packaged under Spack. They are required to prepare and make available for auditing all required deployment pipelines/scripts.
- TC will establish the migration plan and supervise the actual migration, providing technical assistance. Further, it will ensure the readiness of all base infrastructure resources, allocate them to COs and facilitate them on a technical level if needed. In case a CO is not available to perform the migration, this effort will be handled by the TC.

The migration has also accelerated the plans presented in this report (see Sections 2.6, 6.2 and D5.15) towards streamlining the operational footprint of the EBRAINS RI, increasing its security, and overall increasing its technical maturity. Towards this, several of the proposed future revisions and interventions are implemented as part of the migration effort (e.g. central operations, sustainable resource allocation, strict horizontal enforcement of security practices, move towards plain Kubernetes as a PaaS framework).

6.2.2 Base infrastructure resources

JSC and CINECA sites have already provided dedicated Cloud resources, including Virtual Machines and Archival Object storage.

- **Available Resources:** JSC offers cloud resources through the JSC Cloud[49], which is based on OpenStack v6.2.0 and includes support for the Octavia service (load balancer) and Swift Object storage. Table 10 presents a breakdown of the resource types, and their respective upper limits that JSC can offer at the time of this writing[50]. Additionally, it illustrates the currently allocated and in-use resources.

Table 10: JSC resource types

Resource	Upper Limit	Currently in Use	Currently Allocated
• *Virtual CPUs (vCPUs)*	3000	176	768
• *RAM (in TB)*	4.5	0.536	1
• *Storage (in TB)*	100	5.3	20
• *Floating Ips*	50	11	20
• *Swift Object storage (in TB)*	100	0	100

ADA cloud offers cloud resources through OpenStack Wallaby. Table 11 presents a breakdown of the resource types, and their respective upper limits that CINECA can offer.

Table 11: CINECA resource types

Resource	Upper Limit	Currently in Use	Currently Allocated
• *Virtual CPUs (vCPUs)*	384	-	-
• *RAM (in TB)*	1000	-	-
• *Storage (in TB)*	36	-	-
• *Floating Ips*	10	-	-
• *Swift Object storage*	750	-	-

- **Sizing**: The MTF requested specific base infrastructure requirements from COs for the deployment of their components, including size and type of resources. To achieve an equitable distribution of resources, a process of rationalizing the required base infrastructure resources for the operation of every component was implemented. By doing so, the MTF established a fair allocation system, aiming to improve resource utilization and ensure that the available resources could effectively meet the needs of all components.
- **Technical Assessment and Revisions**: The MTF began by evaluating the new JSC and ADA cloud resources to identify differences from our previous ones in Fenix ICEI. This preparatory work aimed to minimize the burden and potential obstacles for component owners during the migration. To streamline the migration process further, the MTF established direct and effective collaboration channels with the JSC Cloud and ADA Cloud teams, feeding them with configuration and operational requirements, which were effectively and timely resolved.
- **PaaS environment**: The OKD environment currently embedded into the EBRAINS RI will be replaced by a plain Kubernetes PaaS environment (with Rancher as a management tool). While our intention was to minimize any changes in the underlying environments, the deployment of OKD over the IaaS was highly problematic, introducing operational overhead in the future, and thus repeating our technical debt. Towards this, we accelerated the move to a CNCF[51]-centric cloud environment, providing expert assistance to all COs for the transition. We anticipate that

[49] JSC cloud differs from the JUSUF cloud which is scheduled to be decommissioned by Q3/2023
[50] The JSC Cloud is expected to further scale out in the near future, adequately addressing future EBRAINS RI needs
[51] https://www.cncf.io/

the overall effort as a result from this change will be *smaller* than a migration to OKD4 (from the current non-maintained OKD 3). Further, it provides increased flexibility and efficiency in cluster management and allocation. Rancher introduces the concept of projects, which are objects designed to allow administrators to manage multiple namespaces as a group and perform Kubernetes operations. EBRAINS COs may be assigned to more than one project, should they wish to do so, indicating a many-to-many relationship rather than a one-to-one association. Within each project there will be one or more namespaces and it is possible for two or more components from the same EBRAINS CO to reside on the same namespace, should this be necessary due to network interconnection of different services and / or administrative reasons. The creation of Rancher projects and their associated namespaces is handled by the MTF.

- **Central Resource Management and Allocation**: The MTF embraced the proposal elaborated in D5.15 towards the deployment of EBRAINS RI components under one project, rather than tens of individual ones. This is an important step towards the reduction of the RI's operational footprint, the efficient utilization of available base infrastructure resources, as well as the increase in security, resiliency, and availability. In this new operational model, COs are provided with resources from one common EBRAINS project directly by the MTF.

6.2.3 Planning and Preparation

The migration process for EBRAINS services and tools involves a series of concurrent tasks designed to ensure a smooth transition, which are presented next.

