Interim EBRAINS Infrastructure Implementation Report (D5.4 SGA3)

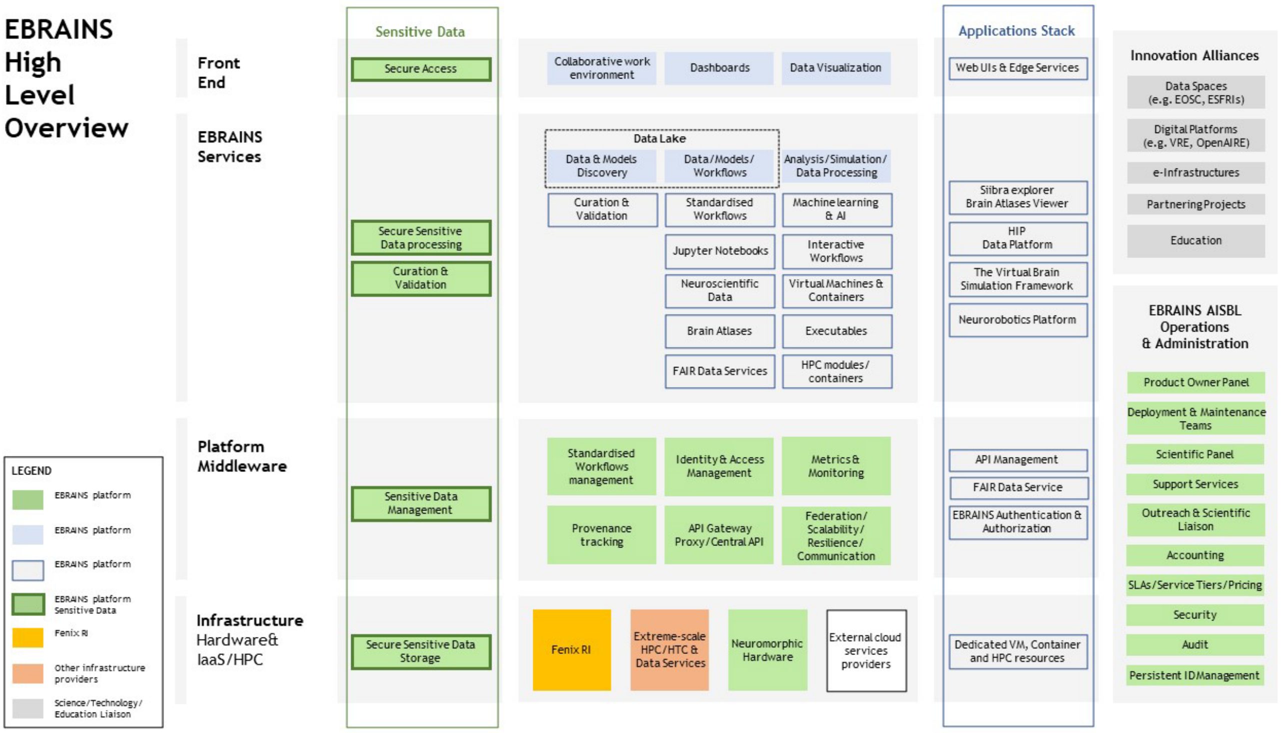

EBRAINS
High
Level
Overview
Front End
EBRAINS Services
Platform Middleware
Infrastructure
Hardware&
IaaS/HPC
Sensitive Data
Secure Access
Secure Sensitive Data processing
Curation & Validation
Sensitive Data Management
Secure Sensitive Data Storage
Collaborative work environment
Dashboards
Data Visualization
Data Lake
Data & Models Discovery
Data/Models/Workflows
Analysis/Simulation/Data Processing
Curation & Validation
Standardised Workflows
Machine learning & AI
Jupyter Notebooks
Interactive Workflows
Neuroscientific Data
Virtual Machines & Containers
Brain Atlases
Executables
FAIR Data Services
HPC modules/containers
Standardised Workflows management
Identity & Access Management
Metrics & Monitoring
Provenance tracking
API Gateway Proxy/Central API
Federation/Scalability/Resilience/Communication
Fenix RI
Extreme-scale HPC/HTC & Data Services
Neuromorphic Hardware
External cloud services providers
Applications Stack
Web UIs & Edge Services
Siibra explorer Brain Atlases Viewer
HIP Data Platform
The Virtual Brain Simulation Framework
Neurorobotics Platform
API Management
FAIR Data Service
EBRAINS Authentication & Authorization
Dedicated VM, Container and HPC resources
Innovation Alliances
Data Spaces (e.g. EOSC, ESFRIs)
Digital Platforms (e.g. VRE, OpenAIRE)
e-Infrastructures
Partnering Projects
Education
EBRAINS AISBL Operations & Administration
Product Owner Panel
Deployment & Maintenance Teams
Scientific Panel
Support Services
Outreach & Scientific Liaison
Accounting
SLAs/Service Tiers/Pricing
Security
Audit
Persistent ID Management
LEGEND
EBRAINS platform
EBRAINS platform
EBRAINS platform
EBRAINS platform Sensitive Data
Fenix RI
Other infrastructure providers
Science/Technology/Education Liaison

Figure 1: EBRAINS High Level Overview (Section 2)

Project Number:	945539	Project Title:	HBP SGA3

Document Title:	Interim EBRAINS Infrastructure Implementation Report
Document Filename:	D5.4 (D51) SGA3 M21 ACCEPTED 220520.docx
Deliverable Number:	SGA3 D5.4 (D51)
Deliverable Type:	Report
Dissemination Level:	PU = Public
Planned Delivery Date:	SGA3 M21 / 31 Dec 2021
Actual Delivery Date:	SGA3 M21 / 20 Dec 2021; accepted 20-May-2022
Author(s):	Amaryllis RAOUZAIOU, ATHENA (P133), Thanassis KARMAS, ATHENA (P133), Sofia KARVOUNARI, ATHENA (P133), Rowan THORPE, ATHENA (P133), Eleni MATHIOULAKI, ATHENA (P133), Orfeas AIDONOPOULOS, ATHENA (P133), Spiros ATHANASIOU, ATHENA (P133)
Compiled by:	Amaryllis RAOUZAIOU, ATHENA (P133), Thanassis KARMAS, ATHENA (P133)
Contributor(s):	Wouter KLIJN, JUELICH (P20), contributed to Section 2 Dennis TERHORST, JUELICH (P20), contributed to Section 4 Andrew DAVISON, CNRS (P10), contributed to Sections 3 and 4
WP QC Review:	Evita MAILLI, ATHENA (P133)
WP Leader / Deputy Leader Sign Off:	Yannis IOANNIDIS, ATHENA (P133)
T7.4 QC Review:	N/A
Description in GA:	Interim report presenting the delivered software, quality assurance performance, and delivery progress of the infrastructure.
Abstract:	Deliverable D5.4 is the latest snapshot of EBRAINS Infrastructure Implementation, refines the initial version of architecture presented in D5.3 and reflects TC's current vision and directions regarding the development, integration, and delivery of the EBRAINS infrastructure and its alignment with the scientific, technical and sustainability requirements of SGA3. The document presents the planning during this period and the work done towards a more coherent vision of the RI with the developed tools, the monitoring services, the updated Software Quality guidelines and the metrics/KPIs for monitoring and assuring the quality of the EBRAINS RI, specifically for the integration process.
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

Table of Contents

Table of Tables

Table of Figures

1. Introduction

In SGA3, the HBP builds 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 serve as the basis for the development of tools and services, and (b) the insight and expertise of scientists flow into the conception and realisation of the infrastructure.

In the framework of SGA3, the final release of EBRAINS, with all the components integrated, will be 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. It will include innovative brain-inspired technologies and computing, enabling also digital applications for industrial and medical use, and it is powered by the Federated Exascale Network for data Integration and eXchange (Fenix[1]) Infrastructure as a Service (IaaS), itself a blueprint for other research communities. EBRAINS aims to 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 final EBRAINS infrastructure, produced during SGA3 and delivered at the end of the Project, will include the components and services developed in HBP.

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 D5.4 is the second deliverable of T5.11. It is the latest snapshot of EBRAINS Infrastructure Implementation, refines the initial version of architecture presented in D5.3 and reflects our current vision and directions regarding the development, integration, and delivery of the EBRAINS infrastructure and its alignment with the scientific, technical and sustainability requirements of SGA3. Reviewers' comments from M9 review have been considered, they were further analysed and provided a basis for our planning during this period, working towards a more coherent vision of the RI with the tools we developed and the monitoring services (Section 3) and updating the Software Quality guidelines (Section 4) and the metrics/KPIs for monitoring and assuring the quality of the EBRAINS RI, specifically for the integration process (Section 5).

Comparing to the previous version of the architecture, notable changes occurred at all levels which resulted in the update of all existing diagrams and introduction of new ones. The most significant change was the introduction of a new (more abstracted) view of the architecture meant to assist the understanding of EBRAINS by external (non-SGA3) stakeholders. Being able to communicate EBRAINS Services and capabilities more effectively is crucial for extending the user-base to community members outside of SGA3. As a consequence of the introduction of this new view, a lot of entities appearing in the architectural diagrams were reframed in scope and repositioned graphically. Another notable change which also resulted in the introduction of new diagrams and several updates in the existing ones, was the introduction of the "Standardised Workflows" concept as a new deployment resource on the EBRAINS RI. Moreover, the diagrams were updated to reflect the current developments and support in the subject of Sensitive Data access, handling, and management in the EBRAINS RI. In general, all the architectural diagrams were updated to reflect the latest development status and communicate the most recent design choices.

Along with the updated architecture, the current document describes how the integration process is organised, providing a high-level roadmap for the delivery of EBRAINS RI, while it also presents the different services and tools that have been developed to help HBP members and the established rules for software quality in SGA3.

The content of D5.3 [1] constituted the basis for some of the sections of this deliverable, while other sections described new approaches that are contributing to the implementation process. Wherever there is a direct relation to the content of D5.3, there is a clear reference in the text, while the

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

changes in relation to the previous deliverable are also detailed in the corresponding sections that are completed sections, not requiring the reading of D5.3.

1.1 Document Structure

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

Section 2, which constitutes the main part of the current deliverable, presents the updates of **EBRAINS Architecture**. 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 current and planned **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 new EBRAINS component developers, aiming to educate and 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. The various views of the EBRAINS RI presented in this section of D5.4 will be further updated in the next version of the deliverable.

Section 3 provides details about the tools that were developed as well as the workflow standardisation aspects that were introduced under the **Platform Middleware**. More precisely, information about the **Software Delivery and Installation for the Lab** is presented, including the main requirements of the delivery process as well as the actual implementation under the EBRAINS RI. **Extra developed tools/services** subsection provide more technical and in-depth information for component DevOps and developers of EBRAINS. In the same context, **Monitoring services** (via Matomo, ELK and more) details in the context of EBRAINS RI will be provided to the readers as a first overview of the work that will continue in the upcoming Phases. Progress about the formalisation and tracking of the status of API ecosystem at EBRAINS will be further introduced by the **APIs development**. **Service for automated headless browser testing of Jupyter Notebooks** was developed in the current Phase and already in use by EBRAINS users, after requirements were gathered and met. Finally, a subsection is dedicated to standardisation aspects provide means and technologies for computational **Workflows** using EBRAINS data, tools, models and software to be defined in a common, well-defined and structured way, executed and monitored by workflow engines running on top of the EBRAINS powerful underlying infrastructure and stored in the central metadata management system, EBRAINS Knowledge Graph.

Section 4 provides details about the **implementation of the integration process**. It presents the **latest developments** that facilitate the integration of the different services in EBRAINS RI and the main axes of **Software Quality guidelines**. This section has been developed under the feedback, contribution and review of the **EBRAINS Software Quality (ESQ-WG)** and **EBRAINS Software Delivery (ESD-WG) Working Groups**.

Section 5 presents the indicators that are used for quality control along with the values measured during the different Phases of integration. It also presents an overview of the most important **achievements** of **Technical Coordination (TC)** so far, where we are **now**, the **risks** we have faced or are expected to face and our **planning** for the immediate future.

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

D5.4 is the latest snapshot of 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 here all the information related to the EBRAINS technical approach. This information is already consolidated through the different meetings, documents or presentations and initially collected and presented in D5.3. D5.4 is focusing on Architecture and the different tools and services developed to facilitate the integration process and the use of EBRAINS RI, so it is not only updating the information of the previous deliverable, but also presents new tools and services. Annexes at the end of the Deliverable provide additional information about the different documents the reader may need for a better understanding of our approach. All these documents are referenced also in D5.4. Besides these documents, D5.4 uses information from wiki pages of EBRAINS Collaboratory or makes direct reference to other Deliverables of the Project.

There is a direct link 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 D5.4 refers to updated QC/KPIs on technical level (development, testing, integration, operations). D5.4 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 and their updated versions D1.2, D2.2, D2.3, D3.5) serve as an information source on possible new RI requirements related to the Showcases, while D4.1, D4.2, D4.4, D4.7, D4.8, D4.9, D5.1, D5.2, D5.5 and D5.6 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.

D5.4 represents the second step towards 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 D5.7 - EBRAINS Infrastructure – M36.

1.2.1 Terminology

Please note that in D5.4, as in D5.3, to avoid confusion, the term **'EBRAINS RI'** refers to t**he final output of SGA3** (i.e. the EBRAINS Research Infrastructure), delivered at the end of the Project. All services currently offered are **assumed to be currently not integrated in the EBRAINS RI**. This terminology has no visible impact on our end-users and SGA3 planning/Deliverables. Instead, it is being used to allow us to introduce (a) a clean-slate approach, and (b) control and clarity over the gradual integration of current services as EBRAINS RI components.

Table 1 summarises the more important terms used in EBRAINS RI; not in terms of acronyms, but regarding actual usage, given the vast set of concepts and technologies addressed by EBRAINS.

Table 1: Terms used in EBRAINS RI

Term	Explanation
User	Depending on context could mean: • end-user/researcher = strictly public-interface user • external contributor scientist/developer = developing external applications bundling EBRAINS public APIs • internal scientist/ internal developer/component owner = contributor of EBRAINS components • EBRAINS itself = as user of core services & Fenix resources

Term	Explanation
	Participants may be more than one of these. Where "user" is not contextually explicit enough more specific wording is used.
Use-case	A specific scientific or technical challenge as encountered in the HBP or EBRAINS and describes the current or expected ICT needs.
Non-interactive tools	Pieces of software that run programs where users parameterize the input only before the runtime (aka execution).
Standardised Scientific Computational Workflows	Common, standard, structured recipes well-defined via Common Workflow Language open, standard, format with specific types of input, output and computational steps for describing workflows producing scientific objectives. Workflow steps refer to computational bundled non-interactive command line tools also well-defined via Common Workflow Language with specific types of inputs, outputs and software to run related to data manipulation.
Interactive tools	Pieces of software that run programs where users are able to parameterize inputs during the runtime (aka execution) shifting the results on-the-fly.
Interactive Workflows	A series of processes offered via Graphical User Interfaces for on-the-fly parameterizing of the tasks and jobs running interactively in the underlying infrastructure (to be better defined in the upcoming Phases)

Table 2 lists the most important acronyms that the reader can find across the different sections.

Table 2: 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 (financially-backed promises to customers)

2. Architecture

In this section, we present and elaborate the **EBRAINS Architecture**. The first version of the EBRAINS Architecture was detailed in "EBRAINS Technical Coordination Guidelines" (D5.3, Section 3, [1]) during SGA3. In accordance with the Technical Coordination's delivered 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 later stage of EBRAINS Phase 1 and during Phase 2 (please refer to section 2.2 for information in Phases). The revised Architecture presented in this section lays the foundation for the subsequent Phases (3 and 4) towards a state-of-the-art, scalable and sustainable architecture upon which the EBRAINS RI will be released at the end of SGA3.

For completeness and context purposes, we can recall from the "Technical Coordination Guidelines" 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 been understandably a challenging endeavour and one of the core outputs of our work.

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 a *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.

Considering the complexity of the RI, as well as the various stakeholders, architectural diagrams have been created at multiple abstraction-levels as appropriate for different purposes and audiences. A 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 to grasp an extensive overview of the platform level services and offerings.

The various views of the EBRAINS RI that are presented in this section, will be continuously iterated, reviewed and updated until the end of SGA3 to accurately reflect the current and projected status of the EBRAINS RI.

This section is outlined as follows:

- Section 2.1 is an addition compared to D5.3 and presents the architecture diagrams design approach, the procedure that directed the effort, as well as a plan for continuously updating the current architecture diagrams and creating new ones as EBRAINS RI evolves, matures and more targeted diagrams need to be created depending on the scope and audience.
- Section 2.2 updates on the status of the development/implementation phases that have been established to keep track of the evolution and progress during SGA3, and also shape the iteration cycles for the mapping of the infrastructure architecture.
- Section 2.3 presents the **EBRAINS High-Level Overview**, meant to assist the understanding of EBRAINS by external (non-SGA3) stakeholders.
- Section 2.4 presents 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 presents the **Logical Architecture** of the EBRAINS RI at two levels of abstraction and detail.
- Section 2.6 includes the past, current and planned **Physical Deployment** of EBRAINS over the available computing, storage, and networking resources.
- Finally, Section 2.7 provides an overview of supported deployment options both for **current** and **new** EBRAINS component developers, aiming to educate and support them in the operationalisation 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.

2.1 Architecture diagrams design approach

Influenced by the C4 model ([2], [3]) which is a lean graphical notation technique for modelling the architecture of software systems, we documented EBRAINS RI Architecture 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 2.3, 2.4) to more representational, focused views (Sections 2.5, 2.6).

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 hand-in-hand 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 is an **ongoing** and **iterative** process with continuous structured input from all levels of the EBRAINS RI. Requirements are 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, and validation/feedback by all involved stakeholders is necessary to identify missing elements, components, or functionality.

Currently, the following EBRAINS Architectural diagrams are available:

- High-Level Overview
- Conceptual Architecture
- Logical Architecture
- Physical Deployment
- An overview of supported deployment options on EBRAINS

It is envisaged that during the next iterations of the architecture the following diagrams will also be produced, updated, and collected:

- A more engaging and improved version of the EBRAINS High-Level Overview will be produced, 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.
- All the individual EBRAINS component diagrams will be updated and collected to fully document all aspects of the RI.
- Descriptive diagrams per SC will need to be created. The example for SC3, which is currently under development, is presented in Annex 8.7.
- The Physical Deployment diagrams will be thoroughly redesigned to make them more sustainable, readable, maintainable, and easier to update.

2.2 Phases

Design, development and implementation activities during SGA3 which will ultimately lead to the official release of EBRAINS are tracked against specific time intervals that have been established by EBRAINS TC (see D5.3, Annex II: TC planning, [1]) in order to frame the scope of the development and integration efforts on the one hand, and give specific context according to the high-level planning and contractual obligations on the other. These time intervals are identified as "EBRAINS Phases" and aim to create slots for targeted allocation of effort in specific tasks along the lines of (a) a pragmatic high-level roadmap for the delivery of the EBRAINS RI, (b) technical and infrastructural stability and (c) inform the development, delivery, operations and maintenance planning of the various EBRAINS RI components.

During EBRAINS Phases the architecture of the RI is continuously revised and updated in order to ensure that the envisaged architecture addresses both current and future requirements of all involved stakeholders that would be a guarantee for the delivery of an RI that facilitates scientific research to the highest possible degree.

The four established phases cover the entire duration of SGA3:

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

EBRAINS Phase 1 laid the foundation for the intense integration efforts that are taking place during SGA3 and will lead to the delivery of EBRAINS. The aim was to utilise the underlying infrastructure provided by ICEI/Fenix to: (a) assess and finalise the ICEI/Fenix-powered services (e.g. containerisation) to be deployed and become available across all sites during Phase 2, (b) set up all TC-related processes and instruments for monitoring software quality and delivery, (c) migrate a select number of components from a project-centred deployment and management mode to an EBRAINS-centred one (see Sections 4.2.1 and 4.2.3.1), (d) evaluate and propose the required middleware components, (e) deliver a series of Proofs of Concept (PoCs), (f) inform the elaboration of the EBRAINS RI Architecture (conceptual, logical, physical) and finally (g) prepare the on-boarding of the 3 additional Fenix sites.

Regarding the aforementioned PoCs, a series of exercises were designed, developed, iterated, and delivered in this phase. They represented the most important architectural and operational decisions established for the EBRAINS RI at that point. The PoCs served to test these concepts, improve them, and deliver blueprints for large-scale adoption in the subsequent phases. More specifically, the exercises spanned the following areas:

2.2.1.1 Continuous Integration (CI)/Continuous Delivery (CD) framework & pipelines

EBRAINS phase 1 components had their official code repositories mirrored in the EBRAINS Gitlab platform with appropriate continuous integration configuration setup. This ensured a central initialisation point for building container images, integration tests, and performance tests as they develop. The CI pipelines have either already started working or are in the final stages of formalisation. Since this PoC laid the foundation for EBRAINS CI/CD, work will be intensively continued in the next phases of integration.

2.2.1.2 Container Orchestration with OKD

Our goal was to prioritise containerised deployment instead of VMs where applicable and apply this rule to the components participating in Phase 1. Containerised (in OKD) deployment is suggested in case of a stateless application or where high-availability and scalability of a stateful application is crucial. With containerised deployments and OKD, application development can be accelerated, modern DevOps practices can be enabled, and developer productivity is increased by allowing integration of all necessary tools. Application of these principles was performed across all the other PoCs (e.g. observability, notebooks). During Phase 1, EBRAINS benefited from a recently-installed second production OKD cluster, allowing for more workloads and providing load-balancing, and fail-over capabilities to EBRAINS applications running on it like the Collaboratory notebooks.

2.2.1.3 Multi-scale, multi-site workflows

During Phase 1 the goal was for a small set of 2-5 workflows to be implemented and working across two of the five Fenix RI sites, with any technical interventions required to deliver them being addressed formally in the following phase. This required clarifying and highlighting contextual usages of the broad term "workflows", since within EBRAINS there are many opportunities for differing interpretations of the general term (computational workflow vs scientific workflow, workflow recipe vs workflow execution). At this stage the main goal and primary exercise was to provide templates for users to easily execute the following: retrieve a dataset or model from Archival storage having located it via metadata in the EBRAINS Knowledge Graph, run a simulation in an HPC system with the retrieved data, send the output back to Archival storage while updating the Knowledge Graph with metadata about the output's location, for easy retrieval.

A small PoC regarding the workflows in EBRAINS was an important stepping stone for a more in-depth pursuit of workflows related to standardised scientific computational ones (Section 2.2.2.3) as well as interactive ones (Section 2.2.2.4) and for providing more insights to the scientific communities, component owners and developers of EBRAINS RI. During the next Phase, it was decided that the formalisation of scientific workflows should be achieved using the Common Workflow Language (CWL), an open standard for describing analysis workflows and tools in a way that makes them portable and scalable across different execution environments. Defining workflows in this open, common, standard way, is essential for ensuring (a) reproducibility and reusability of the scientific work produced in EBRAINS, (b) workflow portability and seamless execution in different scientific infrastructures, (c) findability and accessibility of workflows, both crucial for EBRAINS as a FAIR infrastructure and (d) interoperability and reusability even outside EBRAINS, by other Research Infrastructures and scientific communities.

2.2.1.4 Monitoring & Observability

Effort was dedicated to establishing all available and necessary component-, platform- and infrastructure-specific sources of information, assessing completeness of the collected information, and prototyping implementations for automated sharing (push/pull) of information to a centrally managed monitoring service. At a PoC level, an ELK stack [4] has been installed to integrate with the various infrastructure level monitoring systems, as well as with the various logging and metrics collection mechanisms at the component/application/service level, so that collective and insightful monitoring information for the EBRAINS RI can be aggregated and presented in a unified way through dedicated dashboards. The evolution of the PoC to a production implementation will continue in the following phase.

2.2.1.5 Collaboratory/Notebooks

Effort was dedicated to identifying multiple options for curated images and instances for end-users (back-end & prototype front-end), spawning within the same or remote Fenix RI sites (no longer a single monolithic image for pseudo-integration). The User Requirements were the following. Users must be able to select (a) from a selection of curated images and/or software tools that are available in the containers spawned from these images, and (b) processing environments of different size (tiered, from free to commercial) CPU/memory and/or type (e.g. HPC via service accounts). Furthermore, introduction of simple end-to-end (e2e) testing for production notebooks (pass/fail) was actively pursued.

2.2.1.6 Provenance

Effort was dedicated to identifying all sources for information that needed to be captured (e.g. Collaboratory notebooks, workflows, processing environments, data/models) per use case and the currently available metadata for each source (emphasis on automatically generated metadata/logs). We conducted discussions about prototyping the assembly of the available metadata under one record with PID (persistent identifier) and providing a simple API. Based on this output, during Phase 2 we will establish a roadmap for formal PROV-O metadata/service.

2.2.1.7 Prototype multi-site, high-availability (HA) deployment for the core services

The Physical Deployment architectural diagrams (see Section 3.4) provide more insight regarding the envisaged deployment status that was determined during Phase 1. The actual technical assessment of the alternative options (e.g. active/active, active/standby, etc.) for the HA deployment of the core services and the actual implementation of the solutions will be the focus of the following phases. Currently, and directly relating to the PoC for Container Orchestration with OKD, a secondary production OKD cluster was installed to provide load-balancing and fail-over capabilities to the Collaboratory notebooks.