- **CO Engagement and Communication**: MTF created a dedicated Collab which includes:
  - A publicly available FAQ document, serving as a reference and single point of truth for the migration scope, plan, and requirements for all EBRAINS COs.
  - A comprehensive list of action items for COs, focusing on gathering all specific base infrastructure requirements for their components, including size and type of resources, network specifications, security measures, backup procedures, hardware configurations, and container-related needs.
- **Auditing and Rationalization**: The base infrastructure requirements from COs are assessed and rationalized by the MTF to ensure the optimal use of available resources, successful migration, security, and resiliency. Manifold interventions are being performed under this perspective, which overall further increase the maturity and sustainability of the EBRAINS RI. In addition, the automated deployment scripts/pipelines and comprehensive documentation for (any) manual processes provided by COs are audited (completeness, security, reproducibility, maturity) and improved by the MTF.
- **Infrastructure-as-Code (IaC)**: The MTF has utilized open-source IaC tools (Terraform) to create templates for defining and provisioning VM resources through declarative configuration files. This cloud-agnostic approach ensures the adaptability of these templates across various cloud providers. A set of VMs is provided to different COs, enabling them to deploy their own services effectively, securely, and in a reproducible manner.
- **PaaS Allocation**: The MTF is responsible for creating Rancher projects and namespaces, streamlining the migration process for services and tools. This intricate migration process is guided by lessons learned, aiming to ensure minimal disruption for users and COs.
- **Deployment:** The MTF is actively engaging with COs to facilitate the deployment of their services within the new infrastructure. While the actual migration is scheduled to take place after the Review, COs have already been allocated resources, performed test deployments, with some already successfully deploying their services.

7. References

[1] D5.3- EBRAINS Technical Coordination Guidelines - Human Brain Project (HBP) Specific Grant Agreement 3 (SGA3)

[2] D5.4- Interim EBRAINS Infrastructure Implementation Report - Human Brain Project (HBP) Specific Grant Agreement 3 (SGA3)

[3] The C4 model for visualising software architecture - https://c4model.com/

[4] https://en.wikipedia.org/wiki/C4_model

[5] EBRAINS API Catalogue - https://wiki.ebrains.eu/bin/view/Collabs/api-catalogue

[6] EBRAINS Software Stack Collab - https://wiki.ebrains.eu/bin/view/Collabs/ebrains-software-stack/

[7] D5.14 - Guidelines for security procedures and validation in EBRAINS - Human Brain Project (HBP) Specific Grant Agreement 3 (SGA3) (Confidential)

[8] D5.15 - Report on Infrastructure Security activities - Human Brain Project (HBP) Specific Grant Agreement 3 (SGA3) (Confidential)

[9] Goldman JS, Kusch L, Aquilue D, Yalçınkaya BH, Depannemaecker D, Ancourt K, Nghiem T-AE, Jirsa V and Destexhe A (2023) A comprehensive neural simulation of slow-wave sleep and highly responsive wakefulness dynamics. Front. Comput. Neurosci. 16:1058957. doi: 10.3389/fncom.2022.1058957, PMID: 36714530; PMCID: PMC9880280.

[10] Lee, Kwangjun & Dora, Shirin & Mejias, Jorge & Bohte, Sander & Pennartz, Cyriel. (2023). Predictive coding with spiking neurons and feedforward gist signalling. 10.1101/2023.04.03.535317

[11] Weidler T., Goebel R., Senden M., AngoraPy: A Python Toolkit For Modelling Anthropomorphic Goal-Driven Sensorimotor Systems, bioRxiv 2023.10.05.560998; doi: https://doi.org/10.1101/2023.10.05.560998 (This article is a preprint and has not been certified by peer review)

[12] Gutzen, R., De Bonis, G., De Luca, C., Pastorelli, E., Capone, C., Allegra Mascaro, A. L., Resta, F., Manasanch, A., Pavone, F. S., Sanchez-Vives, M. V., Mattia, M., Grün, S., Paolucci, P. S., & Denker, M. (2022). Comparing apples to apples—Using a modular and adaptable analysis pipeline to compare slow cerebral rhythms across heterogeneous datasets. arXiv:2211.08527. https://doi.org/10.48550/arXiv.2211.08527

[13] Capone, C., De Luca, C., De Bonis, G. et al. Simulations approaching data: cortical slow waves in inferred models of the whole hemisphere of mouse. Commun Biol 6, 266 (2023). https://doi.org/10.1038/s42003-023-04580-0

[14] Robot Framework - https://robotframework.org/

[15] D6.4 (D60 - SGA3 M42) Final release of the federated HPC, Cloud and storage infrastructure for EBRAINS

[16] D6.7 (D113 - SGA3 M42) Health Data Cloud

[17] D4.16 (D47 - SGA3 M42) Release of MIP 8.0

[18] D4.13 (D44 - SGA3 M42) EBRAINS Medical brain activity data platform: human intracerebral EEG database and analysis service (SC5) – status at M36

[19] Ritchie, Felix. (2017). The 'Five Safes': a framework for planning, designing and evaluating data access solutions. Data for Policy 2017: Government by Algorithm? (Data for Policy), London. Zenodo. https://doi.org/10.5281/zenodo.897821

8. Annexes

In this section, we collate updates and links to documents that can offer additional insights and details to the reader of this deliverable.

8.1 Component integration requirements

To facilitate the integration process in a standardised manner and be able to track components progress in a uniform way, we opted to provide the component teams with a list of specific requirements that need to be fulfilled, to streamline the integration activities and centre them around common action items. These action items are commonly referred to as "Component integration requirements". In essence, these integration requirements concretize the "EBRAINS Technical Coordination Guidelines" as presented in D5.3, evolved in D5.4 and finalized during EBRAINS Phases 3 &4. The set of integration requirements is a densely formatted list that can be easily referenced by the component teams without much overhead. The integration requirements were formatted in two distinct levels. The organizational and the technical one. Since Phase 2 the technical level's structure was formatted to two levels, namely, "Baseline" and "For applications/services". This approach was considered effective as it provided one category (Baseline) with mandatory, horizontal requirements that needed to be met by all components regardless of the type of offerings they feature to the EBRAINS RI, and a second category (For applications/services) with requirements concerning accounting, observability, and backup-restore operations that needed to be met in case a component offers some type of "as a Service" functionality to the EBRAINS RI, instead of just a downloadable executable or a package or some other type of static (downloadable) resource. It worths mentioning with respect to the requirements' list, that the ordering is random and does not imply that one requirement is more important than others.