For the migration of a select number of components from a project-centred deployment and management mode to an EBRAINS-centred one, the EBRAINS TC drafted an initial set of requirements to be applied by the component development teams, to ensure they are onboard with the integration path, and that all design and development effort is aligned with the project's mandates. Discussions and hands-on-work with the component owners (COs) following the release of "Phase 1 guidelines" were fruitful and identified several issues that needed to be rectified, including missing items and new requirements for facilitating both the TC's and the CO's activities in the subsequent phases, working towards the sustainable delivery of EBRAINS.

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

In Phase 2, the aim was to integrate nearly 50% of the currently listed components to the EBRAINS RI and, at the same time, ensure integration practices' full conformance with Deliverable "D5.3 EBRAINS Technical Coordination Guidelines" [1].

In total, **20 out of the 51** currently listed EBRAINS components are officially participating in the integration process and are working at all aspects that concern their integration to EBRAINS RI. It is worth mentioning that 4 of the Phase 2 participating components have officially started their integration effort during Phase 1 and the rest onboarded on the integration process at the beginning of Phase 2. More specifically, significant progress has been made regarding testing, automation, development, documentation, and deployment (where applicable) in all the participating components. Progress is closely monitored by Technical Coordination and feedback is continuously provided to the component owners in an effort to intensify the integration activities. Apart from the officially participating components, there are also many others that have expressed interest and are working ahead of their planned onboarded date with the same standards and on the same integration requirements. This is something that ideally will accelerate the integration process given that starting from Phase 3, all listed EBRAINS components will be officially part of the integration process. After the end of Phase 2 a detailed assessment of the participating components will be conducted against the integration requirements. The results will be shared with the component owners to facilitate and better target their efforts.

A key action item during Phase 2 is also the expansion of the EBRAINS stack of services, where possible, to fully utilise at least two of the ICEI/Fenix facilities (namely CSCS and JSC - see Section 2.6), investigate which services could be deployed in more than one site, prepare the on-boarding of the remaining three ICEI/Fenix facilities, and identify site-specific considerations according to the EBRAINS Architecture. Intense work is currently performed to formalise the EBRAINS APIs (de facto, documentation, new) and establish API management as well as gradually integrate middleware components across the RI. Regarding EBRAINS RI-wide Monitoring, Accounting and Observability services, these are actively being moved forward. The main systems that support these services have been designed and are either already implemented or close to completion. The processes that will leverage the operation of these services and will lead to the achievement of EBRAINS Observability goals are continuously improved and are beginning to be applied on the various EBRAINS RI components that are participating in the Integration Phases. With the beginning of Phase 3, where all components are deemed to participate, we will have metrics and monitoring information from all the EBRAINS applications/services/APIs feeding the EBRAINS Web Analytics and back-end monitoring services and thus we will be able to produce critical insights on the EBRAINS RI usage and utilization of the underlying resources.

More specifically, we report the progress achieved during Phase 2 in each of the important areas that were demarcated during Phase 1 in the sub-sections below:

2.2.2.1 Continuous Integration (CI)/Continuous Delivery (CD) framework & pipelines

The EBRAINS DevOps platform (EBRAINS Gitlab) is fully operational and in production. The EBRAINS RI components that participate in Phase 2 have their official code repositories mirrored in the EBRAINS Gitlab and as a result the EBRAINS CI pipelines for building container images, integration tests and performance tests on the Container Orchestration Platform, on the OpenStack clusters and on HPC, now, have a **central initialisation point**. The components participating in Phase 2 are actively working on preparing/improving new/existing CI jobs to automate, in collaboration with the Technical Coordination, the respective integration and testing steps. Technical Coordination is actively engaged in the development of the EBRAINS delivery pipelines where appropriate. It is expected that while components onboard, they will often initially include the subdirectory-namespacing-based delivery/deployment configuration described in Section 4.1.2 in their upstream repository, where they have usually already been providing similar configurations for deployment external to the EBRAINS RI. Once under central management by the EBRAINS DevOps engineers, those continuous-integration configurations could more easily be extended for comprehensive system/performance/security-testing, deduplication with other components, code reuse, platform-wide caching, more uniform deployment as part of an interoperable system with high-availability, etc. This is the interpretation of "integration" closer to "tight integration".

2.2.2.2 Container Orchestration with OKD

EBRAINS currently features two production OKD clusters at different Fenix RI sites that enable Kubernetes workloads. This has resulted in the multi-site deployment of certain key services for load-balancing and high-availability purposes. More and more components gradually realise the benefits of containerised deployments and thus more services are being deployed on the two available OKD clusters. Moreover, a third cluster is currently targeted at the third Fenix RI site that became operational recently. Gradually, each of the Fenix RI sites should feature one production OKD cluster as instructed by the EBRAINS architecture diagrams. This will allow for the distribution of many core EBRAINS services across the available infrastructure resources, leading to the scalability and overall robustness of the EBRAINS platform.

2.2.2.3 Standardised Workflows

After the work that was presented in the Deliverable D5.3 (Section 2.2.1.3, [1]), it was determined that EBRAINS RI consists of various types of workflows. During this time, Technical Coordination team defined in concrete ways different aspects related to workflows alleviating any semantic misunderstandings (Annex 8.5). Technical Coordination team is currently focused on the means and technologies to be provided to the EBRAINS users for standardised workflows with non-interactive command line tools used as workflow steps. Standardisation is related to different aspects; from defining in a common and standard way workflows and tools, to executing and monitoring them via workflow management systems as well as storing them in registries for easy accessibility and findability. More information related to standardisation aspects can be found in Section 3.2.

In every given opportunity (from workshops arranged, to Human Brain Project Summit[2] and the Human Brain Project CodeJam #12[3]), Technical Coordination team depicted the different definitions of "workflows" terminology to the Consortium (developers and scientists), focusing on the standardisation aspects that we identified as critical and, thus, needed to be adhered to. Those aspects are: (a) a common, structured way of defining workflows and tools as recipes, (b) bundling non-interactive command line tools in a structured way in order to acquire reusability (c) execution and monitoring of common, structured recipes of workflows in different underlying infrastructure, and (d) storing recipes and provenance information inside Knowledge Graph. These workflow recipes (prospective provenance) as well as information after the workflow execution (retrospective provenance) will be later linked with scientific outcomes (data and models) into Knowledge Graph for making neuroscientific work easily findable, accessible and reproducible.

We concluded that the Common Workflow Language (CWL) standard will be used by component owners and developers in order to define standardised workflows. A variety of workflow engines for execution and monitoring of standardised workflows are installed on OKD cluster as well as different HPC systems. Work is on-going between different WPs for assessing various technical solutions for choosing primary workflow engines in different cases. In the meantime, we are in the process of finalising the means for EBRAINS scientists to submit as well as monitor their workflows, in specific endpoints in an opaque way. Also, we are in the process of finalising guidelines for developers to bundle their tools in a reusable manner. Meanwhile, the Technical Coordination team is also highly involved with providing the means and technologies to different Showcases (1,2,3 and 5) in order to standardise their work and enhance its reusability and portability.

2.2.2.4 Interactive Workflows

After the work that was presented in the Deliverable D5.3 (Section 2.2.1.3 [1]) we identified that workflows were divided into two main categories (a) standardised workflows with non-interactive workflow steps (Section 2.2.2.3) and (b) interactive workflows. In the first case non-interactive workflow steps are pieces of software in which users parameterize the input only before the runtime, while in the second case interactive workflows consist of tools that are pieces of software in which users can parameterize the input during the execution. For the latter, work that identifies the current state, the limitations, as well as the missing points has already started by the Scientific Liaison Unit (SLU) and Technical Coordination team. Jupyter notebooks used for executing linear chunks of code, writing documented steps, and retrieving output results, are currently the central point of reference for interactive workflows. In order to make EBRAINS a FAIR Research Infrastructure, it is essential for those interactive workflows to abide by the FAIR principles (findability, accessibility, interoperability, reusability) as well. The main goal in the upcoming Phase is to ensure that Jupyter Notebooks, that are not by definition FAIR objects, will adhere to those principles, by identifying means of integrating them to the Knowledge Graph and storing provenance information related to definitions, executions as well as scientific outcomes of interactive workflows too.

2.2.2.5 Monitoring & Observability

The prototype implementation of automated sharing (push/pull) of information to a centrally managed back-end monitoring service has been completed. The prototype can integrate with the various infrastructure level monitoring systems, as well as with the various logging and metrics collection mechanisms at the component/application/service level, so that collective and insightful monitoring information for the EBRAINS RI can be aggregated and presented in a unified way on dedicated dashboards. Work is ongoing for identifying elusive component-, platform- and

[2] https://summit2021.humanbrainproject.eu/

[3] https://www.humanbrainproject.eu/en/education/participatecollaborate/infrastructure-events-training/codejam-12/

infrastructure-specific sources of information, as well as assessing completeness of the collected information. All the processes that will leverage the operation of the back-end monitoring service and will lead to the achievement of EBRAINS Observability goals are continuously improved and are beginning to be applied to the various EBRAINS RI components that are participating in the Integration Phases.

2.2.2.6 Collaboratory Lab/Notebooks

Work has progressed significantly since Phase 1. End-users can now select different execution sites for their Notebooks running on the Collaboratory Lab environment (Lab). There is also in place a functionality where users can request a more powerful processing environment in case the default one does not cover the requirements of their use case. Moreover, a new delivery strategy (Section 3.1) has been designed and implemented which facilitates the co-existence of software tools with conflicting dependencies within the same Lab processing environment. This is a substantial improvement, as tool development has been separated from the Lab’s container image creation, maintenance, and curation. This leads to a more sustainable Lab environment in the long term. Finally, the service for automated headless browser testing of the Collaboratory Notebooks (Section 3.3) has been completed and is available to end-users. Work is ongoing to deliver the Lab environment at all Fenix sites and to offer the new delivery strategy for software tools to all Lab instances.

2.2.2.7 Provenance

Work on provenance tracking has proceeded on several fronts. Details about the Provenance API (and the OpenMINDS schema which provides the metadata format for submission to the KG) can be found in Section 3.6.1. We have coordinated the efforts of syncing between developers presently working with/on one or both provenance-method classes - prospective (recipe-based, parameter-based) and retrospective (log-based, tracing-based), and varying implementations of each, to ensure consensus and opportunities for shared resources and efforts, for increased uniformity/visibility, and reduced fragmentation of output.

2.2.2.8 Prototype multi-site, high-availability (HA) deployment for the core services

The Physical Deployment architectural diagrams (see Section 2.6) provide more insight regarding the envisaged deployment status. The diagrams emphasize on the key requirement that EBRAINS must exploit the multi-site resources availability for load-balancing and high-availability purposes. The actual technical assessment of the multi-site deployment and the alternative HA options (e.g. active/active, active/standby, etc.) for the HA deployment of the core services and the implementation of the solutions is currently investigated on a case-by-case basis. Some services are already deployed in more than one site and some other have been deployed on an HA configuration. More targeted effort for the multi-site, HA deployment for the core services will be dedicated during the next Phases.

2.2.2.9 Component Integration Requirements

Technical Coordination has developed, since Phase 1, horizontal technical requirements that frame the integration effort of the EBRAINS components. These requirements are continuously updated to reflect the latest development status and homogenously raise the technical bar for the entire SGA3 Consortium towards ensuring the software quality and sustainable delivery and deployment of all EBRAINS components. The current integration requirements (as stand during Phase 2) can be found in the Annex 8.2. The development status and progress of the participating components is assessed against the aforementioned requirements in a standardised way through the assessment template (Annex 8.6) drafted to monitor the components’ progress objectively.

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

In the next phase, our goal is to integrate 100% of EBRAINS components into the EBRAINS RI and, at the same time, ensure integration practices' full conformance with Deliverable "D5.3 EBRAINS Technical Coordination Guidelines". The definition, development and testing of business/pricing models is expected. All the processes for the handover of the RI to the EBRAINS AISBL will be elaborated, bootstrapped, and tested.

2.2.4 Phase 4 (Duration: M34-End of SGA3; Output: EBRAINS RI)

In Phase 4, the EBRAINS RI will be delivered, while the handover of its operations will be transferred to the EBRAINS AISBL.

2.3 EBRAINS High-Level Overview

In this section, the EBRAINS High-Level Overview is presented. This diagram is an abstraction of the EBRAINS Conceptual Architecture diagram (Section 2.4). The conceptual architecture is based heavily on the architecture section of the SGA3. It was presented in the D5.3 (Section 3.2, [1]) as the first level of abstraction for the EBRAINS architecture during Phase 1. Based on the feedback received from all stakeholders involved in the co-design process of the architecture, we identified the need to increase the level of abstraction and readability for audiences not familiar with the EBRAINS ecosystem. For this reason, TC added an "introductory" diagram to convey high-level and fundamental information for the external scientific communities as target-audience - see the EBRAINS High-Level Overview diagram in Figure 2. This diagram is updated with current design & development, and with important aspects of EBRAINS which are less apparent elsewhere, like support for deploying and hosting users' full-stack applications cost-effectively at scale, notably including end-to-end support for Sensitive Data flow.

The diagram in Figure 2 is an abstraction from the conceptual architecture, but also notably differs in a few keyways from the architecture status during Phase 1. For example, "Service Offerings" was renamed to "EBRAINS Services" to unify wording with the EBRAINS website, "Democratization" was renamed to "Innovation Alliances" to better convey the strategy required to innovate in a dynamic scientific & commercial ecosystem, and support was added for "Sensitive Data", "full-stack deployment", "Data Lake" functionality, and the possibility to deploy workloads to external cloud services providers.

The architectural components that are detailed in the High-Level Overview and are also visible in the conceptual architecture, will be detailed in the present section and will be considered known from their first appearance and onwards.

Section 2.3.1 provides an outline of the High-Level Architecture. Section 2.3.2 provides the next level of detail for the main components of the EBRAINS RI. It is quite extensive and describes in detail all boxes of the High-Level architectural diagram presented in Figure 2.

2.3.1 High-Level Overview outline

The EBRAINS Architecture comprises of the four layers depicted in Figure 2. The architecture is completed with the EBRAINS AISBL Operations & Administration (on the right), along with the Innovation Alliances which are strategic partnerships exposing EBRAINS RI to the world. Starting from the top layer:

- **Front end**. This comprises a non-exhaustive collection of front-facing, end-user interfaces and services, enabling the discovery, use and management of the EBRAINS infrastructure by end users (e.g., scientists, teams, organisations, EOSC) and EBRAINS personnel (e.g. helpdesk, administrators, governance bodies).

- **EBRAINS Services**. This encompasses the scientific and innovation output of HBP SGA3 WPs and SCs, i.e., all software artefacts responsible for the provision of EBRAINS services (e.g. analysis, simulation, data curation).
- **Platform Middleware**. This contains all components responsible for managing, allocating, and monitoring the use of all underlying computing resources provided by "Infrastructure Hardware & IaaS/HPC" layer, affording flexibility, scalability, efficient use of resources, fault-tolerance and dynamic demand-aware provision of services to users.
- **Infrastructure** - Hardware & IaaS/HPC. This contains: (a) the storage and compute services provided by the Fenix RI, with a guaranteed allocation to the EBRAINS infrastructure; (b) the external Extreme-scale HPC & Data facilities provided on a grant basis via EuroHPC, Prace or potentially other 3rd parties; (c) the Neuromorphic Hardware provided by the HBP Facility Hubs, SpiNNaker and BrainsScaleS; as well as (d) External cloud services providers that can be used to offload non-sensitive data processing workloads.

The EBRAINS platform inherently supports:

- **Sensitive data** applications, as it provides the required components for sensitive data handling.
- Hosting of the **full stack for applications** and services deployed alongside the core EBRAINS Services.

2.3.2 Architecture components

2.3.2.1 Front End

A non-exhaustive set of end-user components, primarily at the user-facing end of the EBRAINS RI, serves to introduce potentially new or adapted end-user interfaces with relatively small effort during the RI's lifetime (e.g. scalable visualisation libraries/interfaces, enhanced EOSC-powered discovery of relevant scientific assets/services, integration of new AutoML frameworks). This ensures that EBRAINS remains fully aligned with evolving user needs and pertinent scientific output by repurposing existing components and service endpoints, thus extracting added value from EU's investment.

This group comprises a collection of stand-alone applications and services providing integrated user interfaces which can collectively expose the full breadth of EBRAINS Services. Furthermore, they support all types of users (from new to advanced), allowing a user to increase their know-how and sophistication/complexity of EBRAINS interactions, within the same UI, thus adapting to their needs. Currently, the group comprises of the following:

- The **Collaborative work environment** is currently the core entry-point for the discovery and invocation of EBRAINS services. It provides access to documentation, training material, introductory videos and examples in a streamlined, intuitive and informative manner, to assist and guide end-users in using EBRAINS facilities. It enables users of EBRAINS to collaboratively access tools and discuss results in a shared digital (web) space, serving as the first point of contact with EBRAINS for groups and communities of new or inexperienced users, as well as scientific experts, and as a hub for the collaborative writing of scientific publications close to simulation, analysis and visualisation tools.
- Several **Dashboard** categories are also provided from the various EBRAINS services that offer more intuitive and purpose-specific interfaces to all types of users.
- **Data Visualization** options enable users to discover the full breadth and depth of the highly curated, growing, and valuable EBRAINS data assets, including links with relevant publications (i.e. how they have been produced, how they can be used) and EBRAINS services/workflows (i.e. where and how to use them).

2.3.2.2 EBRAINS Services

This is the second layer of the EBRAINS RI (illustrated in Figure 2), closely linked to and powered by the third layer (Platform Middleware). It encompasses (in standard software architecture terms) the 'business logic' of EBRAINS, i.e. all software artefacts responsible for the provision of EBRAINS services (e.g. analysis, simulation, curation). The sub-components in this layer may interact directly with each other, or indirectly via the underlying "API Gateway Proxy/Central API" services, enacting the specific interrelations and workflows required for their operation. In the same manner, they support the definition, execution, and monitoring of complex data-processing-simulation pipelines & workflows engaging a series of ad hoc components to instantiate open-ended scientific research. "Data & Models Discovery" in conjunction with the "Data/Models/Workflows" form the EBRAINS "Data Lake" layer. This is a centralized repository for storing structured and unstructured data at any scale, and for running different types of analytics. Those can include dashboards and visualizations to big data processing, real-time analytics, and machine learning to guide decision-making.

2.3.2.2.1 Data & Models Discovery

These comprise a collection of internal and external components and workflows that implement EBRAINS curation workflows for data assets, methods and models. Regardless of the type of process supported (e.g. expert-driven manual curation), their physical deployment (e.g. within external non-ICEI sites), or level of automation (i.e. from entirely manual, to fully automated), all curation workflows are extensively monitored and documented, to ensure their reproducibility.

- **Curation & Validation**: This comprises an extensible collection of formalised data and model/method curation workflows, with their final output being data assets forward for indexing, management, linking, and provisioning to the "Data/Models/Workflows" architectural component.

2.3.2.2.2 Data/Models/Workflows

The Data/Models/Workflows component is one of the most important components of EBRAINS, being responsible for the sustained, organised, traceable and streamlined provision of data assets, models and methods to not only all other EBRAINS components but also external applications/services through the provided FAIR Data Services. These assets are invaluable for the scientific community and their availability from a central repository in EBRAINS is of critical importance for sharing, reproducing and promoting Open Science. The assets are:

- A **Standardised workflows** catalogue for uploading, sharing, accessing, and finding validated recipes for complex data-processing pipelines.
- A **Jupyter Notebooks** catalogue for uploading, sharing, accessing, and finding validated code for interactive experiments.
- **Neuroscientific Data** assets which can be either the input or the output of experiments.
- **Brain Atlases** which are the entry point for finding and analysing data based on location in the brain. Users can find brain data in a 3D spatial framework for easy comprehension, comparable to the way Geographical Information Systems organise data in 2D maps of the surface of the Earth.
- **FAIR Data Services**: Tools and services to provide in-depth (Tier 3) curated data and metadata making the data interpretable and discoverable through more advanced EBRAINS Knowledge Graph searches.

2.3.2.2.3 Analysis/Simulation/Data Processing

This grouping contains components encompassing all current and foreseen Analysis, Simulation & Data Processing services provided by EBRAINS. A component typically encapsulates the corresponding models, algorithms and logic of each offering in a self-contained and autonomous manner. This enables the management, improvement, and introduction of new capabilities without affecting the operation of other EBRAINS service offerings and the infrastructure. The technical means by which a new component can be integrated and deployed is standardised. This allows the streamlined introduction of new service offerings in the future to accommodate new research needs and/or critical scientific output, ensuring EBRAINS is sustainably and cost-effectively maintained at the leading edge of brain research. As the complete set of EBRAINS tools and services is very broad, the present section is not suitable to serve as a full reference. Below is a non-exhaustive category-list, mostly from the perspective of the EBRAINS High-Level Overview diagram.

- **Machine Learning & AI**: Central set of machine learning and AI tools, with well-defined APIs and integration with existing & future components.
- **Interactive Workflows**: Complex data-processing pipelines that allow for the orchestration and coordination of different tools and services simultaneously and enable the end-user to intervene during any execution stage to correct or change the required input parameters.
- **Virtual Machines & Containers**: This is to signify that EBRAINS platform provides Virtual Machine services and a Container Orchestration Engine to support any kind of analysis/simulation/data processing workloads.
- **Executables**: EBRAINS services provide a wide range of applications as downloadable executables that can be installed on local workstations by the end-users and leverage their research.
- **HPC modules/containers**: EBRAINS services provide a wide range of applications as HPC modules or packaged as container images that can be executed on HPC scalable/interactive computing resources.

2.3.2.3 Platform Middleware

This grouping contains components responsible for managing, allocating and monitoring the use of all underlying computing resources provided by the “Infrastructure - Hardware & IaaS/HPC” layer, affording flexibility, scalability, efficient use of resources, fault-tolerance and dynamic demand-aware provision of services to users. Furthermore, it provides a level of autonomy and independence from the specific vendor offerings (HPC, IaaS), abstracting their provision details, streamlining integration and deployment, supporting potential new vendor offerings, and ensuring that the required computing/deployment requirements of all EBRAINS components are met.

In preparation of SGA3, we have examined the plethora of different SGA2 software available, with a particular focus on their provisions for assembly and integration into a large-scale infrastructure (e.g. data flows, invocation methods, availability of APIs, runtime/scaling monitoring & requirements). A common observation emerged regarding the extremely limited horizontal instruments for standardising, monitoring, and managing inter-component communication, invocation and provenance tracking. In a typical large-scale system, such operations (message queues/micro-services) are handled by clearly defined/respected APIs. Considering the ‘technical debt’ of EBRAINS (i.e. technical decisions taken in past phases of the Project), as well as the nature of EBRAINS as a scientific infrastructure, the complete implementation of this paradigm is not advisable. Instead, we advocate a loosely-coupled integration model in which (a) large-scale components are served by, and abide to, the middleware for syntactic and semantic interoperability (e.g. strict API signatures, invocation lifecycle/monitoring) and (b) smaller components are packaged by the middleware under terms that allow their use by other components with minimal alterations. For example, the Knowledge Graph will be used as-is for managing data and models, with certain of its offerings (e.g. catalogue service, access to data, unique identifiers) repackaged and provided to other components (e.g. an analysis service being able to directly refer to KG assets). In the same example, the analysis service is packaged by the middleware (i.e. API, runtime orchestration) and slightly extended (e.g. standardised error output/execution logs).

The Platform Middleware comprises of the following core components:

- **Identity & Access Management**: A collection of services responsible for the identification, authorisation and management of EBRAINS users (e.g. single/advanced users, teams, organisations, communities) and their access rights in a highly granular manner (e.g. data assets, applications, services and quota), based on EBRAINS-wide policies reflecting the infrastructure business plan (e.g. SLAs and pricing models). It supports federated authentication providers (e.g. institutional, ORCID and EOSC), integrates with existing IAM layers of EBRAINS applications (e.g. Collaboratory), and mediates with Fenix/FURMS.

- **API Gateway Proxy/Central API**: This is responsible for the efficient communication of EBRAINS platform services. As such, it allows the disciplined, traceable and secure service invocation, data exchange, event handling, and infrastructure-wide monitoring of its operation.

  The API Gateway & Reverse Proxy provide a single point-of-entry (and therefore single point of administration & maintenance) for any APIs which are REST-on-HTTP conformant enough to be gateway-proxied, and for any web/HTTP resources which are appropriate for basic proxying. These are both provided by the same software stack, with the API gateway functionality (primarily in runtime-configurable plugin-form) built as a layer of functionality on top of the web reverse-proxy. This can be leveraged to offload much of the networking and similar overhead as well as the maintenance-burden from component-API developers to maintainers of the shared service for all APIs, enabling API developers to opt-in to simplifying their service so they can focus on their domain-logic. Non-API web-resources, generally using a subset of the HTTP functionality of REST, can similarly be reverse-proxied for many of the same reasons. This also allows multiple separate services to be exposed to consumers from within a unified subdomain or range thereof, and under unified operational requirements. This can encapsulate many or all of the APIs presented in the API catalogue and, as long as that information is robustly and consistently available, their exposure by the gateway and proxy could even eventually be auto-configured, as long as not blocked by misconfigurations of the source APIs (ACLs, credentials, CORS, etc). This can include both public and private (i.e. restricted) EBRAINS APIs, exposed via documented, flexible, and lightweight standards.

  The Central API is the home for any central or coordination endpoints which can't or won't be provided by any component, including low-level middleware and a "plugin-able" interface to high-level simulation/analysis workflows that components and end-users can utilise to orchestrate chains of calls to other component APIs, etc. This will also be an appropriate place for triggering automated regeneration of results based on provenance data ("research replay") when it is possible to automate such functionality.

  The API catalogue is intended to be provided as a metadata-source for searching, browsing and programmatically discovering all APIs in the EBRAINS RI, including component-provided APIs and the EBRAINS central API which was previously described. This ensures reusability, security, extensibility and full provenance/trust over the “who”, “why”, and “where” of each resource’s use. All EBRAINS-wide policies regarding software asset/service/resource-use are defined based on this information, while monitoring services similarly reference them, affording full auditing capabilities at any point in the infrastructure’s operation. This also supports the discovery of EBRAINS data assets and services from EOSC and OpenAIRE. Access to APIs, and corresponding utilisation of resources, can be fully monitored, traced and potentially charged (e.g. credits), according to EBRAINS SLA policies based on this metadata. The API Catalogue as it presently exists in its early incarnation[4] is a webUI for "interactive discovery". The current catalogue (both in implementation and content) is the result of dedicated effort from the EBRAINS API Working Group. Currently, there is ongoing discussion in the working group to discuss alternative options for a better choice regarding the hosting of the API catalogue.

- **Federation/Scalability/Resilience/Communication:** This component encapsulates a wide range of services that are found in the platform middleware that are (or will be) developed with the aim to exploit the federation of resources, boost scalability, satisfy resilience requirements and enable intra- and inter- communication between the various EBRAINS Services.

- **Standardised Workflow management**: This is a collection of different workflow engines (alternatively, workflow management systems) installed on both platform and underlying

[4] https://wiki.ebrains.eu/bin/view/Collabs/technical-coordination/Working%20Group%20APIs%20%28EA-WG%29/API%20Catalogue/

infrastructure's resources ranging from OKD clusters to HPC systems in order to execute and monitor computational scientific workflows that tap into the standardisation aspects, means and technologies proposed by Technical Coordination team (Section 3.2). Currently installed workflow engines (as well as planned ones) are compatible with Common Workflow Language open standard for defining scientific workflows in a common, well-defined and structured way and are responsible for load balancing when workflows are submitted, restarting of failed steps when errors occur as well as fetching logs and outputs.

- **Provenance tracking**: This is a centralised service offering a dedicated API for capturing, storing, and sharing provenance information enabling reproducibility across EBRAINS, including but not limited to: neuroscience models, configuration files, links to execution logs, test results, low level configuration, references to data objects, etc.
- **Metrics & Monitoring**: A unified service collecting, storing, sharing, and selectively informing in real time (e.g. to end-users) monitoring information from integrated EBRAINS components. Black-box metrics can also be provided automatically by API Gateway and Service Mesh (see section 3.5 for details).

2.3.2.4 Infrastructure - Hardware & IaaS/HPC

EBRAINS relies on this final and lowest layer to provide compute and storage resources. It comprises of (a) the storage and compute services provided by the Fenix RI with a guaranteed allocation to the EBRAINS infrastructure, (b) the external Extreme-scale HPC & Data Services, (c) the Neuromorphic Hardware, as well as (d) the External cloud services providers.

2.3.2.5 Other facilities

2.3.2.5.1 EBRAINS AISBL Operations & Administration

A non-exhaustive list of key organisational instruments/bodies critical for the development, delivery and sustained successful operation of the EBRAINS infrastructure is given below. These will be imprinted and detailed in the overall governance structure of EBRAINS. As apparent from the prescribed roles/descriptions, the development of EBRAINS will embrace Agile principles to increase the efficiency of collaborations/contributions, enable rapid prototyping ("Release Early, Release Often" (RERO)), ensure coherence at all levels (from overall vision to implementation details), absolute satisfaction of end-user requirements, and the timely, successful delivery of EBRAINS.

This grouping contains components responsible for the EBRAINS-wide management and monitoring of the RI, its services, users, and resources.

- **SLAs (Service Level Agreements) / Service Tiers/ Pricing**: Collection of provision methods and appropriate cost models for users with an integrated interface for administrators, also supporting high-level decision making by the EBRAINS AISBL.
- **Accounting:** This component collects measurements from all services and actively checks the SLAs / Service Tiers / Pricing against the specifications. It provides the appropriate tools to visualize and monitor these measurements to provide a holistic view of the usage by the end-users and the overall RI utilization of resources.
- **Persistent ID management**: This component handles digital references to data/model assets, contributors, or organizations, as a means of identifying a digital object regardless of changes to its location on the internet. The long-term persistence of identifiers is vital to robust data management strategies. Data/Model publishers and end-users exploit PIDs in their established research workflows to enable the creation of trusted digital connections between objects, contributors, and organizations.
- **Audit**: Exposes the entire body of provenance information assembled by EBRAINS (assets, models, workflows, protocols) to supervising ethics bodies in order to support the ad hoc and/or

continuous evaluation of ethics principles/ requirements/ methodologies required to be applied by EBRAINS end-users.

- **Security**: Collection of the required established processes that govern all aspects of the EBRAINS platform and ensure the overall high-level of security at all systems and services as well as compliance to all regulatory obligations.
- **Product Owner Panel**: Permanent panel comprising the EBRAINS Product Owner and CTO, as well as the corresponding Product Owners and Technical Leads of the various service offerings. Its decisions do not require a consensus. Decisions of the panel are informed by technical input from its members and information on evolving user requirements from the EBRAINS Scientific Panel and the EBRAINS High-Level Support Team. Decisions are taken by the EBRAINS Product Owner and implemented by the SCRUM Masters (if available) and Deployment & Maintenance Teams.
- **Deployment & Maintenance Teams**: Collection of all Teams responsible for the development, delivery, deployment, and maintenance of the EBRAINS RI and its components.

  The teams share source code repositories and issue tracking systems for all software expected to be developed/extended/reused/deployed by EBRAINS. This aims to reduce development and maintenance effort, increase code quality, and promote scientific (intra-SGA3) collaboration at the software development level.
- **Scientific Panel**: Permanent panel of scientific experts providing advisory input on end-user requirements and scientific issues to the Product Owner. Further, it comprises a separate User Panel of end-users involved throughout the development of EBRAINS (e.g. user stories, requirements, testing, piloting, exploitation).
- **Support services**: The High Level Support Team (HLST) provides a single point of contact and monitoring for all support requests, monitors evolving feature requirements, ensures that documentation is easily discoverable and that tutorial materials are available. The HLST will communicate observations from interactions with users to the relevant panels and teams, to facilitate a continuous improvement of EBRAINS services. The HLST will deliver a range of services to facilitate users throughout its interactions with the EBRAINS infrastructure, covering its service offerings in a horizontal and vertical manner (i.e. from a birds-eye view, to highly detailed domain/service-specific). The provision of these services through a homogenised, coherent interface serves to accommodate the full range of user experiences, from new users wanting to learn how they can tap into EBRAINS, to experienced scientists looking for the latest state-of-the-art offering to empower their research.

  The support services also include the Helpdesk along with the back end for the management of support tickets from EBRAINS users.
- **Outreach & Scientific Liaison**: Responsible for inviting users/organisations to EBRAINS, disseminating its service offerings, exploring new synergies/collaborations, and pursuing the established business model/revenue streams.

2.3.2.5.2 *Innovation Alliances*

External key policy and educational activities pursuing the democratisation of the EBRAINS output across the scientific community and society as well as the strategic partnership with other flagship research projects and commercial efforts.

- **Data Spaces:** This component ensures conformance to Open Access and FAIR data policy guidelines, streamlining the exposure and inclusion of EBRAINS-produced scientific output to fundamental data spaces like the EOSC (European Open Science Cloud) and other European Strategy Forum on Research Infrastructures (ESFRIs) towards a coherent and strategy-led approach to policy-making on research infrastructures in Europe, that facilitates multilateral initiatives leading to the better use and development of research infrastructures at EU and international level. This is considered of great importance now that EBRAINS has been included in the 2021 Roadmap of the ESFRI.

- **Digital Platforms**: This component ensures platform level interoperability with other digital platforms like Virtual Research Environment (VRE) at Charite University Medicine Berlin and OpenAire that enables Open Science implementation. It also enables the possibility of deployment of components of these platforms on the EBRAINS RI.
- **e-infrastructures**: Open-ended collection of aligned or complementary e-Infrastructures consuming data or services from EBRAINS and vice-versa.
- **Partnering Projects**: EBRAINS needs to be open to collaborating with other entities which are active in relevant research and infrastructure-building areas. This aids in avoiding duplication of effort and builds momentum behind the global effort to understand the brain and its diseases. Collaborations are sure to help EBRAINS to maximise use of the ICT Platforms and diffuse knowledge better.
- **Education**: Provides support to e-learning curricula and training courses to educate, train and inclusively engage scientists and students in EBRAINS offerings and output through also MOOCs (Massive Open Online Courses).

2.3.2.6 Sensitive Data

The dedicated full stack for Sensitive Data handling and applications exposes the EBRAINS facility for handling sensitive data securely. This demonstrates that EBRAINS is taking all considerations into account, collecting stakeholder’s requirements and implementing the necessary systems. It provides secure access to the front-end services, a sandboxed and secure data processing environment coupled with robust platform services and APIs enabling the encrypted sensitive data storage on the infrastructure used by the EBRAINS RI.

2.3.2.7 Applications Stack

The separate Applications Stack exposes the EBRAINS facility for offering hosting of full-stack applications within the EBRAINS ecosystem, alongside the core EBRAINS RI services, for seamlessly exploiting and connecting to them via well-known interfaces (Web APIs), thus taking advantage of all the scalability capabilities that are offered by the EBRAINS platform and the site-local acceleration of FAIR Data transfers.

Notable examples of full-stack applications that are installed on the EBRAINS platform are the Siibra Explorer (Brain Atlases Viewer), the HIP data platform, the Virtual Brain simulation Framework and the Neurorobotics platform. This emphasizes on the potential for both SGA3 consortium and external end-users, teams, research projects to bring their full-stack applications in the EBRAINS RI and expose them as offerings of this fundamental ecosystem for Brain Research.

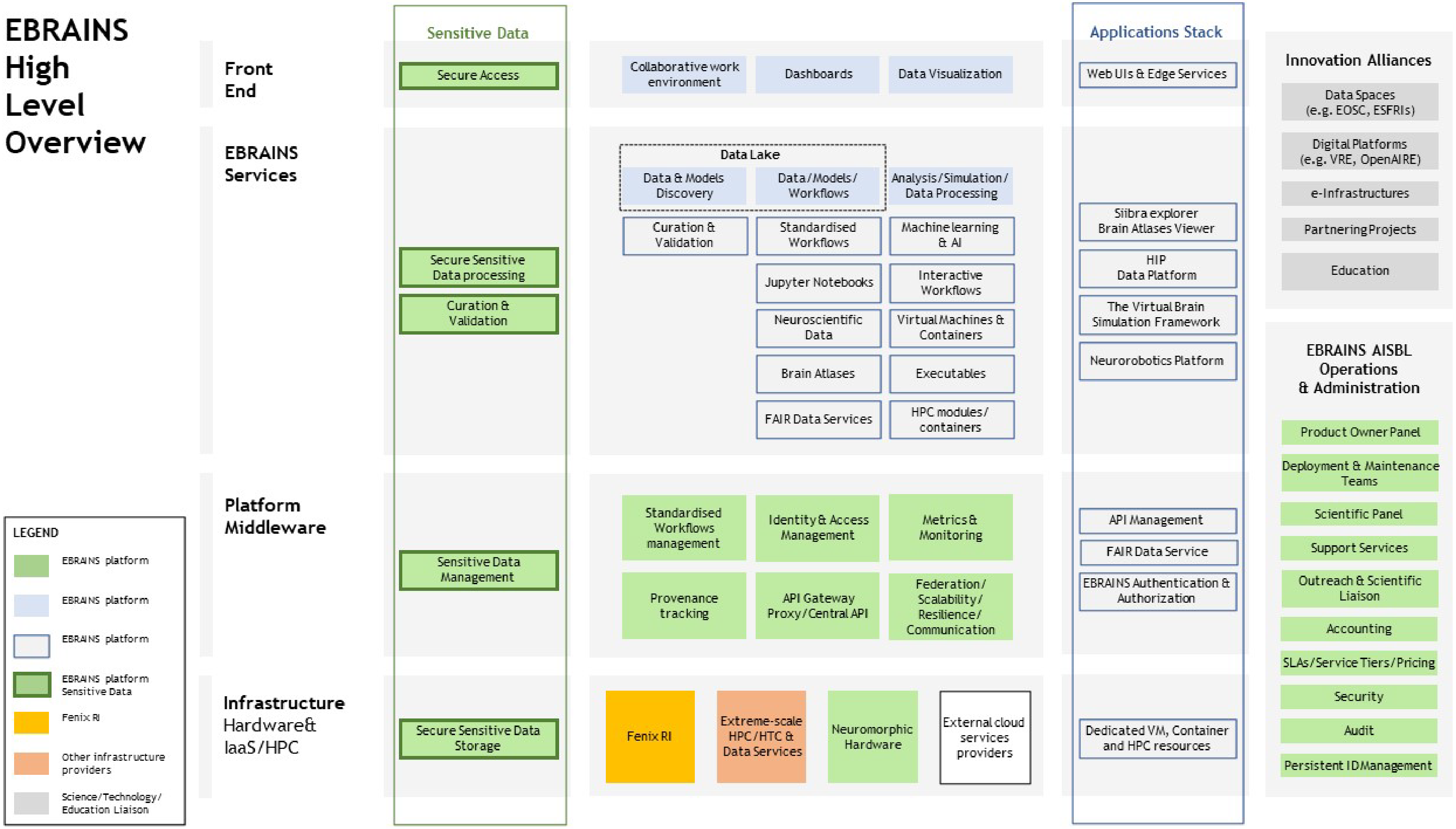

EBRAINS High Level Overview
Front End
EBRAINS Services
Platform Middleware
Infrastructure Hardware& IaaS/HPC
Sensitive Data
Secure Access
Secure Sensitive Data processing
Curation & Validation
Sensitive Data Management
Secure Sensitive Data Storage
Collaborative work environment
Dashboards
Data Visualization
Data Lake
Data & Models Discovery
Data/Models/Workflows
Analysis/Simulation/Data Processing
Curation & Validation
Standardised Workflows
Machine learning & AI
Jupyter Notebooks
Interactive Workflows
Neuroscientific Data
Virtual Machines & Containers
Brain Atlases
Executables
FAIR Data Services
HPC modules/containers
Standardised Workflows management
Identity & Access Management
Metrics & Monitoring
Provenance tracking
API Gateway Proxy/Central API
Federation/Scalability/Resilience/Communication
Fenix RI
Extreme-scale HPC/HTC & Data Services
Neuromorphic Hardware
External cloud services providers
Applications Stack
Web UIs & Edge Services
Siibra explorer Brain Atlases Viewer
HIP Data Platform
The Virtual Brain Simulation Framework
Neurorobotics Platform
API Management
FAIR Data Service
EBRAINS Authentication & Authorization
Dedicated VM, Container and HPC resources
Innovation Alliances
Data Spaces (e.g. EOSC, ESFRIs)
Digital Platforms (e.g. VRE, OpenAIRE)
e-Infrastructures
Partnering Projects
Education
EBRAINS AISBL Operations & Administration
Product Owner Panel
Deployment & Maintenance Teams
Scientific Panel
Support Services
Outreach & Scientific Liaison
Accounting
SLAs/Service Tiers/Pricing
Security
Audit
Persistent ID Management
LEGEND
EBRAINS platform
EBRAINS platform
EBRAINS platform
EBRAINS platform Sensitive Data
Fenix RI
Other infrastructure providers
Science/Technology/Education Liaison

Figure 2: EBRAINS High Level Overview

2.4 EBRAINS Conceptual Architecture

In this section, the Conceptual Architecture of EBRAINS (Figure 3) is presented. The 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]).

This view of EBRAINS architecture is the next available zoom level right after the High-Level Overview of the previous section and is based on the architecture section of the SGA3. It aims to better communicate the EBRAINS Services to internal end-users and others familiar with the EBRAINS ecosystem. Following the submission of SGA3, many improvements have been made in a structured manner, resulting in the current version that is presented in this section. High-level abstractions are favoured to facilitate communication, portray a common shared vision and address past negative feedback. The aim of the architecture presentation is to function as a shared understanding between science practitioners and infrastructure experts.

Section 2.4.1 introduces the conceptual architecture design principles. Section 2.4.2 provides the next level of detail for the components of the EBRAINS infrastructure. It is worth mentioning that Section 2.4.2 describes in detail only the boxes of the Conceptual architectural diagram presented in Figure 3 that were not detailed in the High-Level Overview (Section 2.3).

2.4.1 Conceptual Architecture design principles

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 offering, but rather the types of components that one or more offerings 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).

There are two overarching design principles behind the EBRAINS architecture:

- **Loosely-coupled**. Given the high degree of heterogeneity in the underlying technologies implementing the various current EBRAINS service offerings, the need to minimise development and integration effort, the evolving requirements of the scientific community in terms of services to support excellent research, as well as best practices in the design of similar large-scale scientific infrastructures, we have opted for loosely-coupled integration, where components adhere to a core set of principles: (a) they are described and invoked via self-describing programmatic APIs abstracting their internal implementation and operation methods, (b) their inputs are provided by EBRAINS automatically (e.g. data, configuration parameters, resources) in a uniform manner according to user needs/privileges and computing resources, (c) their outputs (e.g. analysis results, models, logs) are handled by EBRAINS, ensuring that they are deposited and become available to the required client interface or subsequent component.

- **Scalable cost-effective deployment & orchestration.** The physical deployment and operation of EBRAINS will be supported mainly by the IaaS cloud and HPC services of Fenix RI. Where possible, components will be instantiated as applications/services within its IaaS offerings, while computing capabilities will be provided either as scalable computing or interactive computing services. Allocation of computing resources, instantiation of the required components, data handling operations, as well as the implementation/orchestration of the corresponding processing workflows, will be performed via resource management components within EBRAINS, ensuring that all service offerings and internal operational components scale cost-effectively, while also affording a level of independence and fault tolerance from the underlying multi-site HPC offerings of Fenix RI.

Detailed explanations of the EBRAINS components illustrated in Figure 3 are presented in the following section. These components have been derived from an extensive collaborative analysis of (a) of the requirements and planned offerings of all SGA3 Science WPs, SCs, and (b) the current and anticipated needs of EBRAINS end-users.

Please note that the specifications of the individual components, their capabilities and interfaces will be constantly adapted according to progress in neuroscience and the needs of the users.

2.4.2 Architecture components

2.4.2.1 Front End

Further detailing the Front-End categories of EBRAINS:

- As mentioned, the Collaborative work environment is the core entry-point for the discovery and invocation of EBRAINS services. It comprises of two components. The Collaboratory and Jupyter Notebooks.
  - **Collaboratory**: The Collaboratory offers researchers and developers an environment to work in teams and share their work with users, teams or the Internet. Workspaces in the Collaboratory are known as collabs.
  - **Jupyter Notebooks**: Jupyter is a web-based, interactive computational environment for creating Jupyter notebook documents, which is the *de facto* cross-domain standard for interactive and scalable exploratory data interaction and analysis. EBRAINS will provide a series of general-purpose and specialised workspace instances in several programming languages and frameworks, which will externalise EBRAINS service offerings, making them simple to use for educational and scientific purposes.
- Dashboards
  - **Workflows Dashboards**: A dedicated dashboard for the workflows management. It covers the entire life-cycle of this particular object from creation, submission, sharing, deployment, execution and monitoring in all stages of execution as well as tracking of processing output.
  - **Component Specific UIs**: This refers to all the UIs that have been developed in the specific context of a component and enable access to that particular component's service providing a particular view of the EBRAINS RI. We are highlighting the fact that there are components that provide a front-end that allows for sensitive data access.
  - **Personal Workspace**: This refers to the Web UI that gives horizontal access to all the assets a user owns or has access to in the EBRAINS RI as well providing detailed information of all the platform and infrastructure resources the user has access to, along with the respective usage and utilization statistics.
- Data Visualization
  - **Knowledge Graph UI:** The external user interface of the EBRAINS Knowledge Graph is the point where users discover the full breadth and depth of the highly curated, growing, and valuable EBRAINS data assets, including links with relevant publications (i.e. how they have been produced, how they can be used) and EBRAINS services/workflows (i.e. where and how to use them). With the planned extensions in the tiered curation processes, as well as linking with OpenAIRE and the broader EOSC ecosystem, the EBRAINS Knowledge Graph UI will support three modes of discovery, further lowering the entry barrier for end-users: (a) data-driven (first discover relevant assets, and then publication and services), (b) publication-driven (first discover relevant publications and then the corresponding data assets used/produced and services), (c) service-driven (first discover an EBRAINS service offering and then corresponding data assets and publications).
  - **Viewers:** A collection of specialised comprehensive, stand-alone, web-based, domain-specific visualisation and manipulation applications and frameworks. Complemented with

scalable visualisation components that can be used “as-is” or embedded into internal or third-party systems and applications.

2.4.2.2 EBRAINS Services

The various EBRAINS Services in greater detail are the following:

2.4.2.2.1 Data & Models Discovery

These comprise a collection of internal and external components and workflows that implement EBRAINS curation workflows for data assets, methods and models. Regardless of the type of process supported (e.g. expert-driven manual curation), their physical deployment (e.g. within external non-ICEI sites), or level of automation (i.e. from entirely manual, to fully automated), all curation workflows are extensively monitored and documented, to ensure their reproducibility.

- **HBP Facility Hubs**. A new concept introduced with SGA3, to significantly enlarge the capabilities available on the EBRAINS RI and provide an added value for the scientific community. HBP Facility Hubs will open the opportunity for researchers Europe-wide to use some of the most advanced resources belonging to the HBP Partners for their own research projects, to start collaborations in this field and to become competitive.
- **Extract Transform and Load**. This includes an extensible suite of standard extract-transform-load (ETL) processing operations, including multiple types and data asset sources. Operations can be assembled into complex ETL pipelines (invoked manually or automatically), as well as broader EBRAINS-wide workflows as needed (e.g. on the fly transformation of data in a specific format for download, pre-processing data to prepare the input for a given workflow). We are highlighting the fact that sensitive data pre-processing requirements are taken into consideration.
- **Data & Model / Method Curation workflows**: This comprises an extensible collection of formalised data and model/method curation workflows, with their final output being data assets, for which metadata is forwarded to the Knowledge Graph for indexing, management, linking, and provisioning. We are highlighting the fact that sensitive data curation and validation is foreseen.

2.4.2.2.2 Data/Models/Workflows

The Data / Models / Workflows is one of the most important components of EBRAINS, being responsible for the sustained, organised, traceable and streamlined provision of data assets, models/methods and workflows descriptions as digital assets to all other EBRAINS components.

- **Knowledge Graph (KG)**: This comprises comprehensive tools and services for publishing FAIR data and computational models. The services provide long term data storage, citable DOIs, defined conditions and licenses for use of data and tags to make the data discoverable, interpretable, and re-usable. The actual storage of datasets is provided by the Fenix RI (Infrastructure - ‘Hardware & IaaS/HPC’ layer). The EBRAINS Knowledge Graph encompasses the following sub-components:
    - **Data & Metadata**: Collection of metadata models inside Knowledge Graph for researchers to find and share FAIR (Findable, Accessible, Interoperable, Reusable) data, models, and to associate them with software for their research.
    - **Provenance Metadata**: Comprehensive collection of metadata capturing in full detail the provenance of data assets, methods, and models created within EBRAINS. Among other things, captured metadata pertains to input/output files, software version, environment under which computation was run, agent or person who started the run, hardware system, and configuration files. Different use cases within EBRAINS may need further information to be captured for the overall process to be reproduced (reproducibility) – note that the set of described functionalities is currently not fully implemented.

  - **Models Catalogue:** Comprehensive models catalogue that contains all models produced by scientific research performed in the context of SGA3 but open to contributions from the neuroscientific community. FAIR models (validated & non-validated) available to be linked with other digital assets and ready to be used in processing pipelines.
  - **Service & Software Catalogue & Core software modules:** Comprehensive services and software catalogue, of offerings both within EBRAINS and community/industry available neuroscientific tools with access and documentation links.
  - **Standardised Workflows Catalogue:** A comprehensive catalogue of all workflow descriptions (recipes) produced by scientific research, offered as digital assets. These recipes along with information produced in the workflows' post-execution stages will be linked with Knowledge Graph for making the neuroscientific work easily findable, accessible and reproducible.
- **Brain Atlases**: Brain atlases are the entry point for finding and analysing data based on a particular location in the brain. Users can find brain data in a 3D spatial framework for easy comprehension, comparable to the way Geographical Information Systems organise data in 2D maps of the surface of the Earth.

2.4.2.2.3 Analysis/Simulation/ Data Processing

This grouping contains components encompassing all current and foreseen Analysis, Simulation & Data Processing services provided by EBRAINS. A component typically encapsulates the corresponding models, algorithms and logic of each offering in a self-contained and autonomous manner. This enables the management, improvement, and introduction of new capabilities without affecting the operation of other EBRAINS Services and the underlying infrastructure. The technical means by which a new component can be integrated and deployed is formalized. This allows the streamlined introduction of new services in the future to accommodate new research needs and/or critical scientific output, ensuring EBRAINS is sustainably and cost-effectively maintained at the leading edge of brain research. As the complete set of EBRAINS tools and services is very broad, the present section is not suitable to serve as a full reference. Here is a non-exhaustive list of categories. Future components not matching these categories are also supported.