The finalized integration requirements that governed EBRAINS Phase 4 of integration can be retrieved from the following link: https://drive.ebrains.eu/f/9c93540f920f4597bc13/

8.2 Monitoring Service

8.2.1 Index Vs Data Stream

Non-time series data is stored in indices. An index in Elasticsearch is a collection of documents that have somewhat similar characteristics, equivalent to a table in databases. It provides a way to store, search, and analyze data. Each index is identified by a name, which is used to refer to the index when performing various operations. We store all data categories, except for Metricbeat data, in indices since they are not captured and stored in real time. Instead, they are pulled from available sources at predefined time intervals.

On the other hand, time-series data, which includes sequentially organized data points collected over time, is stored in data streams. A data stream in Elasticsearch is a group of hidden, auto-generated indices that can efficiently store append-only time series data. This process simplifies the management of time series data by automating the management of the indices and shard generations, thereby helping to improve indexing and search performance. System OS metrics, collected by Metricbeat agents, are stored in data streams since these data are collected and sent to Elasticsearch every couple of seconds.

8.2.2 Component Onboarding Process

In this section we briefly outline the initial onboarding process required for a component to be registered for monitoring.

- The first step involves creating a dedicated Kibana space named after the respective component. This space serves as an isolated work area within Kibana where all the pertinent data and customizations for the component are stored.
- The next phase involves setting up the necessary infrastructure for data collection and storage, which includes creating the appropriate data streams and indices for the component.
- To ensure the relevant data is conveniently accessible and visualizable, corresponding data views are established for the data streams and indices.
- Lastly, to maintain effective control over access to the data, specific roles associated with these spaces, data streams, and indices are created. These roles are then assigned to the respective component owners, allowing them to interact with the data pertinent to their components.

8.2.3 ILM And SLM Polices

As part of data management and to ensure optimal disk capacity utilization, we utilize distinct strategies for data retention and backup. Specifically, we employ Index Lifecycle Management (ILM) to manage our data retention policy, which retains data for only the last two months. Prior to any data deletion under the ILM policy, a Snapshot Lifecycle Management (SLM) policy is triggered. This SLM policy ensures that a snapshot is taken to securely back up everything, safeguarding our historical records. Consequently, while older data, possibly less pertinent to current analysis, is systematically removed via ILM, it is always preserved in our backups thanks to the SLM process. This approach allows us to maintain a balance between the availability of historical data and the restrictions of disk capacity.

8.2.4 Component Owner Level Dashboards

Back-End Monitoring Level Dashboard (Figure 41): It serves as a comprehensive information centre for the underlying virtual machines supporting the component. This dashboard provides insights into the performance and health of the VMs, such as the number of hosts, cores, disk space, and memory allocated to each host. It also details essential system-level information, including memory usage and CPU usage, which are critical performance metrics for any back-end system. Additionally, the dashboard presents details about various processes running on the hosts, which aids in identifying resource-intensive processes. Notably, for components comprised of multiple hosts, the dashboard offers the flexibility for component owners to specify the host they're interested in, which is something that enables them to analyze the performance and resource utilization of individual hosts.