- **Machine Learning & AI**: Central set of machine learning and AI tools, with well-defined API and integration into existing components.
- **Brain Imagery Analysis**: Component for post-processing and segmentation of data stored in the Brain Atlases and other Data/Models/Workflows resources. The output of this component is in a format ready for consumption by the model simulation.
- **Data pre-processing**: Computing service for pre-processing large data sources before registration in the Knowledge Core. The computing part of the curation process.
- **In-transit Analysis**: This is for in-transit analysis or reprocessing of simulation output. Data will be streamed between simulator or reader possibly running on different compute nodes. Optionally with interactive support for the end users.
- **Embodied AI**: A (simulated) robot framework, integrated using a common API to the existing (multiscale) simulation components. This is deployed in a reproducible, concurrent and scalable fashion. Optionally with interactivity from the end users and in-transit analysis.
- **Multiscale simulation**: This is a reproducible, concurrent and scalable deployment of simulations at different scales. They run on HPC or Fenix resources and communicate via high-performance APIs with each other, in transit analysis and visualisation.
- **Robotics**: This refers to the Neurorobotics platforms installed on the EBRAINS platform that offers a simulation platform which enables users to choose and test different brain models for their robots. It also allows the use of multiple neural simulators and AI frameworks (e.g. NEST, TensorFlow) to implement a wide range of brain simulations (e.g. spiking neural networks) and controllers.

- **Interactive workflows**. Standardised, integrated workflows, implementing WP-specific functionality. They are implemented and supported by the WPs, typically combining specific sets of sub-components in this grouping, in task-specific chains.

2.4.2.3 Platform Middleware

As presented in the High-level overview (Section 2.3), this grouping contains components responsible for managing, allocating and monitoring the use of all underlying computing resources provided by the Storage and Compute component, affording flexibility, scalability, efficient use of resources, fault-tolerance and dynamic demand-aware provision of services to users. Furthermore, it provides a level of autonomy and independence from the specific vendor offerings (HPC, IaaS), abstracting their provision details, streamlining integration and deployment, supporting potential new vendor offerings, and ensuring that the required computing/deployment requirements of all EBRAINS components are met.

The Platform Middleware's components were mostly detailed in the High-Level Overview. In this increased zoom level, we also observe the **Orchestrator component** which includes the development and deployment of a central Orchestrator that is responsible for the execution and monitoring of (a) system-wide workflows related to the deployment, initialisation, computation, and delivery of all components required for the sustained operation and provision of EBRAINS services, as well as (b) the seamless execution and monitoring of complex workflows involving an ad hoc assembly of EBRAINS services and applications. For example, an ETL workflow from the Data Curation component is instantiated and executed by allocating the appropriate resources defined in the workflow component (e.g. number of nodes, target processing environment). In another example, the scalable execution of multiscale simulation, is similarly instantiated by the workflow component, which allocates the corresponding HPC resources for the specific execution setting and user. The Orchestrator is also capable of prioritising and monitoring the execution of VMs, containers and jobs, enforcing policies for maximising the efficiency in using the available computing resources, and in accordance with the required SLA per given user/service offering.

It also important to mention the sensitive data management capabilities that are offered from dedicates services via respective Web APIs.

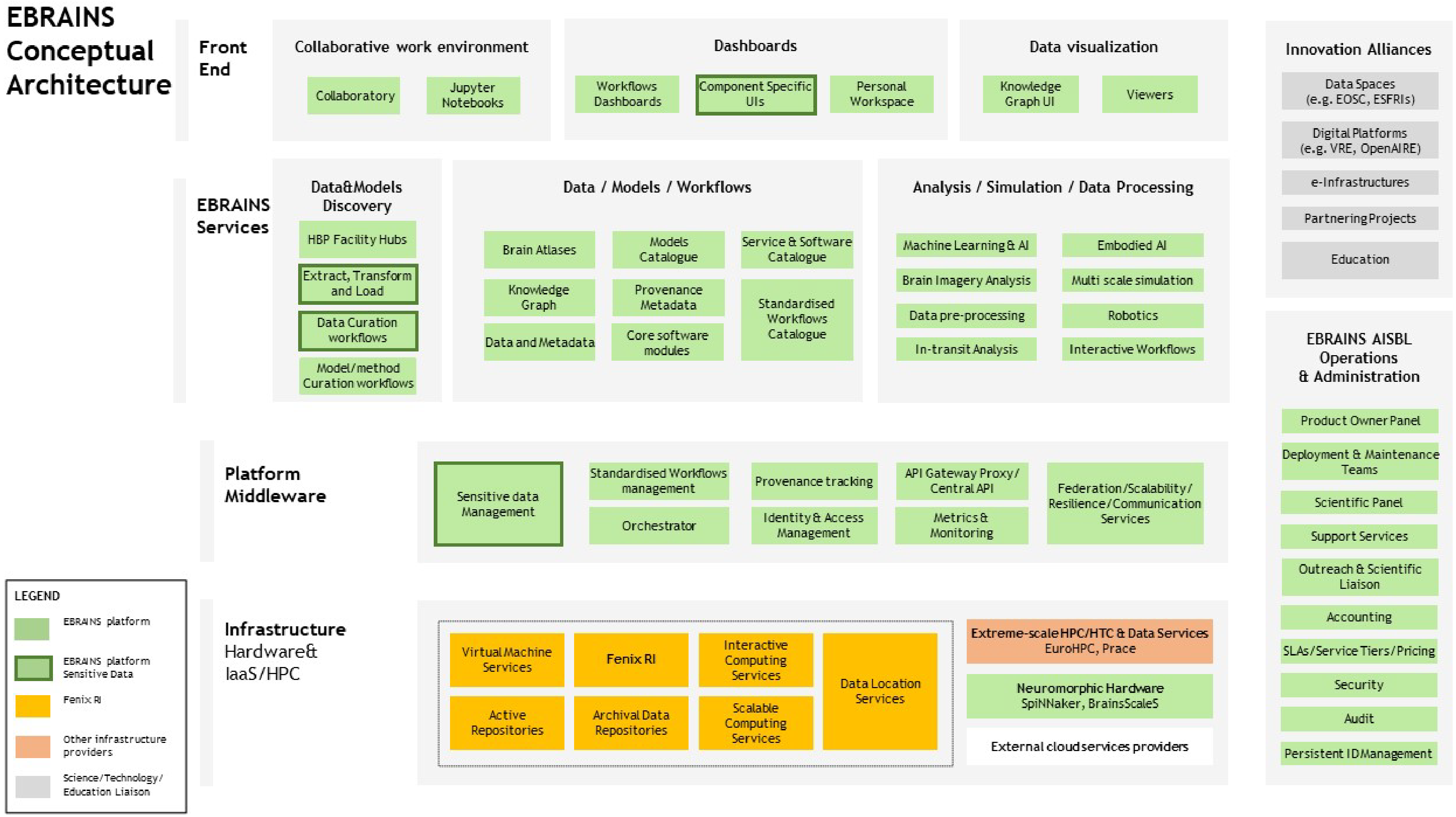
2.4.2.4 Infrastructure - Hardware & IaaS/HPC

EBRAINS relies on this final and lowest layer to provide compute and storage resources. It comprises of (a) the storage and compute services provided by the Fenix RI with a guaranteed allocation to the EBRAINS infrastructure, (b) the external Extreme-scale HPC & Data facilities provided on a grant basis via EuroHPC, Prace or potentially other 3rd parties, (c) the Neuromorphic Hardware provided by the HBP Facility Hubs, SpiNNaker and BrainsScaleS, as well as (d) the possibility for exploiting resources from External cloud services providers for non-critical, non-sensitive data processing workloads.

- **Fenix RI**
    - Virtual Machine Service: Service for deploying virtual machines in a stable and controlled environment, suitable for deploying platform services like the Collaborative work environment, EBRAINS information catalogues, image services, etc.
    - Data Location Service: Central service supporting basic functionalities for discovery of the physical location where data objects are stored.
    - Scalable Computing Services: Parallel HPC systems for scalable applications with possibly elastic access.
    - Interactive Computing Services: High-end servers with high-bandwidth interconnect to Scalable Computing Services and to Archival and Active Data Repositories for interactive data processing using, for example, frameworks like Jupyter Notebooks.

  - Active Repositories: Site-local data repositories located close to computational and/or visualisation resources and used for storing temporary slave replicas of large data sets.
  - Archival Data Repositories: Federated data store optimised for capacity, reliability and availability and used for long-term storage of large data sets that cannot be easily regenerated.
  - Other Fenix Services: AAI, Internal/External Network Services, Fenix User Management Service, Infrastructure monitoring services.

- **Extreme-scale HPC & Data Services**: Tiers 0 and 1 access to large HPC facilities; resources are typically acquired via European or national grants. For instance, EuroHPC, Prace or other 3rd parties.
- **Neuromorphic Hardware**: Neuromorphic hardware services provided by the corresponding HBP Facility Hubs SpiNNaker and BrainScaleS.
- **External cloud services providers**: Access to external cloud services providers to utilize resources typically acquired via European or national grants or possibly through partnerships via commercial cloud providers.



EBRAINS Conceptual Architecture

Front End

- Collaborative work environment: Collaboratory; Jupyter Notebooks
- Dashboards: Workflows Dashboards; Component Specific UIs; Personal Workspace
- Data visualization: Knowledge Graph UI; Viewers

EBRAINS Services

- Data&Models Discovery: HBP Facility Hubs; Extract, Transform and Load; Data Curation workflows; Model/method Curation workflows
- Data / Models / Workflows: Brain Atlases; Models Catalogue; Service & Software Catalogue; Knowledge Graph; Provenance Metadata; Standardised Workflows Catalogue; Data and Metadata; Core software modules
- Analysis / Simulation / Data Processing: Machine Learning & AI; Embodied AI; Brain Imagery Analysis; Multi scale simulation; Data pre-processing; Robotics; In-transit Analysis; Interactive Workflows

Platform Middleware

Sensitive data Management; Standardised Workflows management; Orchestrator; Provenance tracking; Identity & Access Management; API Gateway Proxy/ Central API; Metrics & Monitoring; Federation/Scalability/ Resilience/Communication Services

Infrastructure Hardware& IaaS/HPC

Virtual Machine Services; Fenix RI; Interactive Computing Services; Data Location Services; Active Repositories; Archival Data Repositories; Scalable Computing Services

Extreme-scale HPC/HTC & Data Services EuroHPC, Prace

Neuromorphic Hardware SpiNNaker, BrainScaleS

External cloud services providers

Innovation Alliances

Data Spaces (e.g. EOSC, ESFRIs); Digital Platforms (e.g. VRE, OpenAIRE); e-Infrastructures; Partnering Projects; Education

EBRAINS AISBL Operations & Administration

Product Owner Panel; Deployment & Maintenance Teams; Scientific Panel; Support Services; Outreach & Scientific Liaison; Accounting; SLAs/Service Tiers/Pricing; Security; Audit; Persistent ID Management

LEGEND

- 🟩 EBRAINS platform
- 🟩 EBRAINS platform Sensitive Data
- 🟨 Fenix RI
- 🟧 Other infrastructure providers
- ⬜ Science/Technology/ Education Liaison

Figure 3: EBRAINS Conceptual Architecture

2.5 EBRAINS Logical Architecture

In this section the Logical Architecture of the EBRAINS RI is presented. The Conceptual Architecture (see Figure 3) 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 natural exercise 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 the current SGA3 software components in a single diagram.

Following the paradigm of the previous diagrams, the EBRAINS Logical Architecture displayed in Figure 4 and Figure 5 is abstracted into four layers. Starting from the top layer:

- **Front end**. This comprises a collection of front-facing, end-user interfaces and services, enabling the discovery, use and management of the EBRAINS infrastructure by end users and EBRAINS personnel.
- **EBRAINS Services**. This encompasses the scientific and innovation output of HBP SGA3 WPs and SCs.
- **Platform Middleware**. This contains all components responsible for managing, allocating, and monitoring the use of all underlying computing resources provided by “Infrastructure Hardware & IaaS/HPC” layer, affording flexibility, scalability, efficient use of resources, fault-tolerance and dynamic demand-aware provision of services to users.
- **Infrastructure** - Hardware & IaaS/HPC. This contains: (a) the storage and compute services provided by the Fenix RI, with a guaranteed allocation to the EBRAINS infrastructure; (b) the external Extreme-scale HPC & Data facilities provided on a grant basis via EuroHPC, Prace or potentially other 3rd parties; (c) the Neuromorphic Hardware provided by the HBP Facility Hubs, SpiNNaker and BrainsScaleS; as well as (d) External cloud services providers that can be used to offload non-sensitive workloads (data processing and other non-neuroscience-specific cloud processes).

The EBRAINS platform inherently supports:

- **Sensitive data** applications as it provides the required components for sensitive data handling.
- Hosting of the **full stack for applications** and services deployed alongside the core EBRAINS Services.

For envisioning the Logical Architecture our efforts were concentrated on mapping all SGA3 components into a logical view of EBRAINS by outlining the way components are relating/interfacing with each other, and to their respective processing levels.

At the start of SGA3 an important objective pursued by the EBRAINS TC was to identify, disambiguate and set up a comprehensive, authoritative and always up-to-date knowledge base of the EBRAINS RI components.

The immediate follow-up exercise to this task of identifying all the individual components in the Knowledge Base 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 the previously identified components.

All SGA3 Partners concerned were contacted individually to contribute the detailed architectural logical diagrams of their components to provide a solid starting point that would kickstart our work towards designing an accurate logical architecture of the RI in the context of the EBRAINS TC and the EAI-WG activities.

Past and current HBP architecture mapping efforts, technical presentations of the involved platforms, systems and services, along with detailed exploration/navigation of Collaboratory 1 and 2 platforms, resulted in the collection of the material required to elaborate a concrete logical architecture design. By considering all the aforementioned material and keeping it in sync with current developments, changes and efforts of the project, a draft version of the logical architecture of EBRAINS RI was compiled by the TC team and presented for further discussion and refinement by the EAI-WG. Constant iterations have resulted in the diagrams presented in Figure 4 and Figure 5, which will remain live diagrams as iterations will continue until the end of SGA3.

In Figure 4, 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.

In Figure 5, we go one step further from the Figure 4. Figure 5 places all EBRAINS components (for further details see D5.3, Section 3.7, [1]) 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.

This diagram may appear overloaded, but it consists of only EBRAINS components and identifies the EBRAINS Front Ends, Services, Platform middleware and Infrastructure components. EBRAINS Services can be further decomposed in three layers, namely, (a) the Visualisation / Validation / Analysis of Activity Data, (b) Model creation engines, and (c) Simulation engines.

The diagram can be approached in various ways and each one identifies a different aspect of the RI.

One way to approach the diagram is **top-down**. In most cases, upper levels communicate with lower levels to establish workflows and deliver results. Connecting arrows were deliberately not designed as one component can interface with others in many ways and at different processing stages and this would increase unnecessarily the complexity in such an already dense diagram.

Another way to approach the diagram is from **left to right**. By observing the “Simulation Services” box for example, components at the same layer have (mostly) the same scope. It is key to note that, for the Simulation engines, Model creation engines and the Platform Middleware layers, components belonging in a particular layer can interact with other components in the same layer in a number of ways to create simulation workflows.

All EBRAINS Service Categories (SCs), namely Data & Knowledge, Atlases, Brain-Inspired Technologies, Simulation, Community and Medical Data Analytics, are also clearly visible with dotted line boxes denoting the **scope of each SC** and the components that belong to it. Components from different service categories can interface with each other in a number of ways (through APIs, data services, etc.) and complex science workflows can be configured and executed as a result.

Component boxes are **coloured**, based on the available deployments that were identified for each component (please note that current status may have since diverged from collected information that resulted in the present diagram). The deployments can be on OpenStack Virtual Machines, on OKD container orchestration platform, as Desktop applications, as HPC libraries and, finally, as Jupyter notebooks.

This section aims to give a wider view of EBRAINS RI and display the current state of available products/services to provide a collective overview of EBRAINS offerings. For further analysis of the internals of each SC, information can be found in Section 2.7. For further information about the Deployment types available at EBRAINS see Section 2.8. Information about the Physical deployment of EBRAINS and how it enables the underlying infrastructure can be found in Section 2.6.

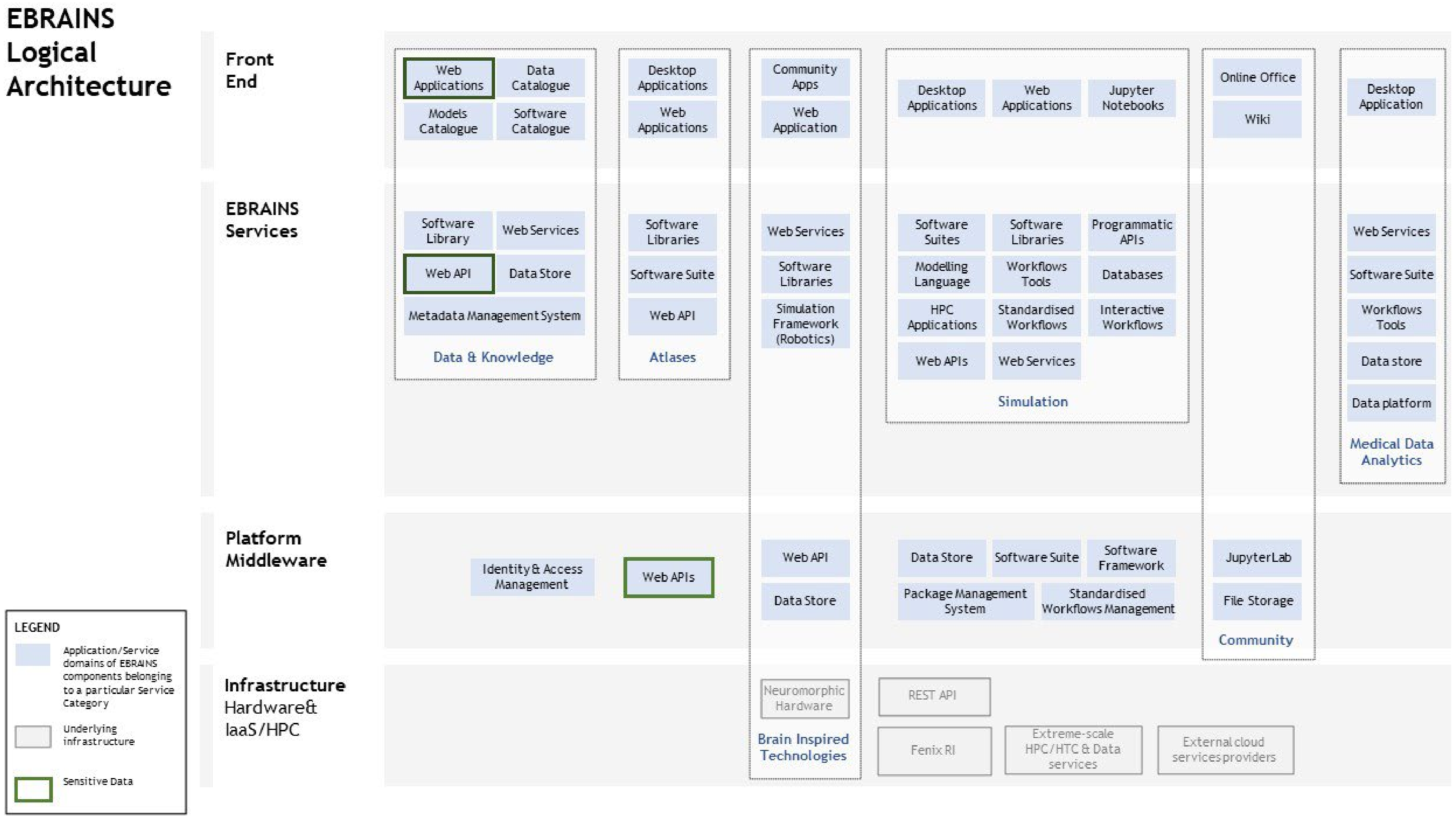


Figure 4: EBRAINS Logical Architecture

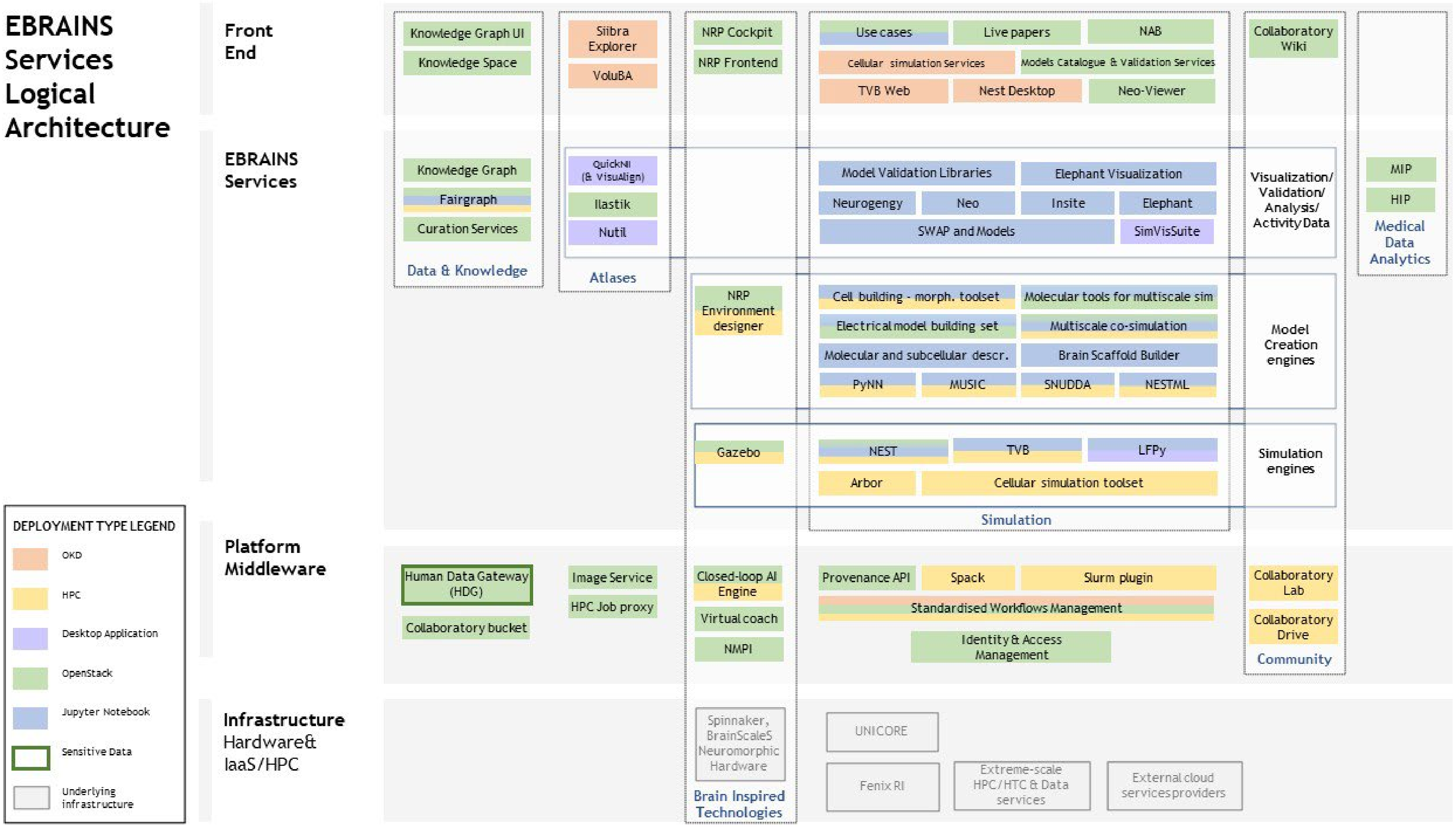


Figure 5: EBRAINS Services Logical Architecture

2.6 EBRAINS Physical Deployment

Following the presentation of “EBRAINS Logical Architecture”, the next immediate step is the elaboration of the EBRAINS Physical deployment. It is quite a challenging endeavour, as one must drill down from the EBRAINS logical architecture and identify what exactly is in place for each component and where it is located. Physical deployment diagrams are of great value, as they reveal the layout of the RI, provide deep understanding of the EBRAINS internals, enable architects and operators to have an overview of the current status, identify missing points/functionality and design new features, strategies and deployments. The task of elaborating and updating the Physical Deployment diagrams had an extra layer of complexity due to EBRAINS federation (EBRAINS is a distributed platform) and the fact that many base infrastructure facilities are still slowly coming online to full capacity. There are some sites that have almost all of their infrastructure resources readily available, some that have a good percentage of their resources available and some that are in the final stages of making available the full extent of their offerings. The need to exploit the resources available to EBRAINS from Fenix RI is self-evident, as otherwise the SGA3 Consortium would not be able to efficiently and effectively use the Fenix RI guaranteed resources allocation for EBRAINS. In a nutshell, the EBRAINS RI must be able to fully utilise the dedicated resources and operate in a multi-site environment in order to be able 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).

The following physical deployment diagrams present: i) the **status** of EBRAINS physical deployment as it had been determined during EBRAINS Phase1 (Section 2.6.1), ii) the **current status** of EBRAINS Physical Deployment as it is towards the end of EBRAINS Phase 2 (Section 2.6.2) and iii) the **projected state** of physical deployment as it is envisaged for the final phases (Section 2.6.3). Information that supplemented their 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 [3], ICEI available resources [4], 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 idea was to present the different available facilities (different locations) for EBRAINS as distinct slots and to display in each slot the installed/available platform/infrastructure services. External services as well as basic/core APIs are also indicated.

Each slot follows a **vertical stack approach**.

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.

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

Finally, it is worth mentioning that UNICORE/X is the server component providing the APIs for HPC access and data-movement (currently) via UNICORE TSI. Fenix sites are responsible for the UNICORE/X and UNICORE TSI installations because there is no secure and robust way to delegate that to external admins. UNICORE services are deployed on VMs and physical servers under the respective site's control.

2.6.1 Phase 1

In this section, the status of the Physical Deployment Diagram is presented, as it was illustrated in Figure 6 during EBRAINS Phase 1. It can be clearly observed that **not all Fenix RI sites were yet fully operational**, as more of the base infrastructure services were coming online gradually. As such, **CSCS** was considered as the de facto main site (supported more workloads and the base infrastructure services are all online) and **JSC** was considered the de facto secondary site.

Most of EBRAINS platform middleware services were installed on **only one site** and as a result key platform and simulation services were able to **run at only one location**.

The majority of the EBRAINS federation was accomplished using UNICORE for cross-site functionality and the EBRAINS IAM (Keycloak) for authentication, authorisation, and user registry handling.

2.6.2 Phase 2

In this section, the status of the Physical Deployment Diagram is presented, as it was illustrated in Figure 7 towards the end of EBRAINS Phase 2:

- This showed that some sites had almost all infrastructure resources readily available, some had a fair percentage available, and some were in the final stages of providing their full offerings.
- It also became clear that CSCS was being used as the de facto main site (supporting a better range of workloads, with the base infrastructure services all online) and JSC was being used as the de facto secondary site.
- Some of the EBRAINS platform middleware services became available on a secondary site as well. Preparation for installation of more platform middleware services to multiple sites is an ongoing task, with much of the progress expected to be during the early stages of Phase 3.
- Advancements also happened with respect to sensitive data, with the Human Data Gateway (HDG) component now available for access to sensitive data through the platform middleware.
- A certain number of EBRAINS Services started to employ multi-site deployments, with clear indications that most EBRAINS components are due to follow soon, as they reach a mature development status.
- UNICORE was installed at the UHEI site meaning that Neuromorphic Computing users can now also access the BrainScaleS resources through the UNICORE API service.

- Infrastructure monitoring was extended and now covers all Fenix RI sites, something that will prove vital to the extension of the platform middleware services at all locations.
- The majority of the EBRAINS federation requirements to-date were accomplished using UNICORE for cross-site functionality and the EBRAINS IAM (Keycloak) for authentication, authorisation, and user registry handling.

2.6.3 Projected Physical Deployment

In this section, the projected state of physical deployment, as it is envisaged for the final phases, is presented. The Projected for Phase 4 Physical Deployment Diagram can be seen in Figure 8. The goal is **for all sites to be fully operational** (all base infrastructure services online) with Accounting and Fenix AAI (federation services) running. All EBRAINS core services including Monitoring services (both internal and external) are to be online in order to support the EBRAINS DevOps team.

The federation should be transparent (no main/secondary sites) so that all workloads/services have the foundation to be deployed and scaled anywhere in the federation. Critical EBRAINS applications as well as platform middleware are intended to be “Highly Available” for fail-over and load-balancing purposes.

EBRAINS-wide federation will be further developed and enhanced by use of technologies such as DNS-LB (DNS-based load balancing), API gateways (which can also provide lower-level reverse-proxying, load-balancing, some service-mesh functionality, etc). Attempts will be made to leverage OKD not only as clusters on multiple sites, but ideally supporting multi-cloud functionality across sites. If cross-site redundancy can be achieved (at infrastructure level, at the application-level, or a hybrid of both) then, in coordination with DNS-LB/dynamic-DNS and health-check daemons, it could be possible to support a degree of geo-redundant and geo-optimised traffic-handling, which could treat Fenix sites as an opaque, latency-optimised resource. Careful use of methods like “sticky sessions”, would be able to allow even dynamic traffic to be optimised for geographical distribution across Fenix in a fashion similar to how CDNs cache static assets “at the edge” closer to a user. Even if this is achieved to any degree, there are expected to always be use-cases where targeting of specific sites or specific countries will be required, so such things would have to include opt-in/opt-out “trapdoors” for site-selection.

As a sidenote, the Projected Physical Deployment diagram provides an idealised view of the envisaged architecture. There may be parts that will be different from what the diagram specifies, as some things may be revised or considered not efficient or will be given a priority that is currently undetermined.

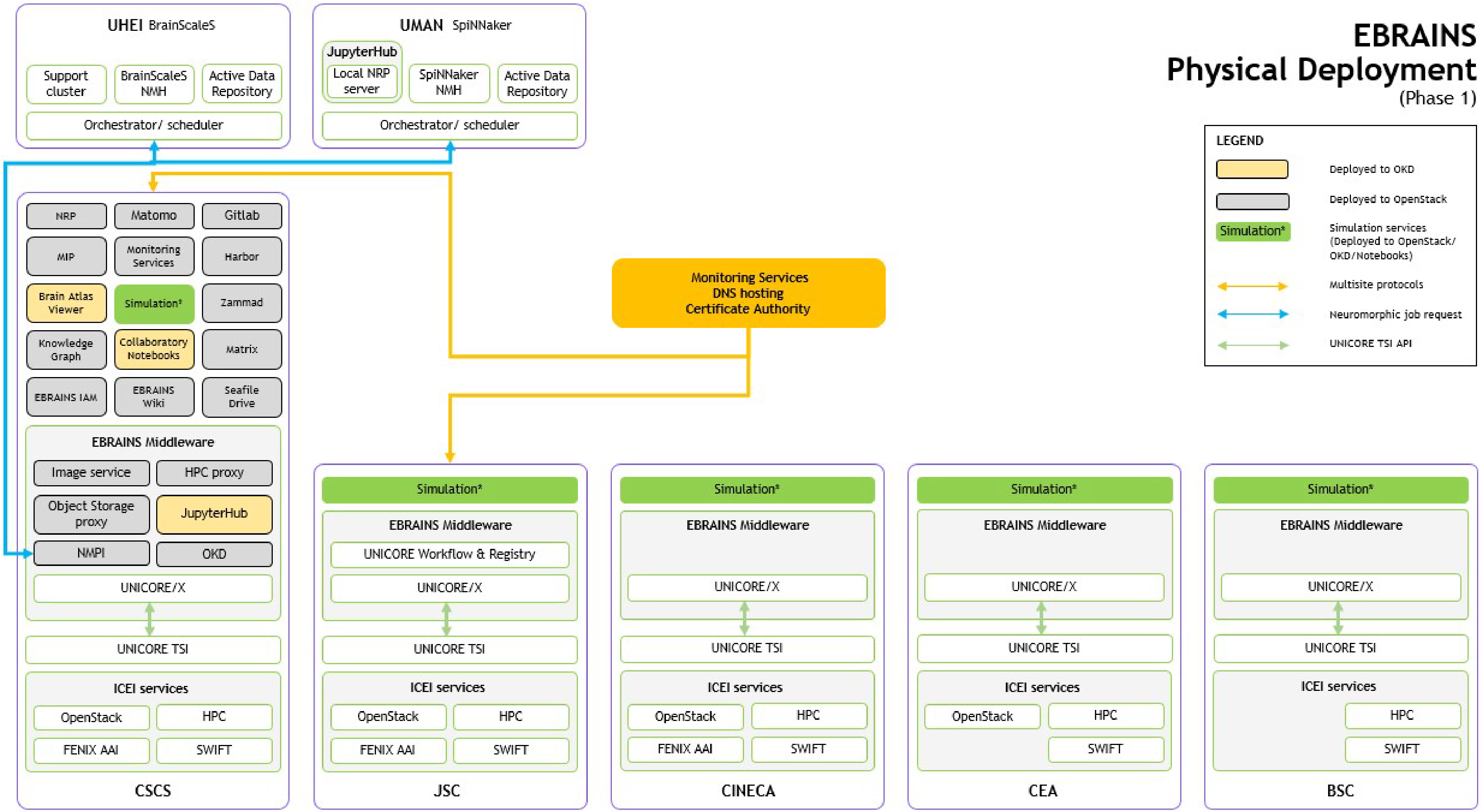


Figure 6: Physical Deployment Phase 1

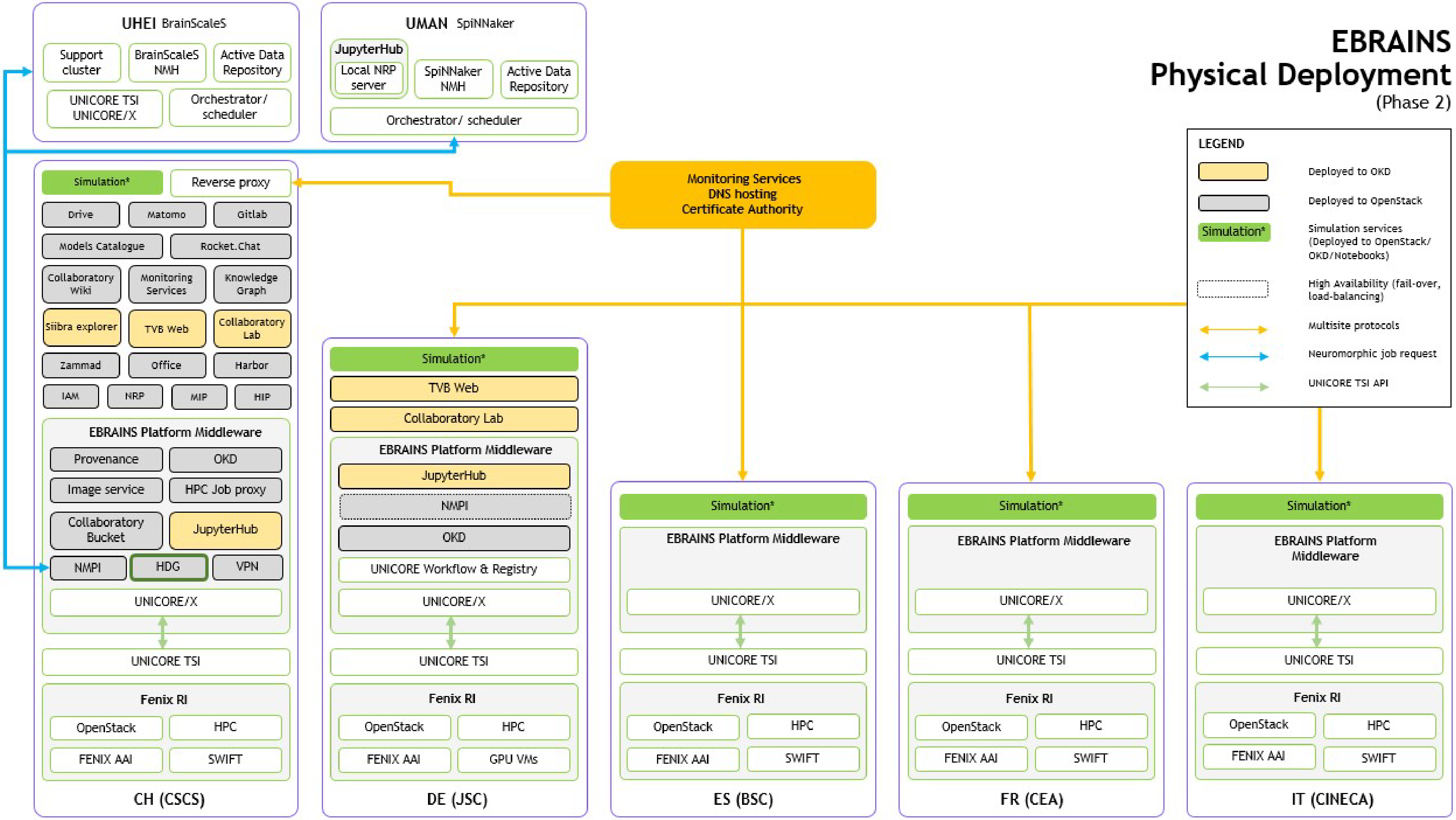


Figure 7: Physical Deployment Phase 2

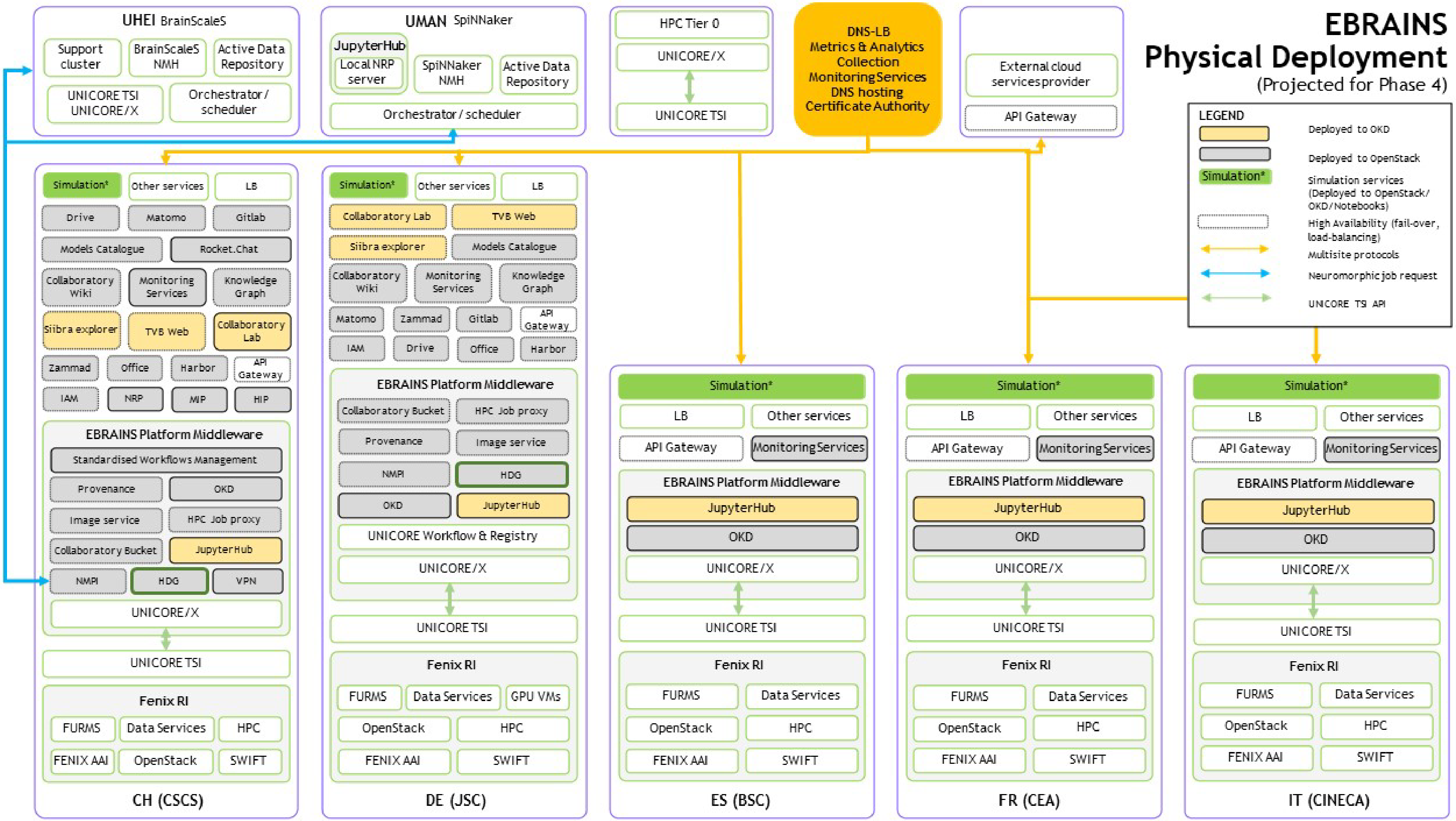


Figure 8: Physical Deployment Phase 4

2.7 EBRAINS Deployment Options for Component Developers

This section provides EBRAINS application/service developers with an overview of the deployment types 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. Upcoming iterations of the RI are expected to additionally include resources or possibilities which may not be present in this iteration.

The initial version of this section was presented in D5.3, Section 3.8, [1]. We include all the respective sections in the present document to outline the updates that occurred during Phase 2. The most significant update is the introduction of the Standardised Workflows (Section 3.2), as a new vertical column on the EBRAINS Deployment Resources diagram (Section 2.7.2). In order to formally detail the Standardised Workflows as a deployment resource, a new section and its respective diagram were added (Section 2.7.2.5).

Moreover, Collaboratory and IAM/AAI were merged to the EBRAINS Services box in all diagrams of the present Section. This change was applied to safeguard against potential confusion that Collaboratory is outside of EBRAINS Services. That particularity resulted in slight adjustments to all the diagrams regarding data and access control flows.

2.7.1 Overview of EBRAINS Deployment Resources

EBRAINS is composed of a small 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, being able to provide their functionality as an independent service in tandem with having it subsumed as part of the RI, and so on.

To assure minimum thresholds for the RI, there are still QA requirements of SoS components (see Section 5.1), 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 to EBRAINS RI

Except for edge-cases[5] , 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.
- **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.

2.7.2.1 Deployment type I – VM-hosted services

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. In later iterations of the RI though, this could be deduplicated to RI-provided centralised services for shared usage, at which point components' deliverables would just need to include the appropriate settings-fragments to plugin to those services. (Figure 10)

2.7.2.2 Deployment type II – Container-hosted services

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

[5] One important edge-case comprises the Neuromorphic Compute Systems (NMC), consisting (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

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 – Jupyter 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. (Figure 12)

2.7.2.4 Deployment type IV – HPC Library / Container runtimes

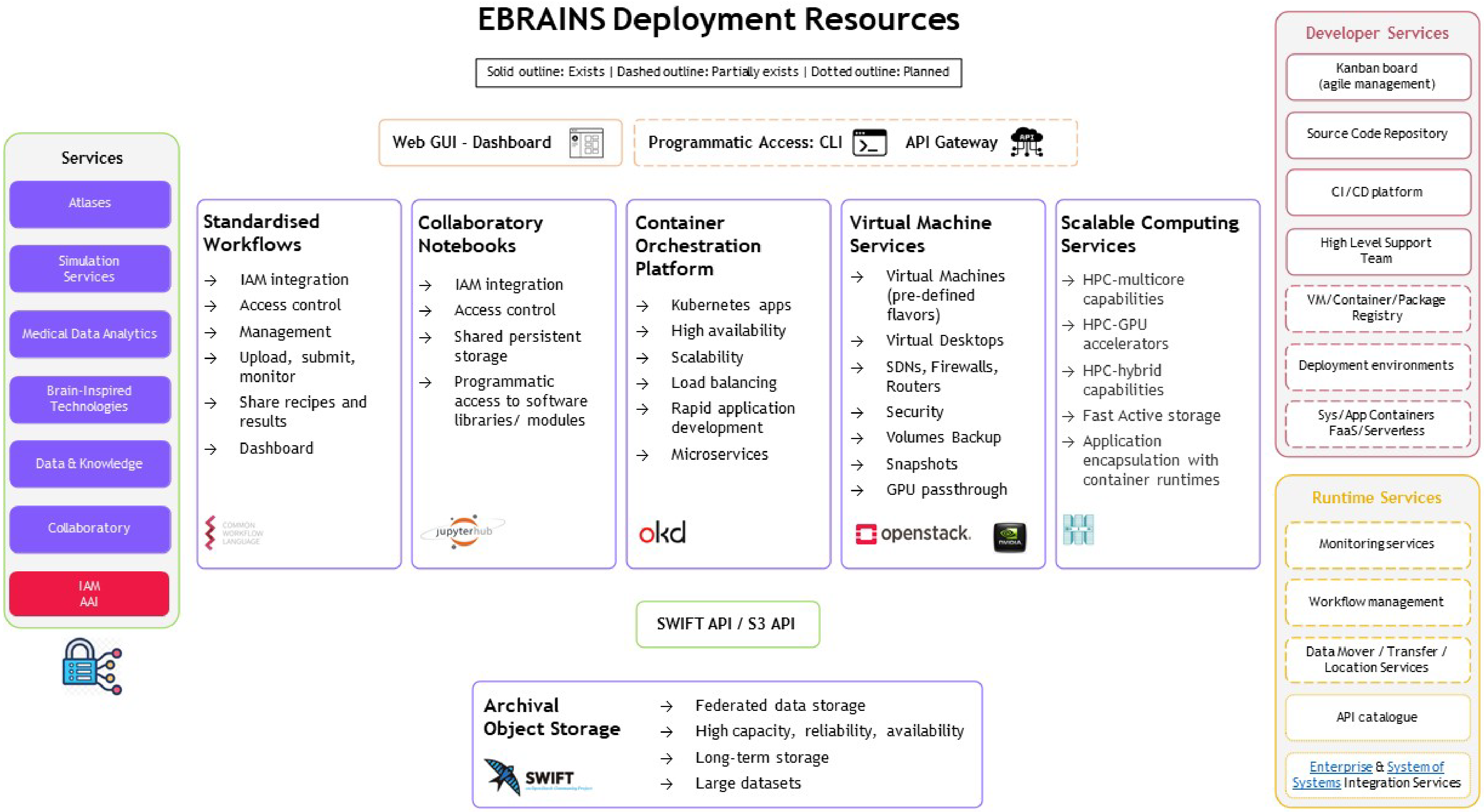
Executable code can be “deployed” (released, installed, and configured for runtime-activation by users rather than auto-activated) to the HPC resources, either as self-contained applications to be launched by service/user-workloads, or as libraries for providing functionality to other HPC applications. 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. (Figure 13)

2.7.2.5 Deployment type V - Standardised Workflows

Standardised workflows can be delivered for defining in a common and structured way computational workflows with steps associated with non-interactive command line tools with specific types of input parameters and specific type of provided output. The connections with other endpoints from platforms to data spaces related to different research scientific communities as well as possible industrial ones. Component developers can bundle non interactive command line tools in different ways (containerization methods, package managers, and more) in order to support dependency resolution, configuration management, software versioning and reusability of the workflow recipes as digital assets. 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 will be able to submit (deploy) standardised workflows to endpoints via a dedicated interface. (Figure 14)

2.7.2.6 Notes common to all deployment types

In addition to VMs, cloud containers, HPC containers and software-bundles being provided in their respective “Infrastructure-as-Code” (IaC) and “packaging-source” forms (so that the resulting outputs can be built, audited, and reproduced entirely from source if necessary), any constituent custom software also needs to be provided in source form (for scripting languages this is implicit, but for compiled languages there must be no “binary only” custom components). A VM image may include binary-only elements from well-known/trusted OS-level sources (e.g. a proprietary graphics driver for handling OpenGL over GPU-passthrough, interface firmware for high-speed networking, etc). The "no binary-only custom component" requirement does not apply to such things but applies to new/custom code. If a component does have a clear and justifiable reason for needing an exception to this (for example code that can otherwise be vouched for but needs to remain at least temporarily closed-source due to a pending/disputed patent, etc.) then these would need to be discussed with TC on a case-by-case basis and would most likely also need escalating to other members for final higher-level policy decisions. At present, where it is infeasible during early stages to provide a VM entirely as reproducible "provisioning sources on a base OS-spec", VM-images may be accepted in image-form only, on a case-by-case basis, but even those cases would be expected to provide highly transparent documentation of what built the image, from where, and how. The domain of pre-trained neural nets (and their origins/transparency/reproducibility) is a fluid subject, so although transparency is still encouraged, for this particular case it is necessarily more of a request for "best effort".