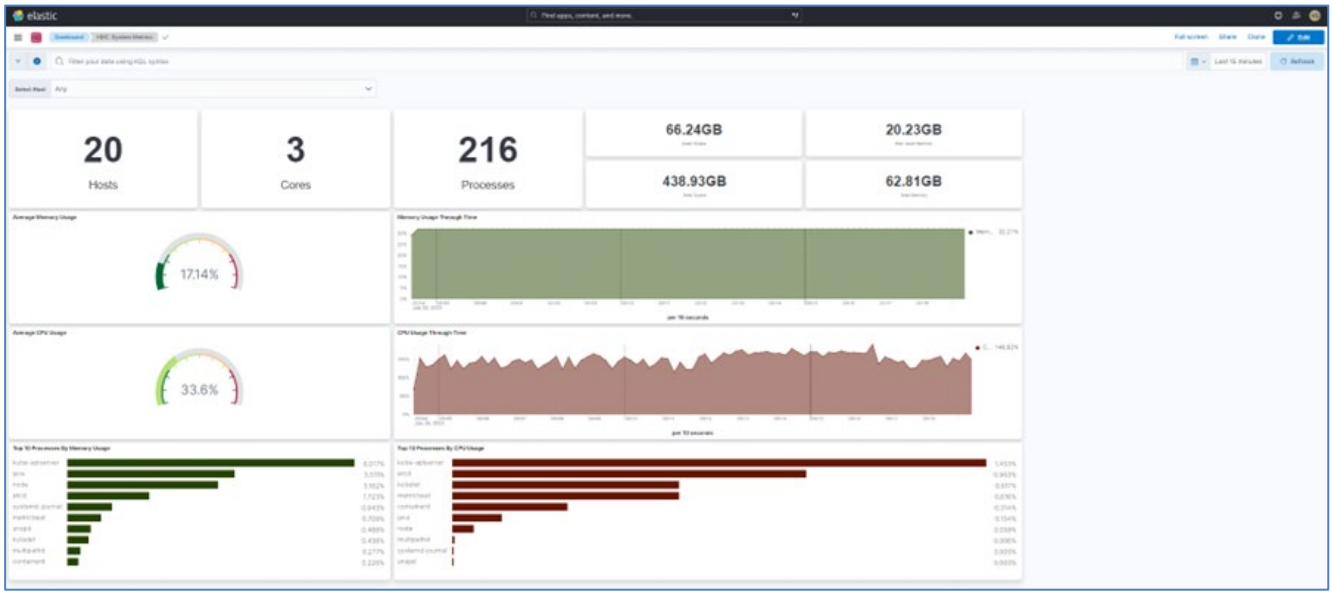

Figure 41: Back-End Metrics Dashboard

Front-End Monitoring Level Dashboard (Figure 42): It serves as a critical resource for understanding user interactions with the component's webpage. This dashboard, which essentially functions as a specialized subset of EBRAINS Matomo offers detailed insights into visits and visitors to the component's webpage. This information allows component owners to track user engagement, identify usage patterns, and understand visitor demographics. Metrics such as the number of visits, duration of visits, geographic location of visitors, and devices and browsers used to access the webpage offer valuable feedback on the component's accessibility and user experience.

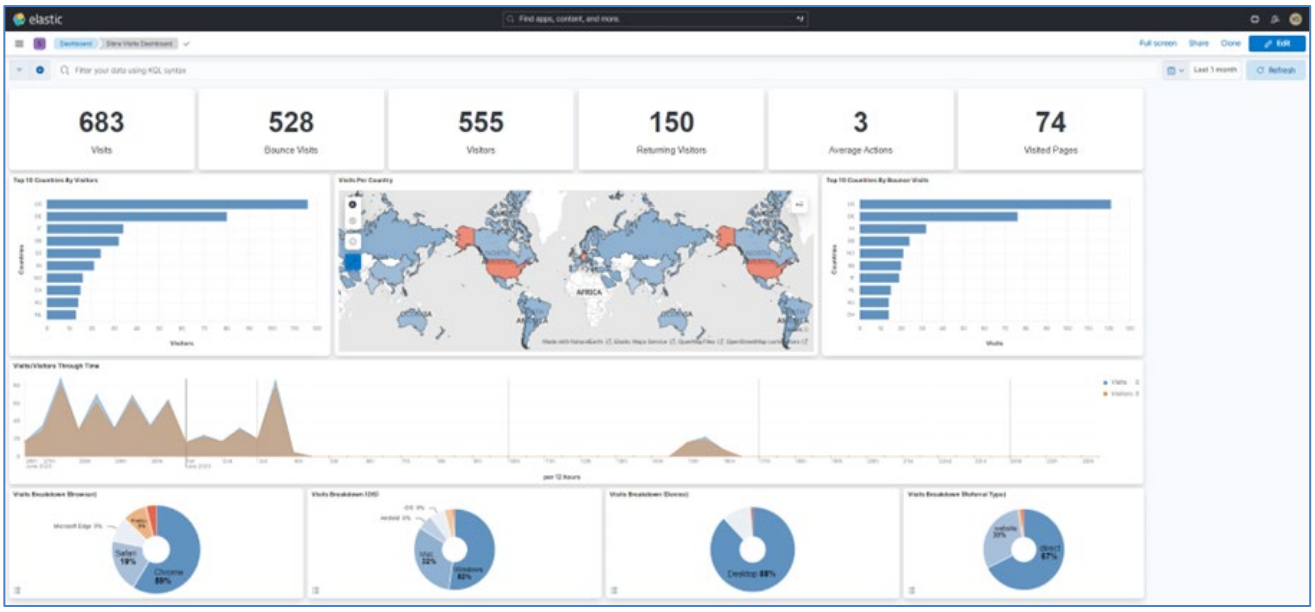

Figure 42: Front-End Visits Dashboard

Application Performance Monitoring Level Dashboard (Figure 43): It differs significantly from the previous two Monitoring Levels in that it is not standardized but customized according to each component owner's unique requirements. This level of customization is necessary because the types of logs and fields depend heavily on the specific needs of the component owner. Therefore, this dashboard is tailored to the component, providing specific insights into application-level performance. Whether it's tracking error rates, understanding execution times, or analyzing user transactions, this dashboard focuses on the metrics that matter most to the specific component. By offering a more granular, tailored view of the component's performance, it aids component owners in pinpointing problem areas, optimizing resource usage, and improving overall application performance.

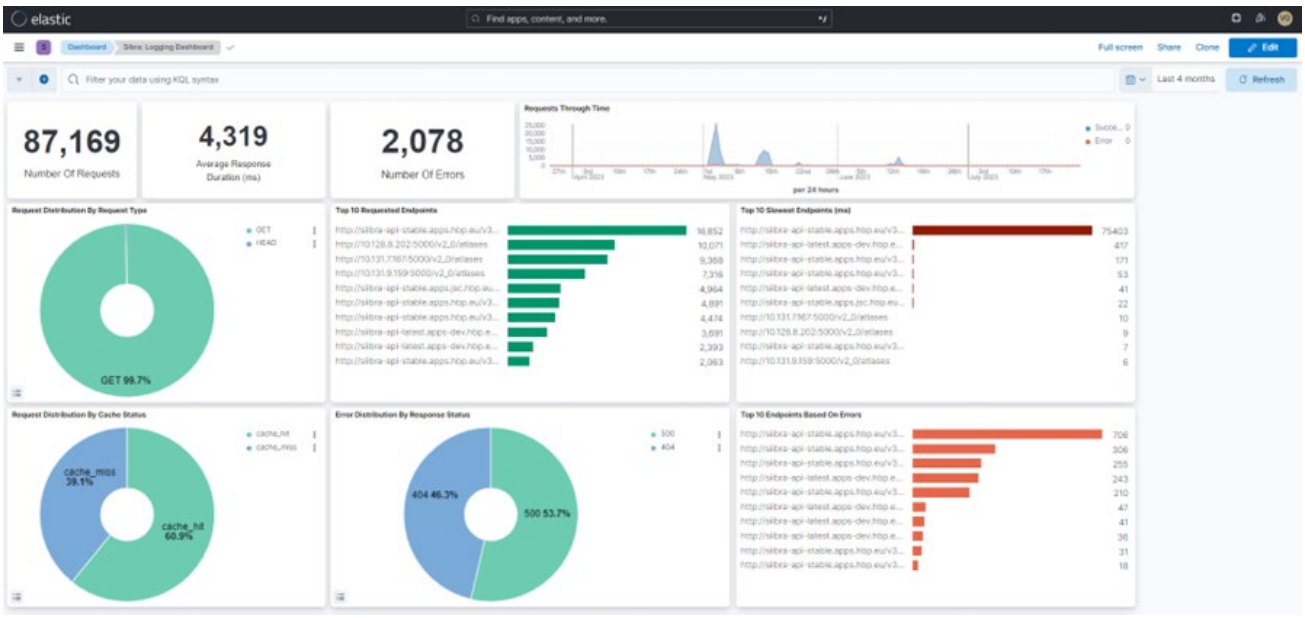

Figure 43: Application Performance Dashboard

Development Operations Monitoring Level Dashboard (Figure 44): This dashboard gives a concise view of Continuous Integration (CI) job metrics. With key details like the total executed jobs, average duration, and status distribution, component owners can quickly gauge their pipeline's health. While it doesn't provide the detailed CI/CD analytics of GitLab, where component owners can delve into specifics about their repository's jobs, it seeks to enhance the "all in one place" capability central to the monitoring service.

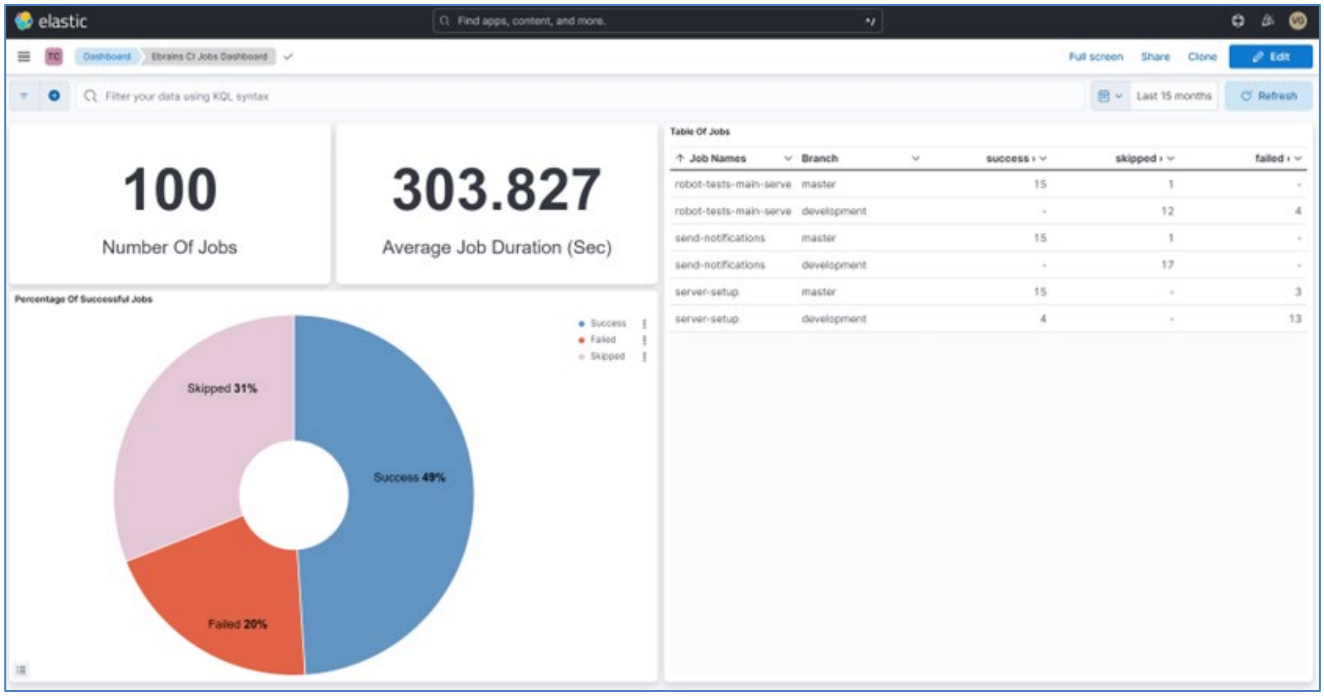

Figure 44: Development Operations Dashboard

Overall System Performance Dashboard (Figure 45): It offers a high-level overview of the system's performance by presenting relevant metrics for every component registered in the monitoring service. This dashboard, like the Back-End Monitoring Level Dashboard, includes information about each component's VMs, such as the number of hosts, cores, disk space, memory, basic system information like CPU and memory usage, and data about processes. However, it is more simplified to facilitate a quick review of system-wide performance.