EBRAINS Deployment Resources

Solid outline: Exists | Dashed outline: Partially exists | Dotted outline: Planned

Web GUI - Dashboard

Programmatic Access: CLI

API Gateway

Services

- Atlases
- Simulation Services
- Medical Data Analytics
- Brain-Inspired Technologies
- Data & Knowledge
- Collaboratory
- IAM AAI

Standardised Workflows

- → IAM integration
- → Access control
- → Management
- → Upload, submit, monitor
- → Share recipes and results
- → Dashboard

Collaboratory Notebooks

- → IAM integration
- → Access control
- → Shared persistent storage
- → Programmatic access to software libraries/ modules

Container Orchestration Platform

- → Kubernetes apps
- → High availability
- → Scalability
- → Load balancing
- → Rapid application development
- → Microservices

Virtual Machine Services

- → Virtual Machines (pre-defined flavors)
- → Virtual Desktops
- → SDNs, Firewalls, Routers
- → Security
- → Volumes Backup
- → Snapshots
- → GPU passthrough

Scalable Computing Services

- → HPC-multicore capabilities
- → HPC-GPU accelerators
- → HPC-hybrid capabilities
- → Fast Active storage
- → Application encapsulation with container runtimes

SWIFT API / S3 API

Archival Object Storage

- → Federated data storage
- → High capacity, reliability, availability
- → Long-term storage
- → Large datasets

Developer Services

- Kanban board (agile management)
- Source Code Repository
- CI/CD platform
- High Level Support Team
- VM/Container/Package Registry
- Deployment environments
- Sys/App Containers FaaS/Serverless

Runtime Services

- Monitoring services
- Workflow management
- Data Mover / Transfer / Location Services
- API catalogue
- Enterprise & System of Systems Integration Services

Figure 9: EBRAINS Deployment Resources

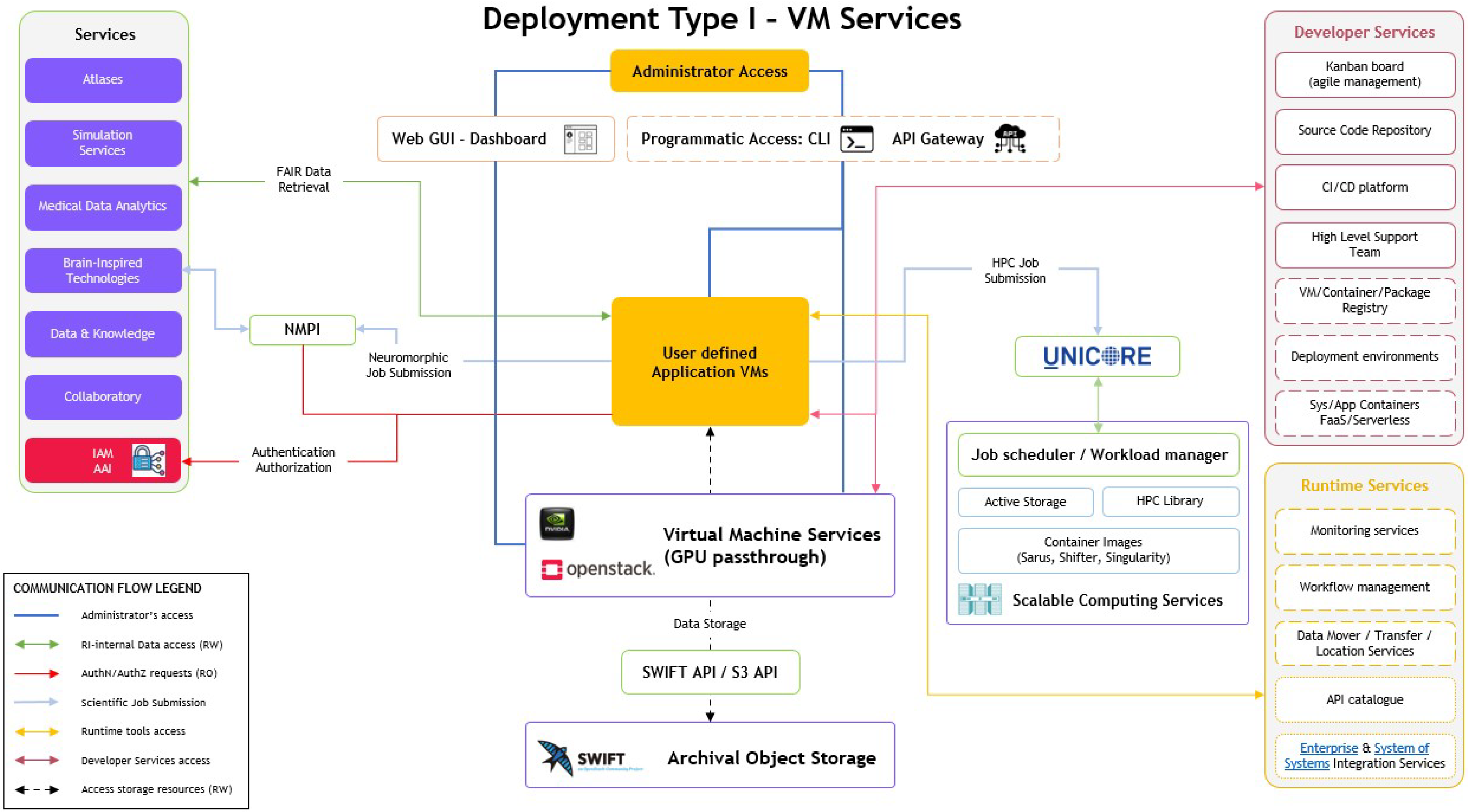

Deployment Type I - VM Services
Services
Atlases
Simulation Services
Medical Data Analytics
Brain-Inspired Technologies
Data & Knowledge
Collaboratory
IAM AAI
Administrator Access
Web GUI - Dashboard
Programmatic Access: CLI
API Gateway
FAIR Data Retrieval
NMPI
Neuromorphic Job Submission
User defined Application VMs
Authentication Authorization
HPC Job Submission
UNICORE
Job scheduler / Workload manager
Active Storage
HPC Library
Container Images (Sarus, Shifter, Singularity)
Scalable Computing Services
Virtual Machine Services (GPU passthrough)
openstack.
Data Storage
SWIFT API / S3 API
SWIFT
Archival Object Storage
Developer Services
Kanban board (agile management)
Source Code Repository
CI/CD platform
High Level Support Team
VM/Container/Package Registry
Deployment environments
Sys/App Containers FaaS/Serverless
Runtime Services
Monitoring services
Workflow management
Data Mover / Transfer / Location Services
API catalogue
Enterprise & System of Systems Integration Services
COMMUNICATION FLOW LEGEND
Administrator's access
RI-internal Data access (RW)
AuthN/AuthZ requests (RO)
Scientific Job Submission
Runtime tools access
Developer Services access
Access storage resources (RW)

Figure 10: Deployment Type I VM Services

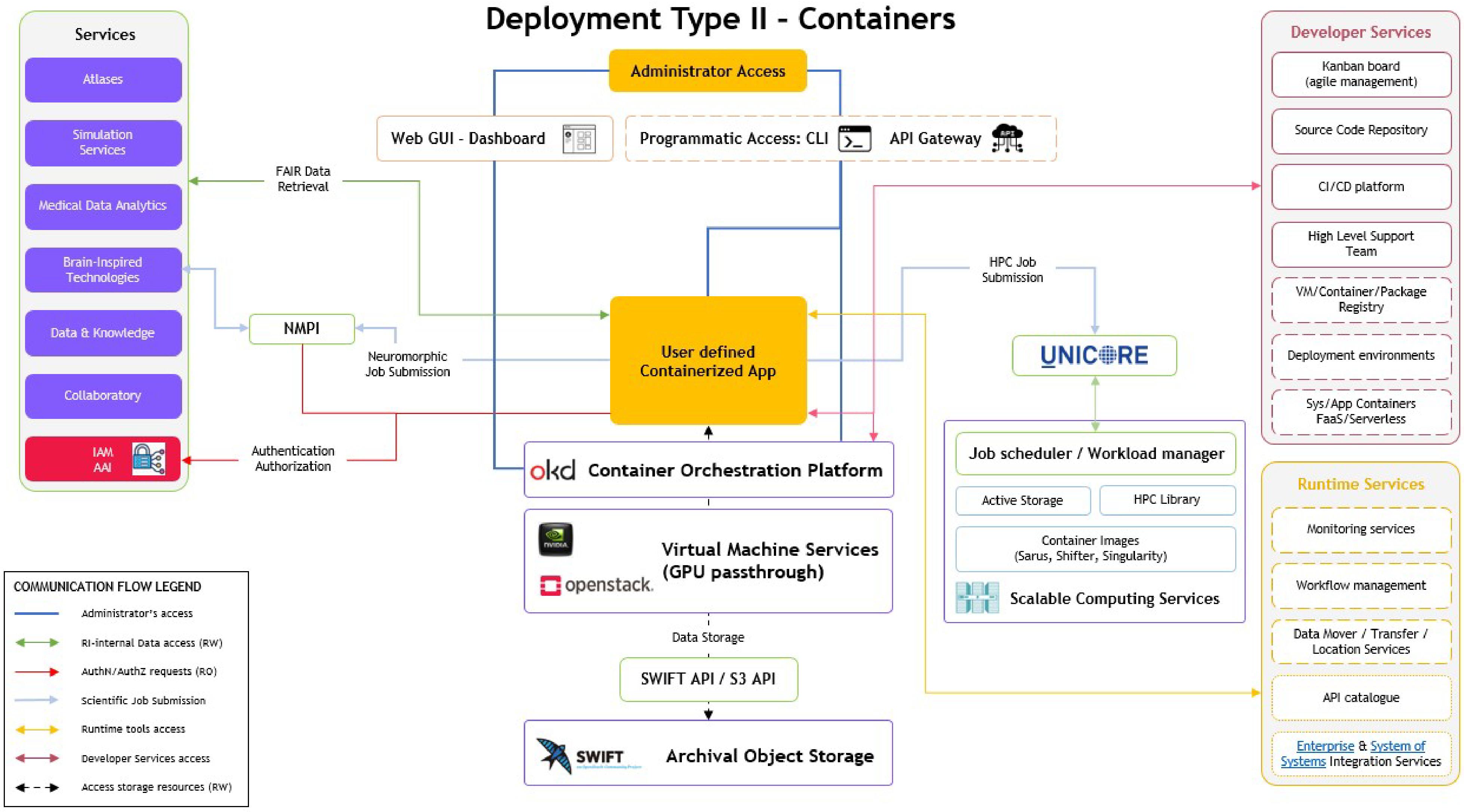

Deployment Type II - Containers
Services
Atlases
Simulation Services
Medical Data Analytics
Brain-Inspired Technologies
Data & Knowledge
Collaboratory
IAM AAI
Administrator Access
Web GUI - Dashboard
Programmatic Access: CLI
API Gateway
FAIR Data Retrieval
NMPI
Neuromorphic Job Submission
User defined Containerized App
HPC Job Submission
UNICORE
Authentication Authorization
okd Container Orchestration Platform
Virtual Machine Services (GPU passthrough)
openstack.
Data Storage
SWIFT API / S3 API
SWIFT
Archival Object Storage
Job scheduler / Workload manager
Active Storage
HPC Library
Container Images (Sarus, Shifter, Singularity)
Scalable Computing Services
Developer Services
Kanban board (agile management)
Source Code Repository
CI/CD platform
High Level Support Team
VM/Container/Package Registry
Deployment environments
Sys/App Containers FaaS/Serverless
Runtime Services
Monitoring services
Workflow management
Data Mover / Transfer / Location Services
API catalogue
Enterprise & System of Systems Integration Services
COMMUNICATION FLOW LEGEND
Administrator's access
RI-internal Data access (RW)
AuthN/AuthZ requests (RO)
Scientific Job Submission
Runtime tools access
Developer Services access
Access storage resources (RW)

Figure 11: Deployment Type II Containers

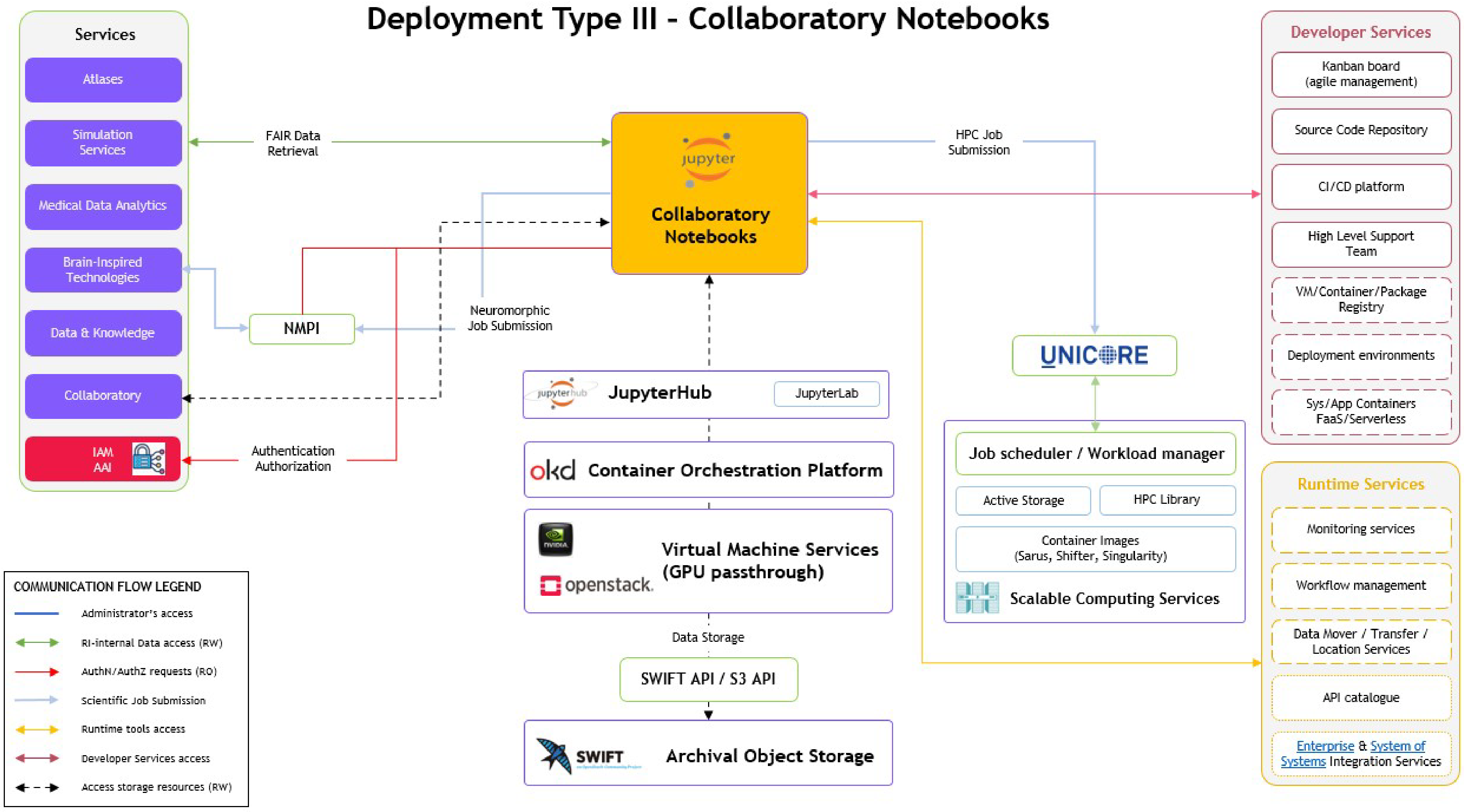

Deployment Type III - Collaboratory Notebooks
Services
Atlases
Simulation Services
Medical Data Analytics
Brain-Inspired Technologies
Data & Knowledge
Collaboratory
IAM AAI
FAIR Data Retrieval
NMPI
Neuromorphic Job Submission
Authentication Authorization
jupyter
Collaboratory Notebooks
HPC Job Submission
UNICORE
JupyterHub
JupyterLab
okd Container Orchestration Platform
openstack.
Virtual Machine Services (GPU passthrough)
Data Storage
SWIFT API / S3 API
SWIFT
Archival Object Storage
Job scheduler / Workload manager
Active Storage
HPC Library
Container Images (Sarus, Shifter, Singularity)
Scalable Computing Services
Developer Services
Kanban board (agile management)
Source Code Repository
CI/CD platform
High Level Support Team
VM/Container/Package Registry
Deployment environments
Sys/App Containers FaaS/Serverless
Runtime Services
Monitoring services
Workflow management
Data Mover / Transfer / Location Services
API catalogue
Enterprise & System of Systems Integration Services
COMMUNICATION FLOW LEGEND
Administrator's access
RI-internal Data access (RW)
AuthN/AuthZ requests (RO)
Scientific Job Submission
Runtime tools access
Developer Services access
Access storage resources (RW)

Figure 12: Deployment Type III Collaboratory Notebooks

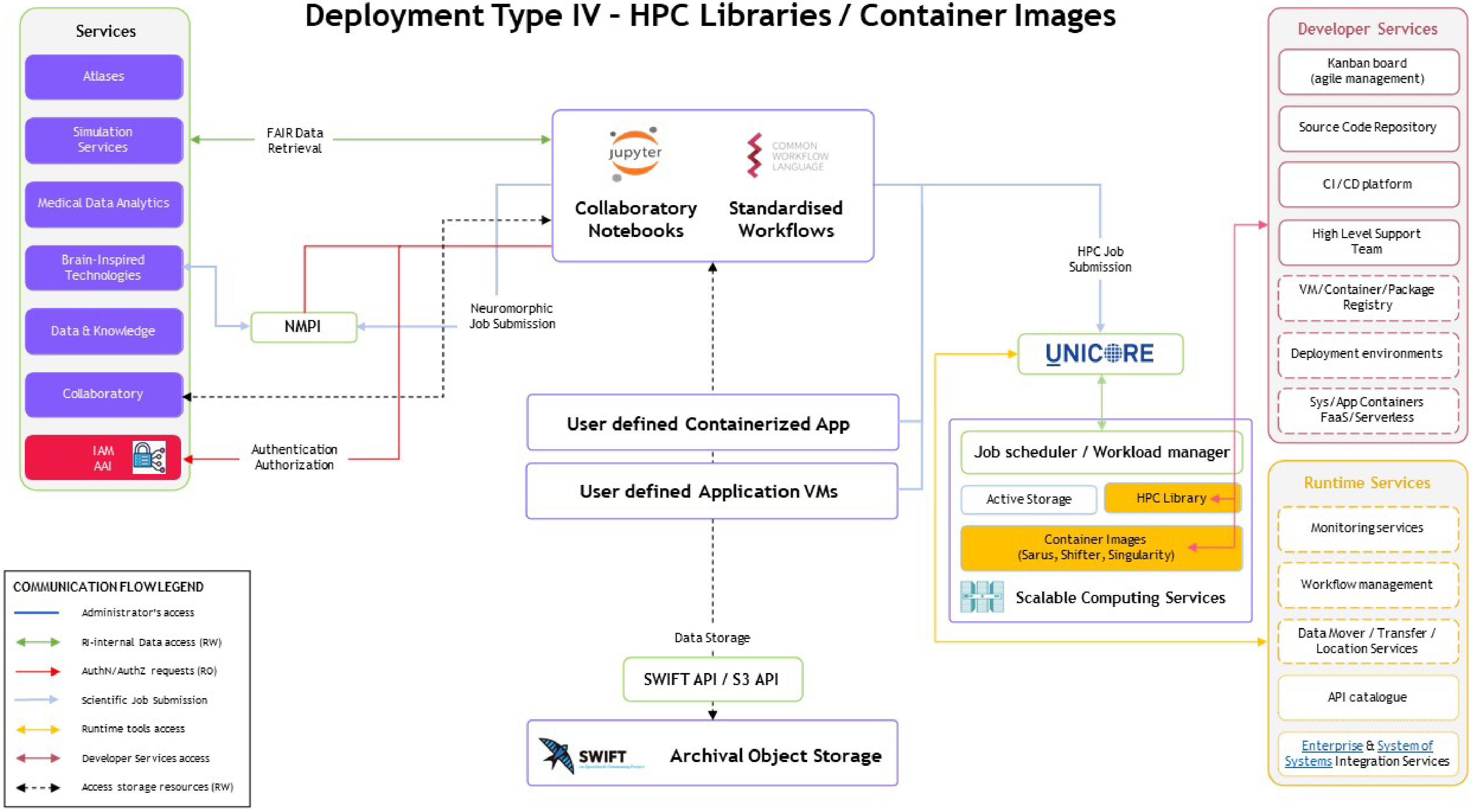

Deployment Type IV - HPC Libraries / Container Images
Services
Atlases
Simulation Services
Medical Data Analytics
Brain-Inspired Technologies
Data & Knowledge
Collaboratory
IAM
AAI
FAIR Data Retrieval
NMPI
Neuromorphic Job Submission
Authentication Authorization
jupyter
Collaboratory Notebooks
COMMON WORKFLOW LANGUAGE
Standardised Workflows
User defined Containerized App
User defined Application VMs
Data Storage
SWIFT API / S3 API
SWIFT
Archival Object Storage
HPC Job Submission
UNICORE
Job scheduler / Workload manager
Active Storage
HPC Library
Container Images (Sarus, Shifter, Singularity)
Scalable Computing Services
Developer Services
Kanban board (agile management)
Source Code Repository
CI/CD platform
High Level Support Team
VM/Container/Package Registry
Deployment environments
Sys/App Containers FaaS/Serverless
Runtime Services
Monitoring services
Workflow management
Data Mover / Transfer / Location Services
API catalogue
Enterprise & System of Systems Integration Services
COMMUNICATION FLOW LEGEND
Administrator's access
RI-internal Data access (RW)
AuthN/AuthZ requests (RO)
Scientific Job Submission
Runtime tools access
Developer Services access
Access storage resources (RW)

Figure 13: Deployment Type IV HPC Libraries/Container Images

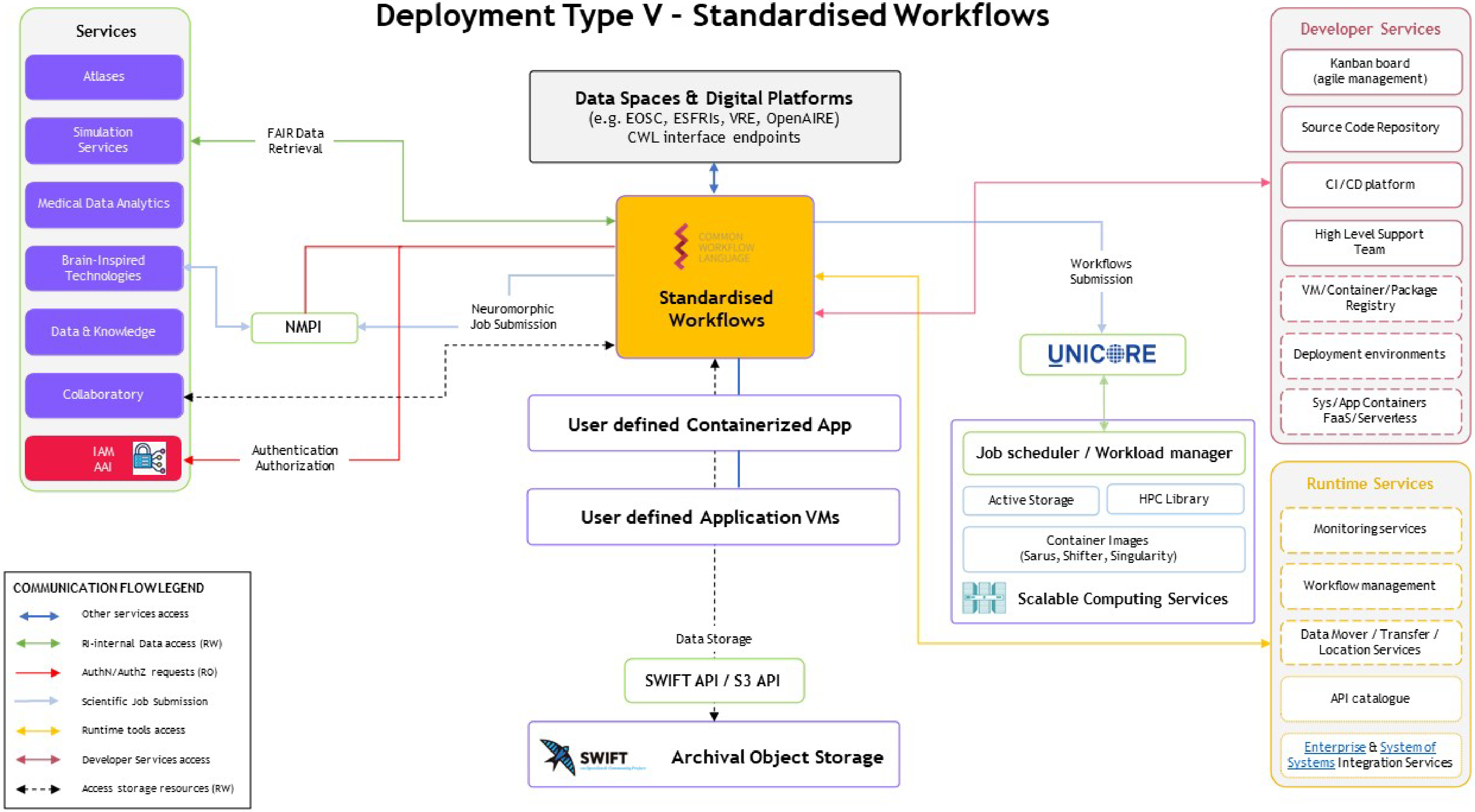

Deployment Type V - Standardised Workflows
Services
Atlases
Simulation Services
Medical Data Analytics
Brain-Inspired Technologies
Data & Knowledge
Collaboratory
IAM
AAI
FAIR Data Retrieval
NMPI
Neuromorphic Job Submission
Authentication Authorization
Data Spaces & Digital Platforms
(e.g. EOSC, ESFRIs, VRE, OpenAIRE)
CWL interface endpoints
COMMON WORKFLOW LANGUAGE
Standardised Workflows
User defined Containerized App
User defined Application VMs
Data Storage
SWIFT API / S3 API
SWIFT
Archival Object Storage
Workflows Submission
UNICORE
Job scheduler / Workload manager
Active Storage
HPC Library
Container Images
(Sarus, Shifter, Singularity)
Scalable Computing Services
Developer Services
Kanban board
(agile management)
Source Code Repository
CI/CD platform
High Level Support Team
VM/Container/Package Registry
Deployment environments
Sys/App Containers
FaaS/Serverless
Runtime Services
Monitoring services
Workflow management
Data Mover / Transfer / Location Services
API catalogue
Enterprise & System of Systems Integration Services
COMMUNICATION FLOW LEGEND
Other services access
RI-internal Data access (RW)
AuthN/AuthZ requests (RO)
Scientific Job Submission
Runtime tools access
Developer Services access
Access storage resources (RW)

Figure 14: Deployment Type V Standardised Workflows

3. Platform Middleware

This section presents the different tools of the platform middleware we developed. They are demonstrated to facilitate application/service developer's familiarisation with the set of offerings of the EBRAINS RI. Scientists, developers and component owners can benefit, not only from the **developed tools and services**, but also from the **standardisation** of scientific computational **workflows** that use EBRAINS data, tools, models, software and take advantage of the powerful underlying infrastructure that EBRAINS offers.

3.1 Software Delivery and Installation for the Lab

Towards seamless integration of the various components to the EBRAINS platform, there is a particular Use Case concerning the "Build, Delivery and Activation of software for direct use in Jupyter Notebooks in the Collaboratory Lab environment (from now, the Lab)". More specifically, how to efficiently and optimally build, deliver and activate the various simulation engines and other software that participate in the EBRAINS integration activities and make them available to Collaboratory users in the Lab, without requiring from them to perform any action to install the tools, while at the same time offer a streamlined process to the various Component Owners (COs)/software developers, through which they can make their software available to EBRAINS users in the Lab.