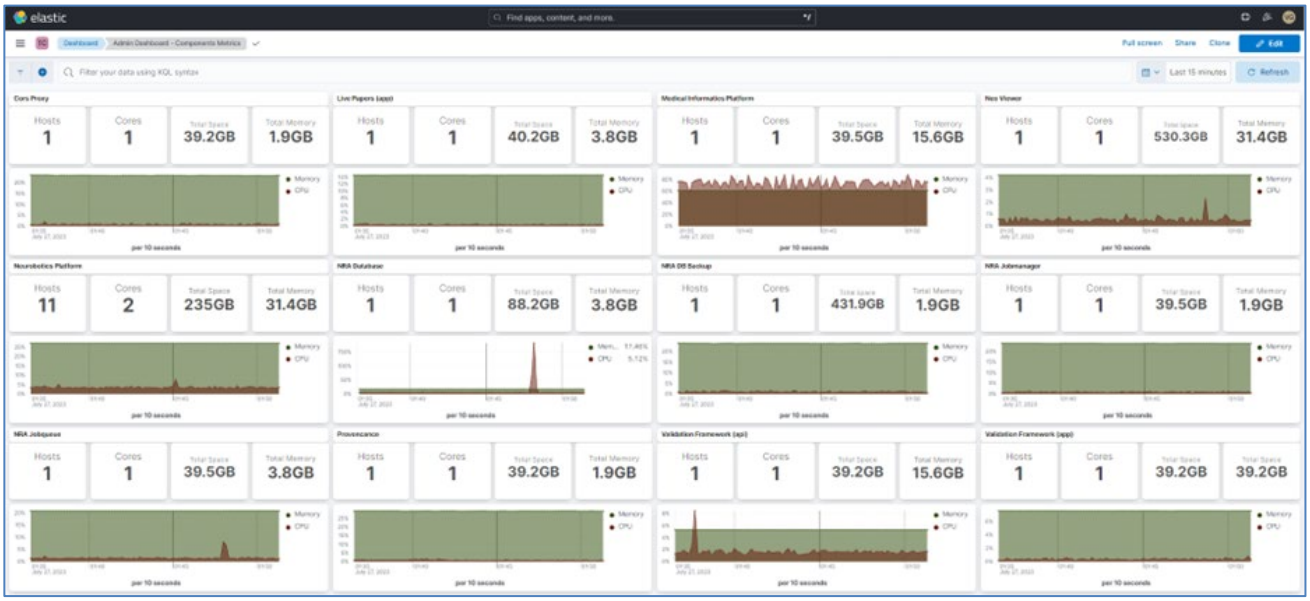

Figure 45: Overall System Performance Dashboard

EBRAINS Website Visits Dashboard (Figure 46): It mirrors the functionality of the Front-End Monitoring Level Dashboard but is specifically dedicated to monitoring the EBRAINS website. It offers detailed insights into the visitor traffic, profiling the activity and engagement on the website. By

tracking metrics such as visits, unique visitors, page views, and other key visitor statistics, this dashboard provides a comprehensive understanding of the EBRAINS website's performance and user interaction patterns.

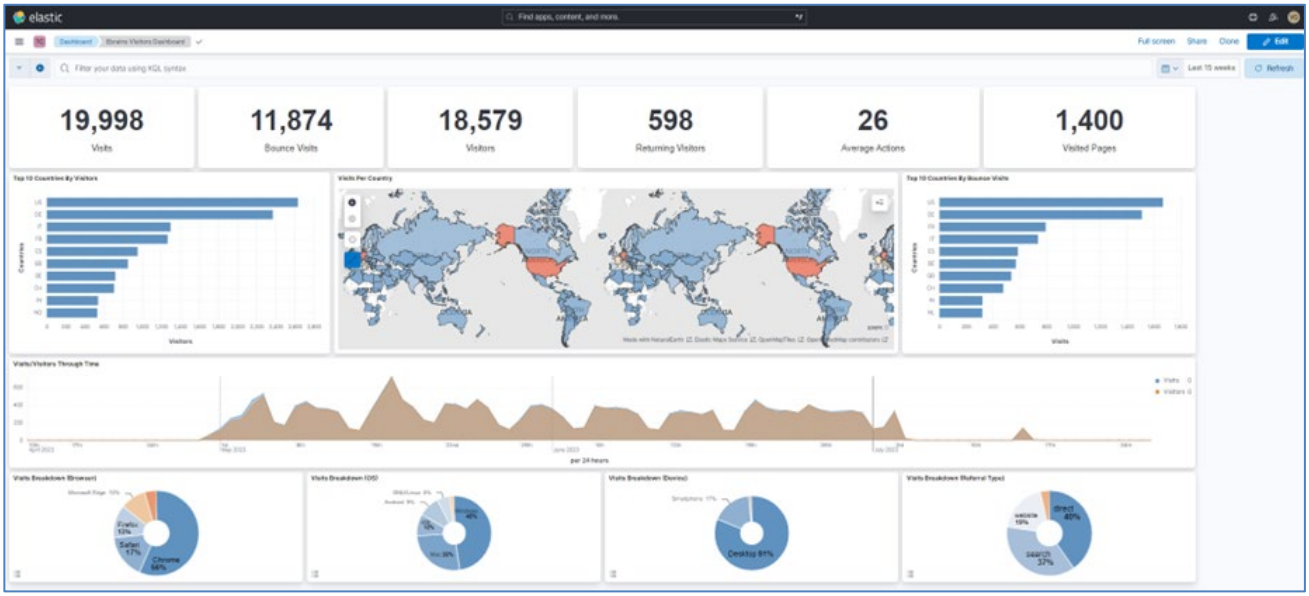

Figure 46: EBRAINS Visits Dashboard

Workflows Executions Dashboard (Figure 47)**:** It serves as a rich source of insight into the functioning and performance of various workflows within the system. It provides an array of valuable information including the total number of workflows, an overview of workflow execution which includes metrics such as success and failure rates, detailed information about the time taken for each workflow execution, and similar details for individual tasks within those workflows. Lastly, the Resource Usage section delivers critical data on the system resources consumed during these workflows and tasks.

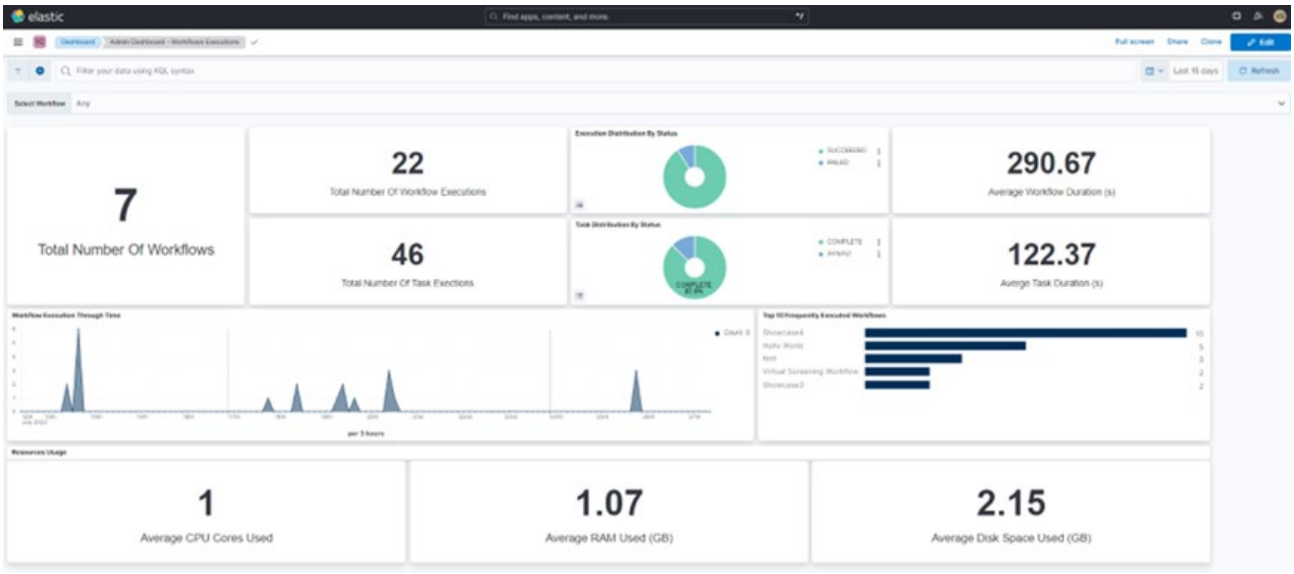

Figure 47: Workflows Executions Dashboard

Notebooks Executions Dashboard (Figure 48)**:** This dashboard offers a detailed overview of test executions within the notebooks system. It displays the overall number of tests, provides a chronological representation of test executions, differentiates results through a visual chart and illustrates divisions among different operational modes.