This section details the design and implementation of the delivery strategy that was defined and became operational during Phase 2 for the software delivery and installation for the Lab. The delivery strategy instantiates the general software delivery process for EBRAINS components that was initially drafted during Phase 1 (D5.3, Section 4.2.3, [1]) and further specified during Phase 2 (Section 4.1) and streamlines the integration activities of components to the EBRAINS platform for that purpose.

In the following sections, the main requirements for the delivery process in this domain are detailed, along with some background material for the core design choices of the solution and information on how the delivery strategy is implemented in the EBRAINS RI.

More details and documentation for the established process can be found in Annex 8.4. There are several actors that participate in the process, namely, the end-users, testers, developers, and DevOps/Administrators and so a dedicated section for each actor exists in the documentation to provide all the required information and facilitate the integration activities.

3.1.1 Requirements

The main requirements for all the involved actors (i.e. Collab Notebook users, Developers/COs, DevOps/Maintainers) in making available software for direct use in the Lab stem mostly from the difficulties the Lab has presented. These can be summarized in the following points:

- Collaboratory 1 and Collaboratory 2 Lab environment did not previously have a defined and scalable methodology for updating tools in the Docker image used to execute Jupyter notebooks
- Difficult to update/add software, because
  - it would break dependencies of all existing notebooks.
  - library dependency conflicts are difficult to solve by builders of Docker images.
  - tool installation and testing were often insufficiently documented to be performed by a central team.

Accordingly, the method to deploy tools was defined around the following points:

- smaller base container image without specific neuroscience tools
  - easier to maintain

    - decoupling the system (container image) from the application software (mounted onto the image)
    - separately testable from the neuroscience tools
- decentralised software development, testing and packaging in hands of the tool developers
- central toolbox
- modular software stack to activate specific tools and limit incompatible dependencies among tools
- as many of the tools as possible in the same activation environment to work out of the box
- load and unload tools with ease

If all the involved actors are now considered and the different domains are sliced, the main requirements for a new approach in Building, Delivering and Activating software for direct use in the Lab can be formed:

- requirements of end users of the Lab
    - Internal/external user of the Lab
    - Wants something that works out of the box (tested, no additional install/parametrization, seamless)
    - Wants to reproduce experiments published via Jupyter notebooks
    - Start a JupyterLab instance and start working with all necessary software tools already installed
    - Possibility to run today’s notebooks in the future by rolling back to today’s official release of tools
- requirements of tool developers
    - Same as the above end user’s
    - Have recent versions of compilers (i.e. C, C++) and Python available for building and running software respectively
    - Have the ability to install/load additional tools from the ones already available in the spawned container for testing/development purposes and/or for setting up more complex experiments
    - Streamlined process for making his/her component/tool available in the Lab
    - Software maintenance/update/testing in parallel
- EBRAINS platform DevOps/Maintainer’s requirements
    - Distribute software development, testing and packaging (COs responsible)
    - Centralize software deployment and distribution (Technical Coordination (TC) responsible, single integration point, central toolbox)
    - Easier maintenance of Collaboratory Lab base container image

3.1.2 Spack package manager

With the purpose of meeting the Requirements of such an integration and efficiently handling in a central toolbox different software with multiple versions, and conflicting dependencies while at the same time achieving distribution of the software development effort to the responsible COs, separation of concerns (for developers and DevOps/Maintainers) and easier maintenance, we strived for a solution that would be the heart of this integration, fulfil all the requirements and create a seamless integration “experience” both for the COs and for the Lab users.

After examining several alternatives (custom solution, NixOS, and others), we concluded that the Spack package manager ([5][6][7][8][9][10]) was the best fit for our use-case.

As mentioned in its documentation, Spack is a package manager that makes installing scientific software easy. With Spack, we can build a package with multiple versions, configurations, platforms, and compilers, and all of these builds can coexist on the same machine. This feature is ideal for our Use-Case where we envisaged a central toolbox holding the installation of all components that require to be available in the Lab.

Spack is not tied to a particular language; you can build a software stack in Python or R, link to libraries written in C, C++, or Fortran, easily swap compilers, and target specific micro-architectures. Spack can be used to install software without root privileges in your home directory, to manage shared installations and modules, or to build combinatorial versions of software for testing. Component owners can specify the package version, compiler, compile-time options, and cross-compile platform along with other options in order to specify their component's build in a shared environment. Spack installs every unique package/dependency configuration into its own prefix, so new installs will not break existing ones. Spack avoids library misconfiguration by using RPATH to link dependencies. When a user links a library or runs a program, it is tied to the dependencies it was built with, so there is no need to manipulate LD_LIBRARY_PATH at runtime.

Finally, creating portable Spack packages that enable the build and installation of a component by a Spack instance is an easy and straightforward process. Two key concepts are governing the packaging of software with Spack. These are the i) Spack package and ii) Spack spec concepts. A Spack package is a Python module that describes how to build software according to a Spack spec. A Spack spec is an expression for describing builds of software (descriptor). Specs allow a user to describe a particular build in a way that a package author can understand. Packages allow the packager to encapsulate the build logic for different versions, compilers, options, platforms, and dependency combinations in one place. Essentially, a package translates a spec into build logic. A Spack package and spec are needed to be provided by COs to allow for the central "build, delivery and activation of their component for the Lab". For more information on Spack you need to refer to the respective documentation.

The next section addresses how Spack was leveraged to fulfil the Requirements along with providing the necessary perspectives for this workflow to the respective audiences to understand how to utilize the new functionality and what it entails for them.

3.1.3 Perspectives

In this section, we present the details of the implemented solution. The High-level perspective section provides a high-level overview of the solution without much technical details. As mentioned, technical details can be found in Annex 8.4.

3.1.3.1 High-level

The main aim from an operational side was to distribute software development (responsibility of COs) and centralize the build, deployment, and distribution (responsibility of TC, with a single integration point) of the software for the Lab capitalizing on an externally maintained, open source, robust and widely accepted solution (i.e. Spack).

Based on the EBRAINS CI/CD workflow that was established in (D5.3, Section 4.2.3, [1]) and can be visualized from the respective flowchart[6] the integration steps for instantiating the release process for software for the Lab were identified, designed, and implemented.

Architecture wise, the involved entities on EBRAINS are:

- Gitlab platform and Gitlab CI [build environment]

[6] https://drive.ebrains.eu/f/b9f56875697e46dbaf0b/ (p.14)

- Docker registry [build environment]
- OKD Container Platform [build environment]
- JupyterLab (Lab) [runtime environment]

The implementation of the delivery strategy for software for the Lab adheres to the software delivery guidelines of EBRAINS RI for this particular use case. It capitalizes on the mature integration status of the EBRAINS components abiding by the Integration Requirements (Annex 8.2) and manifests through a streamlined CI/CD pipeline that has its steps described below:

- [Component's official code repository] Development effort
- [Gitlab] Init step – Get EBRAINS released code through the component's mirror
- [Gitlab] Component step – child pipeline, preparation step:

  for all components copy the respective Spack directories/subdirectories that contain the necessary Spack packages and specs to a shared location
- [Gitlab] Integration step
  - Prepare the OKD job that will perform the build
  - Initiate the OKD build job
  - Transfer all necessary Spack folders/files from the shared location and all Spack environment configuration files, and scripts from a TC repository to the OKD pod that performs the build
  - Wait for completion of the OKD build job that builds and installs all software in the same Spack environment (on an NFS partition that is mounted on users' Lab containers)
  - Propagate the logs from the OKD build job back to Gitlab
- [Gitlab] [Integration Lab] Review step – pause the pipeline for the Reviewers to check that everything is installed as expected (read access to the Integration Lab)
- [Gitlab] Delivery step
  - Prepare the OKD job that will perform the build
  - Initiate the OKD build job
  - Transfer all necessary Spack folders/files from the shared location and all Spack environment configuration files, and scripts from a TC repo to the OKD pod that performs the build
  - Wait for completion of the OKD build job that builds and installs all software in the same Spack environment (on an NFS partition that is mounted on users' Lab containers)
  - Propagate the logs from the OKD build job back to Gitlab
- [Manual step] Deployment - Make it available to users – update the Collaboratory container image to point to the newly built environment (JupyterLab kernel)

Implementation can be abstracted in simplified way from Figure 15.To sum up, the main points from the High-Level perspective of the implementation of the integration process for the release of software for the Lab are the following:

- Describe software stack, dependencies and build instructions with Spack
  - Each component (responsibility of COs) must provide a Spack package
- Build and install components on a shared NFS partition shared by the build environment and the runtime environment
  - The process is coordinated and performed centrally by TC in Gitlab CI
- The output is all software deployed in this way is available to users of the Lab who can then choose to activate specific software or use the default latest release activated by default

- The process encourages the use of a common modern version of core compilers (e.g. GCC) and Python interpreter
- Dependencies for each executable are tracked by Spack in a dependencies graph
- Reproducibility, a release of tools in the Lab features:
  - a well-defined Spack environment (list of software packages with fully traceable dependencies)
  - a corresponding JupyterLab kernel
  - a version tagged Build environment (TC container image that builds and installs software on the shared NFS partition)
  - a version tagged Runtime environment (Lab container image)
  - All the aforementioned metadata relating to a release can be stored in the Knowledge Graph.

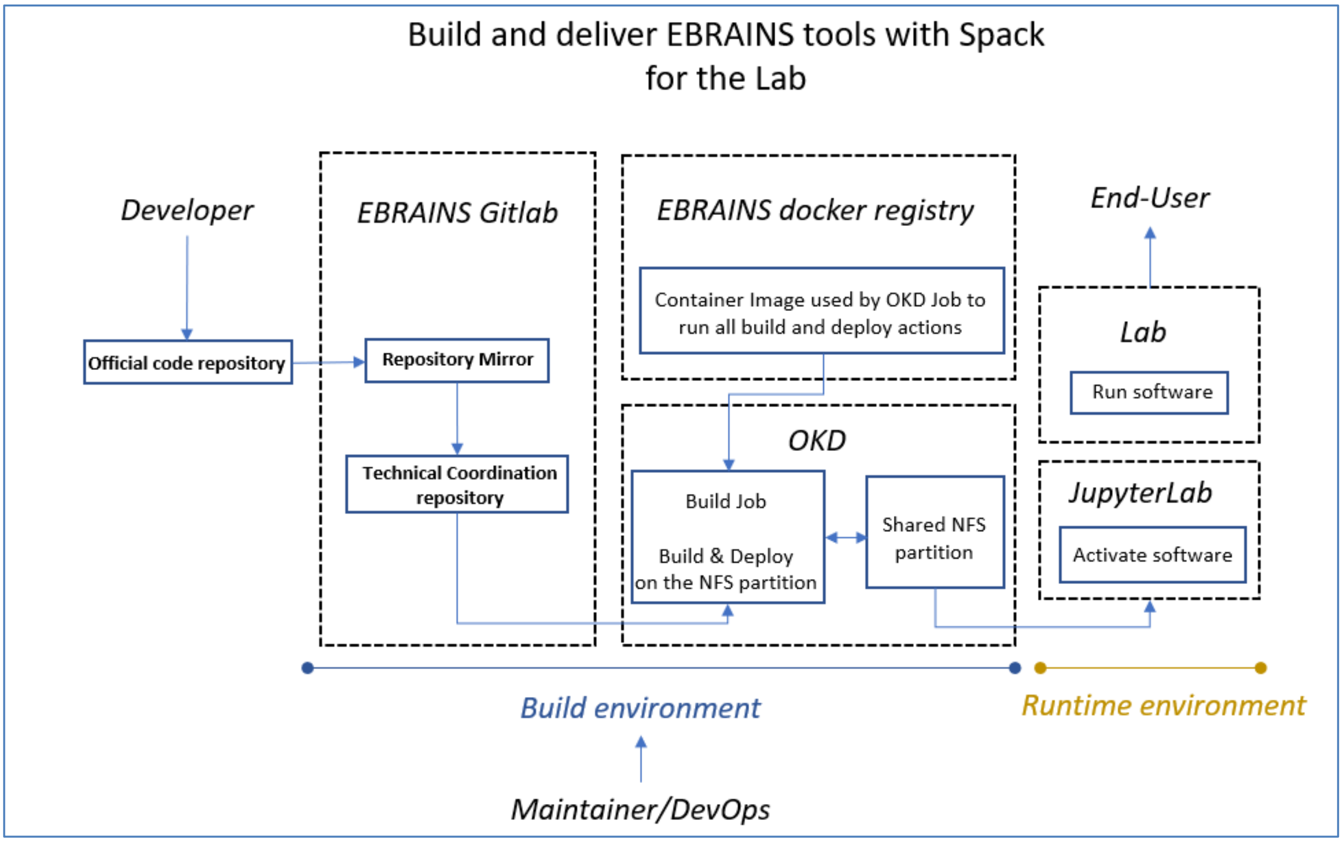


Figure 15: Implementation diagram

3.2 Workflows

Our work was dedicated to the standardised aspects of scientific computational workflows using EBRAINS data, tools, models, software and the underlying infrastructure that EBRAINS offers. As a first step, the Technical Coordination team focused on defining different generic terms related to workflows. In general, we define:

- Scientific Computational Workflows: descriptions of series of tasks that can be linked together to create Directed Acyclic Graphs (DAGs), loops or branches, for accomplishing a scientific objective, usually expressed in terms of tasks and their dependencies. Typically, workflow steps are interactive or non-interactive computational tools for scientific data manipulation or even whole sub workflows.
- Interactive tools: pieces of software, with well-defined inputs and outputs, that allow user interaction during the runtime. In that way, parameterisation of the results can happen on-the-fly.

- Non-interactive tools: pieces of software, with well-defined inputs and outputs, that allow user parameterisation of the inputs only before the runtime.

For EBRAINS RI, we define:

- Standardised workflows: series of non-interactive EBRAINS tools and services, linked together in order 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.

Technical Coordination team focused on the standardisation of scientific computational workflows, in order to alleviate the challenges and limitations that EBRAINS users were facing when defining their scientific workflows, in terms of automated execution and monitoring, as well as portability, scalability and FAIRness. Different Research Infrastructures and projects associated with Life Sciences, such as ELIXIR [11], EOSC-Life [12], Global Alliance for Genomics and Health (GA4GH) [13] and others, all abide by fundamental standardised workflow concepts. These concepts include bundling different tools along with their dependencies on the one hand, and the common and structured definition of workflows that use their developed scientific tools as steps on the other. These steps are executed and monitored via workflow engines. The workflows themselves are stored as digital assets in separate public hubs and registries (such as WorkflowHub [14], myExperiment [15]) for shareability, accessibility and findability purposes. However, in EBRAINS, the means and technologies that component owners/developers and scientists had available hindered the shareability and findability of their scientific work. This applied not only to external scientists and research communities but also internally between different EBRAINS teams. Jupyter notebooks were used as the main paradigm for accessing different, non-homogenous HPC systems in which non-interactive tools and series of tasks were submitted. Often, scientists were accessing and using different HPC systems by using Secure SHells (ssh) in order to launch jobs from custom templates and recipes for executing tools, directly in the workload manager of each HPC system. This approach, although based on FAIR data, models and tools, was not FAIR in itself, since it was only available to communities via notebooks. Notebooks have limited capabilities because they are interactive environments used for executing linear chunks of code, writing documented steps and retrieving output results.

This resulted in associating only the produced output (data and models) with publications, papers, and metadata in the Knowledge Graph (the central place for metadata management in EBRAINS) after going through the Curation Process, which is mandatory for public datasets hosted in EBRAINS. The lack of associating the code as well with publications and papers made reproducibility, accessibility and interoperability of scientific work related to brain research quite hard to establish.

Considering common methodologies applied for workflows in different scientific fields (e.g. Life Sciences) as well as current limitations in the EBRAINS RI, Technical Coordination focused relevant work on four central pillars. Firstly, we proposed means for scientists to define standardised workflows as recipes associated with unique identifiers in a structured, common, and well-defined format. Secondly, we focused on proposing ways for scientists and component owners to execute and monitor the already defined workflows via pre-installed workflow management systems. Thirdly, we focused on storing the outputs and workflow recipes inside Knowledge Graph for shareability and accessibility purposes. Finally, we provided component owners and developers with ways to bundle non-interactive command line tools together with their dependencies and libraries so that they can be easily (re-)used as workflow steps in different workflow definitions.

The benefits of the proposed work for the standardisation aspects (define, bundle, execute and store) pertain not only to EBRAINS as a Research Infrastructure, but also to scientific communities that are related to EBRAINS, as well as to external scientific communities. For scientific communities, having ways to describe scientific computational workflows as structured, well-documented recipes facilitates automation and flexibility. For the execution of the workflow recipes, automation and portability are also introduced. Automation is achieved by different workflow engines taking care of the execution of the workflows, the failures and restarts of intermediate workflow steps as well as fetching logs and outputs after the execution. Also portability is ensured through transparent software dependency management of the underlying infrastructures in which workflows are executed. Having ways of defining workflows in a standard format gives an exceptional value to the scientific work itself, and not only to the scientific outcome, since work will be easily reproducible and accessible by different scientists and communities. For EBRAINS as a Research Infrastructure, benefits stem from the fact that scientific workflows created in the scope of the Research Infrastructure can themselves become Digital Objects, thus, can be referred to in publications, papers, journals and citations. In that way the horizon can be expanded not only to EBRAINS users but also to external ones, for the sake of FAIRness and Openness. To lower the entry barrier for new users, EBRAINS provides a concrete way to define workflows in a broadly accessible format which different research communities are already familiar with. In that way, the audience could be expanded to more scientific fields ranging from Life Sciences to Earth Sciences, not focused only on fields related to neuroscience and brain research. Scientists wishing to execute their standardised workflows in different environments, with different capabilities, requirements and specifications, and without the need to reconfigure or readjust their work, will be able to do so on the EBRAINS RI.

As mentioned, Technical Coordination team focused on four critical pillars (define, bundle, execute, store) relating to standardisation. More precisely, for scientists to define in a common and standard way different workflows and tools, we identified that Common Workflow Language (CWL) could best fit our purposes. Common Workflow Language is an emerging open standard. It is designed for defining analysis tools and workflows with specific inputs, parameters and resources, to be executed portably and scalably across many software (workflow engines) and hardware (local, cloud, HPC) environments, resulting in consistent outputs. CWL is compatible with a plethora of workflow engines that can/may be installed in various environments. By using CWL as a standard format in defining workflows as structured recipes, component owners, developers and scientists do not have to parameterise them for execution in different infrastructures. This is important since CWL decouples the description of a scientific computational workflow from its execution in different environments, with different requirements, and with different configurations. For this purpose, we have already defined and made available to all EBRAINS users a small reference set of workflows in CWL. This set of workflows can be retrieved from a dedicated EBRAINS Gitlab project.

In order to use CWL to define standardised workflows, we must first have CWL descriptions of EBRAINS non-interactive command line tools that will/can be used as workflow steps. These CWL descriptions have to provide concrete information about the input parameters, the expected outputs as well as the resources needed for the command line tool to run. For that reason the Technical Coordination team proposed to component developers to use Common Workflow Language (CWL) to also define EBRAINS non-interactive command line tools for data manipulation and data analysis. These well-defined tools can be used as steps in various workflow definitions; thus, it is important to be easily reusable and be used as-is. Another critical requirement is bundling command line tools with their dependencies, libraries and binaries so that they can easily be referred to as workflow steps in various workflows. For our purposes it was decided to use containerisation to bundle tools with their dependencies, after several alternative solutions were explored. This will help with describing well defined tasks and their dependencies as steps in scientific computational workflows in a flexible, simple, robust way within the EBRAINS RI. This gives a degree of freedom to scientists and component owners who can use different predefined and bundled command line tools in their definition of scientific computational workflows. We have already containerised a small set of tools with accompanying CWL notations for use as workflow steps. To showcase the benefits of reusability and replaceability of different bundled command line tools we introduced two tools with different ways of fetching data in the context of EBRAINS RI. The first tool is related to Data Proxy and the second to Knowledge Graph for getting the location of stored data. Both tools were described with specific types of input and specific types of output in such a way that they can be easily (re)-used in defining different workflows.

Moreover, use of standardised CWL enables various implementations to equivalently execute a single scientific computational workflow definition. A workflow management system is needed to execute workflow steps, monitor their execution, handle possible failures when they occur, and automatically fetch logs and outputs. Since different workflow engines exist, compatibility with CWL was a very important and critical element that was taken into consideration when selecting workflow engines that fit our purposes the best. An extensive list of CWL-compatible workflow management systems is already available in EBRAINS – deployed in OpenStack Virtual Machines, in OKD pods, and in HPC systems. For our test cases different scientific computational workflows defined in CWL (with packaged command line tools as workflow steps) were executed in the different workflow engines. Currently, we are in the process of finalizing different solutions which we identified as important for users to be able to execute their workflows. We are also currently discussing the finalization of long-term solutions like making UNICORE compatible with CWL so that it can be the primary workflow engine for workflow execution, and the installation of a Task Execution Service (TES) [16] server on the OKD cluster where users will be able to submit CWL workflows using clients such as CWL-TES [17]. The implementation used for the TES server is a modified version of Elixir’s TESK [18].

The last aspect of standardisation focuses on making scientific work, thus workflow recipes, easily accessible and findable in a straightforward way by different EBRAINS users and scientific communities for reproducibility reasons. Therefore, a central registry for CWL workflows with unique identifiers is mandatory for future reference. Bundled command line tools defined via CWL and available for use as workflow steps in workflow definitions by different EBRAINS users should also be easily findable and accessible. After some first discussions with the appropriate teams, we identified that Knowledge Graph could be used as a registry for finding, accessing, and storing scientific computational workflows as workflow recipes, and tools to be used as workflow steps, in the context of EBRAINS RI. This is appropriate because it is already used as a multi modal metadata store that combines information from different fields on brain research, data and models, as well as software existing in EBRAINS. Provenance information can also be automatically stored inside the Knowledge Graph using the Provenance API (Section 3.6.1). In that case, all related retrospective information (during the execution), prospective information (workflow recipes) such as descriptions, different inputs and provided outputs, as well as scientific outcome (data and models) will be linked together for reproducibility reasons under the EBRAINS Knowledge Graph.

3.3 Service for automated headless browser testing of Jupyter notebooks

Collaboratory Lab end-users develop Jupyter Notebooks to perform a wide variety of actions such as execute experiments, reproduce scientific/published results, and run workflows. These actions can also be linked to other core EBRAINS components (e.g. Knowledge Graph) or APIs which are necessary for their execution. Consequently, testing and determining the probability of successful execution of important public facing Jupyter notebooks forms an important part of the Quality Assurance (QA) in EBRAINS. Tests should be performed to ensure that all necessary resources are available for a successful execution while the visualization of test results can provide useful insights to Notebook owners. Furthermore, Notebook owners need to run the notebooks they have developed in advance to check that the desired results can indeed be produced before publishing it or sharing it with other users.

To satisfy the above requirements, a dedicated service for executing automated headless browser tests of Jupyter Notebooks has been developed during Phase 2 and a first version has been already released. The service is available and in production operation to all EBRAINS end-users. This particular endpoint returns all the notebooks registered to the service eligible for testing.

The service is based on the robot framework, which is an open-source automation framework, and can be used for testing automation solutions [22]. The execution of the testing service is facilitated by the EBRAINS DevOps platform (Gitlab) through a GitLab CI job, configured and maintained by TC.

Notebook Owners can register one or more notebooks as eligible for testing to a dedicated catalogue via a REST API. The service reads this catalogue and executes periodically (once a week) a dedicated testing pipeline. In case of error(s), the result of the test is FAIL and the user is notified via email. In order to provide informative insights of the reason that the test failed, the message body of the email includes a link to the artifacts generated by robot framework.

3.4 Automation and Back-end Tools and Services

In this section we present a set of tools developed by the Technical Coordination team during Phase 2 to facilitate automation in provisioning, configuring, and deploying infrastructure and services for core platform functionality. The tools, even though they are currently on early stages of development, have already been used in several prototypes and to work-around certain technical difficulties. Once a more mature state has been reached, the repositories will be opened for sharing and can be made available to the Consortium to facilitate DevOps engineers of the various component teams. The developed tools are introduced in the following paragraphs:

- An Automation tool for setting up Openstack infrastructure, provisioning the VMs, deploying container cluster(s) onto the VMs, deploying application-management framework, and deploying applications with that. Version 1 required manually scripted calls to the Openstack API (rather than stacks functionality like that provided by Openstack Heat) due to technical limitations which were since resolved by the sites, and the need to handle multiple different site configurations (with the most restricted as the common-denominator) without exploding code-size.
- A reusable deployment library based on Openstack Heat, cloud-init, and Ansible (with stubs in place for Juju later). Enables short (~10-15 line) shellscripts to configure entire geo-Highly Available (HA) deployments in an idempotent, rollback-capable manner. Used already for DNS Authoritative servers (primary and secondary in different Fenix RI sites), for SSH jumphosts at every site, for a special-case GPU-passthrough VM for one component, for various PoCs including LXD federated cluster (hosting system-containers and KVM VMs on a multi-node cluster, with real-time migrations, snapshotting, etc). Will later be integrated with v1 of the tool listed above to create a more user & developer-friendly "v2.0" UI tool, in addition to retaining the underlying library for developers to reuse.
- A PoC setup of Kong API Gateway for work relating to the EBRAINS Central API architecture component and the activities of the EBRAINS API Working Group. During the PoC setup, mechanisms were implemented for:
    - live-configuration by REST calls from operators
    - snapshotting the runtime configuration to static configuration
    - sharing the configuration in a decentralised way via gossip protocol
    - activating self-healing mode in which the gateway will monitor and revert to a known-state in the face of unexpected external changes

3.5 Monitoring services

A critical need for the EBRAINS RI is to monitor all of its software components in order to detect early any problems that may arise, observe resource utilization, identify usage patterns, pinpoint workload spikes, and even detect potentially malicious behaviour. Monitoring involves the collection of the underlying infrastructure’s metrics information, consumption, and analysis of application logs, as well as service availability and Web Analytics information. To achieve this multi-level global monitoring view, monitoring must be done at different levels. More specifically, monitoring needs to be performed at the service level, at the Web front-end level, and at the back-end level. These levels are further detailed in Annex 8.8.