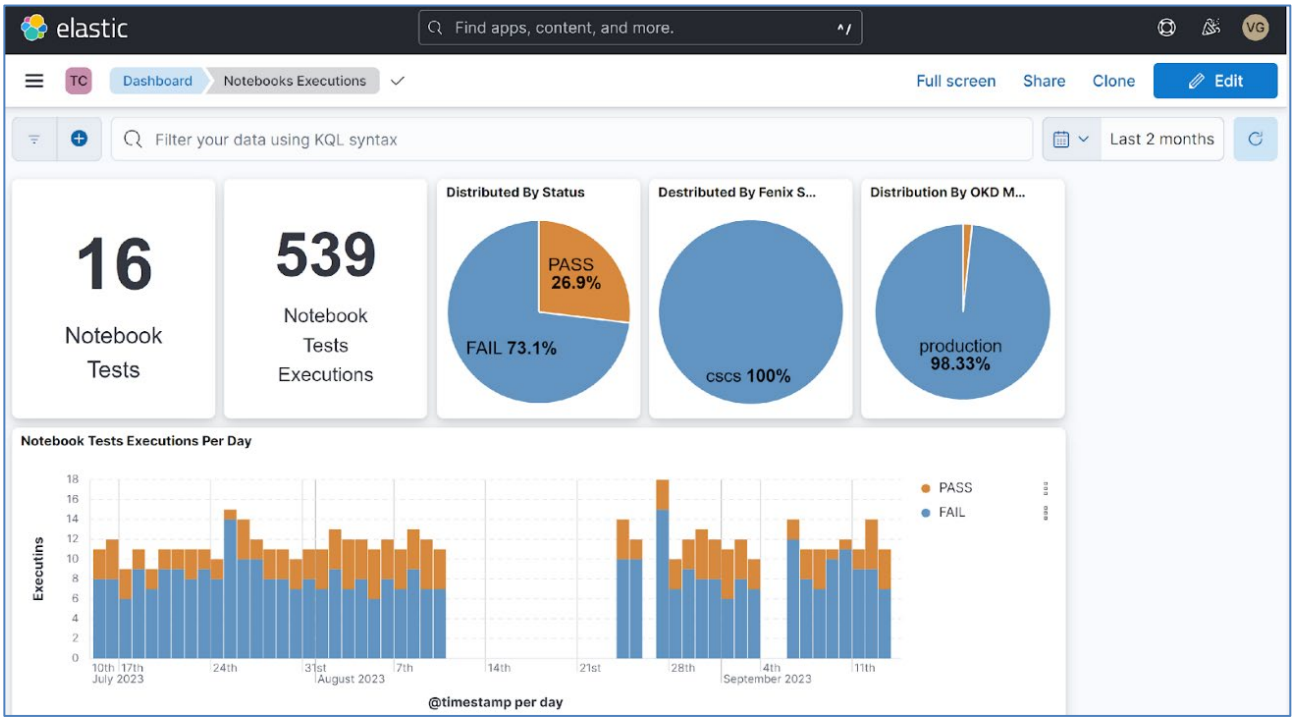

Figure 48: Notebooks Executions Dashboard

8.3 Component information and Integration assessment template

In this Annex, two important documents that facilitated the integration process are included. The "Basic component information" and the "Integration assessment template" documents.

The first one is the document that is intended to facilitate the initial stages of a component's integration path. It requires some basic component information to be provided in a standardised manner so that the Technical Coordination team can effectively engage with a component's team towards initiating the integration activities to EBRAINS RI.

This document can be retrieved from the following link:

https://drive.ebrains.eu/f/5ba703fd933246e38ecc/

The second one, namely, the Integration assessment template was drafted based on the finalized integration requirements, to enable the TC team to track the individual component's progress against every integration requirement relevant to each component. This template was used to facilitate TC to not only monitor each team's progress but also provide targeted feedback to each team based on their specific use case.

Apart from tracking progress relevant to the integration requirements, the Integration assessment template aimed to also capture "Future" level concerns, meaning, information on the vision of the components' development after the end of SGA3, to assess the components' long-term commitment to the RI, and the community in general, with the aim to determine the sustainability of the offerings of each component. The integration assessment template can be retrieved from the following link:

https://drive.ebrains.eu/f/419b3c67b86c47df9226/

8.4 EBRAINS tools software stack

8.4.1 Documentation

The full documentation of the delivery strategy can be retrieved from the following link: https://drive.ebrains.eu/d/0912c976f8a143ff94d8/. The material will be continuously updated to reflect the latest development status.

8.4.2 List of tools

The latest official EBRAINS release, EBRAINS-23.09, includes 59 top-level EBRAINS Spack packages:

- 54 EBRAINS tools: arbor, biobb-analysis, biobb-chemistry, biobb-common, biobb-gromacs, biobb-io, biobb-model, biobb-structure-checking, biobb-structure-utils, hxtorch, nest, neuron, py-bluepyefe, py-bluepymm, py-bluepyopt, py-bsb, py-ebrains-drive, py-ebrains-kg-core, py-efel, py-elephant, py-fairgraph, py-frites, py-hbp-archive, py-hbp-neuromorphic-platform, py-hbp-validation-client, py-hippounit, py-lfpy, py-lfpykit, py-libsonata, py-neo, py-nestml, py-netpyne, py-neurom, py-neuror, py-pynn, py-pyunicore, py-quantities-scidash, py-quantities, py-siibra, py-snudda, py-spynnaker, py-tvb-contrib, py-tvb-data, py-tvb-framework, py-tvb-gdist, py-tvb-library, py-tvb-multiscale, py-tvb-storage, py-viziphant, pynn-brainscales, r-rgsl, r-sbtabvfgen, r-uqsa, sda
- 5 EBRAINS workflow meta-packages: wf-biobb, wf-brainscales2-demos, wf-protein-association-rates, wf-multi-area-model, wf-uq-akar4. Those meta-packages are used to bundle the dependencies of EBRAINS workflows on external tools. Each of those packages corresponds to one EBRAINS workflow or a set of published workflows/tutorials/demos; including them in the official EBRAINS releases ensures that all the dependencies needed for their execution are available as part of the EBRAINS software environment.

8.5 Central vs federated nodes

EBRAINS is a federated e-Infrastructure that offers data, models, and services directly to its users through the EBRAINS AISBL node or via its National Nodes (NNs). It leverages foundational computing and storage resources in a cost-effective, scalable, and reliable manner. From a technical perspective, the Central Hub (EBRAINS AISBL node) is responsible for the following:

- Coordinating the National Nodes.
- Coordinating and managing the operation of services and facilities provided by the Research Infrastructure (RI), including those offered by both the National Nodes and the Central Hub, while ensuring compliance with quality standards established by the Board of Directors.
- Coordinating and managing access to the services provided by the RI, encompassing those delivered by the National Nodes and the Central Hub.
- Identifying and monitoring measurable Key Performance Indicators, focusing on both the excellence of scientific services and the sustainability of the RI as a whole, including its individual services.

A National Node can either be a single EBRAINS Member institution (the Node Partner) or a consortium of EBRAINS Member institutions (the Node Partners) within the same country. Each National Node is responsible for delivering a specific set of EBRAINS services and must also provide supporting services.

Following this concept, the TC has furnished, even before the conclusion of HBP SGA3, additional details about the future architecture (Section 6.1) and preliminary integration guidelines post-SGA3 (Annex 8.1). Concurrently, TC is spearheading a migration initiative to guarantee the sustainability and uninterrupted operation of EBRAINS (Section 6.2).