In Figure 16, the Proof-of-Concept design implementation of such a centralised EBRAINS monitoring system is presented. Three main sources of metrics/information have been identified, to be collected and aggregated in a central data store (i.e. an Elasticsearch DB). Metrics are collected from the different EBRAINS web UIs using the Matomo Web Analytics Platform. The type of metrics currently collected and the way they are visualised in Matomo are described in Annex 8.8.2. Back-end monitoring consists of data shippers (called “agents”) which collect and send performance metrics for the underlying infrastructure’s Operating System (OS) and Application level (APM-Application Performance Monitoring). The agents are installed and configured in the component/application/service back-end layer. More details about these two layers are provided in Annex 8.8.3. Finally, interconnections among the EBRAINS back-end monitoring stack (Elastic stack) and other infrastructure monitoring applications will be established to accumulate as many sources as possible (e.g., Fenix RI/FURMS, EBRAINS platform Container Orchestration Engine (OKD) Neuromorphic Hardware (NMH) monitoring dashboard [30].All collected information can be aggregated, queried and visualised in the Platform Administration Dashboards. At the Component Administration Dashboards, only the collected metrics from OS and APM will be shown - dedicated to each component separately. As such, each Component Owner will be able to view key information regarding their component.

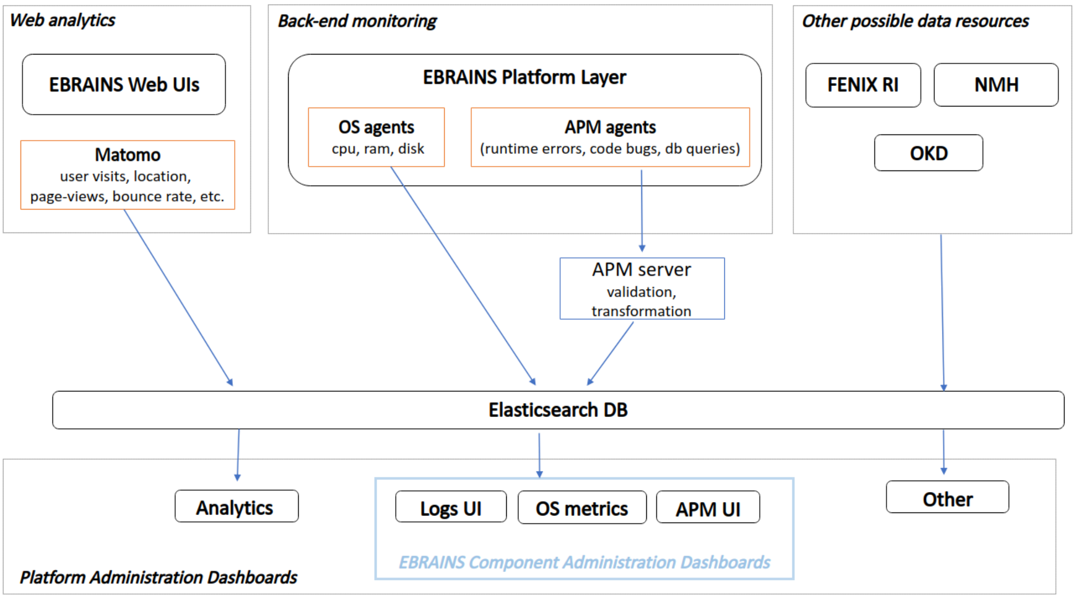

Web analytics
EBRAINS Web UIs
Matomo
user visits, location, page-views, bounce rate, etc.
Back-end monitoring
EBRAINS Platform Layer
OS agents
cpu, ram, disk
APM agents
(runtime errors, code bugs, db queries)
Other possible data resources
FENIX RI
NMH
OKD
APM server
validation, transformation
Elasticsearch DB
Analytics
Logs UI
OS metrics
APM UI
Other
EBRAINS Component Administration Dashboards
Platform Administration Dashboards

Figure 16: Overview of a Proof-of-Concept implementation for monitoring services

3.6 EBRAINS APIs

Led by the API-WG, effort has been made towards formalising and tracking the status of the API ecosystem in EBRAINS. Progress can be reported along the following lines:

- An API Catalogue[7] has been created and all presently known web APIs have been included.
- An internal checklist of requirements for the documentation-endpoint of each API in the catalogue has been created, and a collective effort is underway to audit and coordinate with component-developers for ensuring clarity, coherence, and maintainability of the documentation based on that checklist.
- There is ongoing progress in creating a best-practices and guidelines document. This includes links to general information and includes inline information where appropriate (especially for EBRAINS-specific guidelines or those requiring contextual descriptions). This will become available at a later stage.
- Assistance has been provided for some components wishing to transition from manual API development and documentation to OpenAPI/Swagger.

TC performed a proof-of-concept (PoC) manual deployment of a highly-available, self-healing API Gateway and web Reverse-proxy (Kong plus its underlying Nginx). Effort was also allocated towards combining this PoC with a "universal" service-mesh at a later stage (the most promising for various reasons - including shared-origin and compatibility with Kong - is Kuma). The intention is to eventually move to a central, highly-available, multi-site-deployed API Gateway (as a "core element", outside of the container orchestration clusters) for easy unification and exposure of web/API ingress across all container-cluster-hosted, VM-hosted, and externally hosted API services. This would not only make vulnerability-patching (external auth/ssl/etc) a one-step process, but also would facilitate easy/fast circuit-breaking, and other network traffic management actions.

[7] https://wiki.ebrains.eu/bin/view/Collabs/api-catalogue/

Moreover, for all East-West traffic and for deployment-strategies/rollback/observability/maintenance of all component services, a universal, multi-zone service mesh would be deployed. The envisaged service mesh would also be VM-hosted, external to the container orchestration clusters, so it can "universally" handle all traffic - whether from/to VMs, external services, services deployed to CaaS, etc. This API Gateway and universal service mesh combination would be a feasible and effective solution to the increasing complexity of the multi-location, hybrid monolithic + microservice, constant-onboarding model of EBRAINS.

3.6.1 Provenance API

A unified API for provenance is under development. An OpenAPI specification has been designed[8] [9] and feedback has been solicited, along with ensuing revisions. The specification provides endpoints for simulations, data analyses, visualisations, optimisations, and workflows. The metadata schema which will be used as payload for the API (for storing provenance in the KG and interacting with it) was designed, incorporated into OpenMINDS[10], and deployed to the KG. Several indicative workflows are stored and registered in the KG using the defined schema. Efforts for standardising scientific workflows for use across EBRAINS (Section 3.2) and for supplying a formalised environment-module system as base environment for lab notebooks (Section 3.1) are instrumental in narrowing the scope for the next phase of implementing the API itself, as well as developing a UI to simplify access.

[8] https://prov.brainsimulation.eu/docs

[9] https://prov.brainsimulation.eu/redoc

[10] https://github.com/HumanBrainProject/openMINDS

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

4.1 Software Delivery - main developments and changes

As documented in the Software Delivery section of the Technical Coordination Guidelines (D5.3, Section 4.2.3, [1]), EBRAINS RI is a complex e-infrastructure relying upon federated, multi-site, base computing resources provided by the Fenix RI. This poses various challenges regarding the delivery process and the successful utilisation/operation of several service environments. Case-dependent deployments (both single and multi-site) only add to the challenges that need to be addressed. The software delivery pilots that were conducted during Phase 1 facilitated the initial draft of the Software Delivery guidelines. Phase 1 concluded that individual (component specific) deployments are efficient (fast, bottom-up) but difficult to manage at scale. To sustainably operate EBRAINS, we need to evolve towards a centrally coordinated and horizontal deployment process.

During Phase 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. This section provides an update on the main developments and changes that occurred during Phase 2 and further clarifies important concepts that occur in other parts of this document.

The main strived-for axes of Software Delivery in EBRAINS are detailed in the following sections. In Section 4.1.1 we clarify and expand on the top-level automated (code) monorepos that will be the most fundamental integration points for all components. In Section 4.1.2 we clarify the notion of namespaced integration directories which are a vital part of integrating and gatekeeping the automated monorepos. In Section 4.1.3 we describe facilitating the integration process by handling (almost) everything as an organisationally trackable component. In Section 4.1.4 we clarify how the previous points can be leveraged by test-driven integration, a TDD-like technique which (for TC internal work) has proven effective for integrating disparate components from separate teams in as "lean" and "agile" a manner as possible. Finally, Section 4.1.5 is links to an auxiliary informative document outlining an “Operational Map”, including layers and sequence for evolving the production environment.

4.1.1 Top-level auto-monorepos

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). This was previously left open-ended regarding having exactly that or a single platform-wide monorepo. It since became more apparent that a hotfix process would be needed for each of the following:

- critical regressions, vulnerabilities, etc (especially security)
- Lab environment-module container releases (on their own more dynamic release schedule)
- possibly more subsystems which will also need this process

Due to that and to the explosion in complexity for each exceptional type when merging/rebasing for forward/backports without conflicts/history-rewriting on the main branch, it was decided that the *per deployment-type* model would be the simpler and more resilient variant to maintain. Also, an incidental advantage is that central effort and rigour regarding Quality Assurance (QA) and smooth integration can be prioritised for the preferred deployment-type for each component, implicitly incentivising component development teams to invest development-effort to modernise their deployment process. An example of this would be if a platform-component is deployed as a siloed stack on a VM(s) and virtual network(s) for purely technical-debt reasons (no technical reason for it *not* to be migrated to scalable pods - just human-resource/prioritisation-reasons). In this case the "siloed" deployment can still be accommodated but with a focus on optimising for a potential non-legacy deployment at a later date, also utilising shared protocols and data formats. This will allow TC to allocate, prioritise, and optimise for scaling and resilience (in the engineering and operations senses of each) - removing hurdles and streamlining support for integration of modernised deployments. It will also facilitate organisationally sandboxing (de-prioritising) coordination/integration efforts for components if they wait too long and become a legacy maintenance-burden. For any component that can't be maintained to SLA-standards (due to technical-debt or non-scalable deployment/maintenance-processes) it could be provided as a "contrib/best effort" add-on. More importantly, any fully integrated component should not depend on such an add-on.

4.1.2 Namespaced integration subdirectories

As was prescribed and implemented for software delivery and installation for the Lab (see Section 3.1), it was identified that namespacing by way of standardised base-directory names in component repositories should be encouraged (and eventually required) for *all* deployment types. 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 may 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.

Aiming towards platform-scale continuous delivery, and eventually continuous deployment of services to production as a distant goal (with no concrete timeline), for stable components the "EBRAINS" branch would eventually be maintained by EBRAINS DevOps (in the internal repo which previously mirrored the external "EBRAINS" branch). For exceptional contributions the component developers would then open merge-requests. Eventually this would only be gatekept by component unit-tests and EBRAINS system-tests, with deployment managed by transactional strategies (A/B releases, canaries, rollbacks, etc) as is done by e.g. Netflix. This could only be realistically attempted at a much later stage after microservices are well integrated with a combined central API gateway and service mesh, meaning probably a year or more after initial launch. This is due to the dynamic, evolving state of components expected until then.

4.1.3 Everything as a trackable component

One already established goal for which the implementation and coordination-requirements have become clearer is (as much as possible) treating absolutely everything as a component with a "component owner". This means requiring every "component owner" 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 ideally "just be another component". 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, Kong, 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)

All of these "components" would have their own "resource trees". This includes even base components (e.g. the container orchestration engine" - OKD) being treated operationally as just another component, in which case all containerised components would depend on the COE within their resource trees. The list of components would also include tools and services like Ansible for config/deployment-management and imperative operations, Terraform or Openstack Heat for Infrastructure as Code (IaC) & declarative operations, and Juju for application modelling, declarative operations & day-2 ops. Effectively all other components would at least indirectly depend on the above, instead of being manual installations (requiring “snowflake” maintenance). All deployed services and delivered products at Fenix RI (part of the "Enterprise System" layer, not the "System of Systems" layer) would then also list OpenStack cloud resources (or HPC/NMC resources, etc) as dependencies in their resource tree – the difference being that at that level those dependencies are considered "external resources" as described earlier. Moreover, in the long-term, all "external IaaS resources" (not “external PaaS/SaaS resources”) would only be directly depended on by the kind of DevOps/IaC tools mentioned above, and then indirectly depended on by everything above those. This results in an automatic hierarchy, ordering, and quantifiability of all operations. For example, the COE would need to be deployed and configured before any process-container-based components (ideally in an automated/idempotent way). OpenStack resources (VMs, networks, security-groups, floating IPs, ssh-keys, etc) would need to be deployed/provisioned (by Heat, Terraform, or similar) before components requiring those (which include even components like the COE).

Most importantly this quantification can be encouraged/enforced for 6 aspects of content:

- creation (when not an upstream product)
- maintenance/development (when not an upstream product)
- documentation (when not an upstream product)
- testing
- deployment/delivery
- operations

In distributed development models like that of EBRAINS components, strict quantification and tracking are mostly focused on the first four aspects above (note that system-testing is not yet tracked in EBRAINS). Treating all six aspects as trackable can help TC assessment of global integration status by answering – effectively for everything - questions like "who is the owner?", "has it been properly integrated & documented?", "has it regressed with respect to current EBRAINS?", "which team maintains it?". This even includes OKD, Collaboratory, and infra-provisioning template bundles. For external/upstream components though, the "component owner" would only be responsible for points 4-6 (as obviously 1-3 have “external” responsibility). Fully tracked components would include those TC/AISBL provides, but when tracked the same way would be included in all the same single source of truth locations (free-form documentation at first, structured data models later for automating conformance-tracking later).

This way there would be global maturity/readiness tracking of every component for "green" status, but also for its entire "resource tree", down to external resources. When anything stops being "green" (a regression) the upward-propagating impacts of that will be visible and concerned consumers further up the tree can be alerted, as a form of “burden tracking". This only applies to *shared* resources though - anything that a component vertically integrates within their component in an encapsulated way (that is "vendored in") effectively just becomes "part of the component", effectively invisible at the platform level. This is however at the cost of added maintenance-burden for component developers (and duplication of effort across components when many do such vendoring). A benefit of this method is that the automated CI/CD process should eventually be able to tie all components together in a scalable, automated and heterogenous way, into an operationally unified "infra" (with a visibly unified platform of orchestrated microservices at the core for intuitive UX). In this situation any non-componentised miscellaneous "glue" code or configs could be regarded as "technical debt" (i.e. when short-term pragmatic compromises are made).

4.1.4 Test-Driven Integration

One technique which was started internally, and is being assessed for possible external use, has been dubbed "TDI" (test-driven integration, analogous to test-driven development). For those unfamiliar with TDD (and therefore TDI), the short description is:

- "Red" (traffic light): Write tests which start out failing due to absence of functionality.
- "Green" (traffic light): Quickly write (non-optimised) functionality to pass the test.
- "Refactor": As much as possible within the SDLC (software-development lifecycle) improve functionality, optimality, and readability while remaining “green" - no regressions.

This came about for the same reasons TDD did in the development world. When having to pivot, adapt, onboard, with changing parameters, changing requirements, and changing external environments, a traditional "waterfall" process (and spec-driven development) sometimes falls short. Being "lean" and "agile" when integrating disparate components from separate teams without a strict methodology or formalised process also sometimes falls short and can become an exercise in post-hoc mitigation. Following TDI principles (largely for understanding and reference purposes), TC has created some "greenfield" hosts and installed low-level core functionality to them, and has started a process of working up from "first principles" on those, cloning existing layout and services - but doing so in an IaC, idempotent/self-healing, transactional/atomic, unified way. Although this is a useful practice by itself, the TDI part is that beyond the core elements we integrate components (from the “platform components” level up) into this environment in a system-test-driven way.

The top-level automated monorepos per deployment type start with a basic test that ensures no "Platform & RI component" repo is declared as a git submodule without there also existing a test-suite for it. Therefore, the first step of including a sub-repo is to add a suite of failing integration-tests as "tests of requirements" of the component. Once the repo is included and synced, if the tests don't all pass then issues must be addressed until they do. Those then remain as system regression-tests to catch high-level breaking changes coming from component repos. Whenever any untested drift causes system-breakage, then as well as developers fixing the components involved, the Definition Of Done should include DevOps engineers adding regression system-tests to catch the regression if it happens again. If the breakage can be tested in a component-agnostic platform-wide way then a test should be written as such, but if highly component-specific then the test should be targeted as appropriate, on a case-by-case basis. If we evolve this process beyond internal use, then the goal would be to eventually reach parity with the existing platform (deployed in various sites using various strategies due to historically evolving requirements). At that point the TDI-driven platform could be used to deprecate the existing environment it has cloned.

A nice side-effect should then be ensuring even in a disaster scenario that EBRAINS can effectively be redeployed from scratch in a highly automated way, recovering data from backups (and perhaps from snapshots where appropriate).

The tests themselves would look to software developers like blackbox testing (integration/system testing), with all web-facing backend functionality ideally being behind REST APIs, and with tests being API-calls. For components deployed on HPC the TDI-tests could probably be primarily scripted SSH commands, or minimal dedicated Jupyter notebooks (although the latter risks being akin to end-to-end testing - less useful for a test-driven process due to increased opacity and complexity). Test-driven software-development is usually driven by unit-tests, so the main difference here is the use of system/integration-tests, web-API-calls, scripts-over-SSH, etc as the progress-drivers rather than the small unit-test scripts usually associated with TDD.

4.1.5 *Operational Layers*

An in-depth “Operational Map” (including descriptions of the layers employed, and projected sequence for evolving the production environment) has been written and can be viewed at: https://drive.ebrains.eu/f/85e87627117046c58846/ (shared as auxiliary informative information.)

4.2 Software Quality

Software quality (SQ) as required across all EBRAINS software is defined by the EBRAINS Software Quality Working Group (ESQ-WG) which acts as a central focus point for discussions on the guidelines to developers for assuring software quality (D5.3, Section 4.2.2.1 [1]]) and operates in close collaboration with EBRAINS TC (Technical Coordination) to ensure alignment with all stakeholders involved. The ESQ-WG is organizing monthly meetings that are often supported by extra meetings for dedicated work on the software quality guidelines, while some of the extra meetings were dedicated to multi-faceted discussion with invited representatives from other working groups to avoid overlaps and establishing bi-directional links between the resulting guidelines.

The output of ESQ-WG is a live document including the abovementioned guidelines with requirements and best practices for EBRAINS developers [19], together with a checklist for the convenience of component owners. This checklist is an extract of the guidelines with the core points of the different categories, referencing back to details in the full document. As defined in ISO-25010 model, the following characteristics should be considered to evaluate the properties and the overall quality of a software product [20]:

- Functional suitability
- Performance efficiency
- Compatibility
- Usability
- Reliability
- Security
- Maintainability
- Portability

The checklist was designed to cover these quality characteristics and the included guidelines were weighted based on three priority key words, as known from the IETF RFCs [21] and they have also been described in detail at (D5.3, Section 4.2.2.2 [1]):

- **must**: Requirements - things that are necessary to require from component owners)
- **should**: Recommendations - things recommended as being better) (but not enforceable) for the operation & maintenance of the EBRAINS RI and/or the development & maintenance of the component.
- **may**: Best practices (“nice to haves”): Things that are very helpful but are so optional that even the term “recommendations” would be too strong.

Furthermore, the guidelines are separated into distinct categories as they are derived from the chapters of the guidelines document and are the following:

- Dependency Management
- Software Project Management
- Version Control
- Testing
- Documentation

- Code Quality
- Deployment
- Licensing

This document is continuously updated and accompanies the corresponding requirements of each EBRAINS Integration phase. Every component owner, during the integration process of their component, checks the corresponding fields of the list that is revisited whenever there is an important update, establishing thus a holistic and updated view of the component's quality state in cooperation with TC.

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 infrastructure to be delivered addresses the needs of the offerings.

The current section presents the update of the **metrics/KPIs** for monitoring and assuring the quality of the EBRAINS RI, specifically for the integration process, along with the achieved values for the first Phases of Integration as evaluated by the TC (Section 2.2). Further, we summarize our current status, the risks we have faced or are expected to face, and our planning for the immediate future. The most important achievement of TC so far, along with TC instruments can be found in Annex 8.1.

5.1 Quality assurance updates

EBRAINS RI quality assurance procedures, mechanisms and indicators ensure that: (a) EBRAINS services/components are regularly evaluated against predefined software quality and delivery criteria and are compatible with the infrastructure cornerstone assumptions, and (b) EBRAINS platform services are continuously monitored in terms of their availability and reliability, to mitigate potential risks.

Quality assurance procedures have been established by EBRAINS TC in order to coordinate the integration process and effectively tackle any related technical issues. These procedures are supported via a dedicated series of meetings, the formally scheduled “TC Weeklies” and “TC Task Force” meetings [1], as well as frequent ad-hoc “debrief” meetings for synchronising about KPI outputs (Section 5.1.1).

Metrics/KPIs for monitoring and assuring the quality of the EBRAINS RI platform specifically for the integration process (monitored by EBRAINS TC) have been detailed in D5.3 and have been further refined in this report. These KPIs are calculated from the aggregated scoring of a series of component-specific indicators.

5.1.1 KPIs

In this section, we detail the **metrics/KPIs** for monitoring and assuring the quality of the EBRAINS RI, specifically for the integration process. Their goal is to (a) facilitate components towards the satisfaction of the EBRAINS architecture and integration guidelines in a quantitative manner, and (b) and to report on the overall performance/operational status of the EBRAINS RI in its entirety. In the immediate future and pending the implementation and availability of the respective EBRAINS-wide monitoring facilities, KPIs for the RI’s overall engagement, collaboration and customer satisfaction will be integrated.

The KPIs list defined and monitored by the EBRAINS TC is presented in Table 3 and Table 4. For easy communication, the KPIs list was decomposed into two distinct categories (hence two tables). KPIs relate to project management, as well as to the development effort. An outline of the KPIs list follows, including sub-categories as presently decided:

- Administration and Management
    - Collaboration indicators
- Development and Operations
    - Documentation

      foundational aspect of the delivery of the RI, so dedicated KPIs keep track of the components’ efforts towards documenting their offerings.
    - Service Operations

KPIs exist to track conformance to the integration/delivery guidelines and to monitor "EBRAINS APIs adoption" and timely implementation of EBRAINS physical deployment plans.

- Component maturity

  KPIs exist to track the number of components that are using EBRAINS API and the physical deployment progress for both VM-based and container-based services.

- Responsiveness/Readiness

  The KPIs based on metrics from TC Gitlab issues are for tracking overall responsiveness/readiness of the TC and the development teams, assuming that a very small number of issues should be active for more than a couple of weeks. When such issues stagnate, particularly in the "blocked" state, that usually indicates a need for division into subtasks, soliciting escalation and/or external assistance to facilitate progress, or identifying insufficient/missing resources. Where "TC Gitlab" is referenced in the KPIs, note that this refers to the Gitlab board used for logging of issues relating to the overall integration process, which are created by component owners and the Technical Coordination team. References to TC Gitlab "active issues" mean all issues presently labelled "open", "backlog", or "in-progress". Issues labelled "open" have been recently opened and require preparation, those labelled "backlog" have been adequately prepared (data collection done, done-criteria specified, adequately described, ready to be assigned for work, including difficulty estimation where appropriate), and those labelled "in-progress" have had work started on them.

Please note that the KPIs are intended for goal setting and intention-signaling 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, including specifying which are "point in time" (with schedule for those that will not be continuous), 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 3: KPIs - Administration and Management

Administration and Management	KPI name	KPI description	Target value at M21
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.	>80%
	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.	<2 weeks

Table 4: KPIs - Development and Operation

Development and Operations	KPI name	KPI description	Target value at M21
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.	>40%

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.	>20%
	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.	>40%
	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.	>40%
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 5 and Table 6, we present the values of the defined KPIs for (a) all EBRAINS components (as of M21), (b) only Phase 1 components (start of Phase 1, end of Phase 1, M21), (c) only Phase 2 components (start of Phase 2, M21)

As we can observe in in Table 5, Table 6 and Figure 17, when components officially enter an integration phase, they participate more actively in the different TC processes (meetings, Gitlab Kanban board etc.) and satisfy their integration objectives. Moreover, the values for Phase 2 components are better than the ones of Phase 1, an observation easily explained by the more effective preparation of these components, since they were aware of the general integration requirements even before the start of Phase 2. We can anticipate that Phase 3 components will present even better results in due time.

Regarding TC weeklies and the Gitlab Kanban Board ([1]) used for creating issues and keeping track of development progress and effort, the number of active issues and percentage of issues open for more than 3 weeks demonstrates the progressing elimination of blocking points relating to the development of the EBRAINS RI and the as well as the good progress of the integration process on an RI-level.

Moreover, the prioritisation process put into place, in which different technical priorities are being identified by SCs, evaluated by TC and added as issues in the GitLab Kanban for monitoring and coordinating their implementation, seems effective, with only 58% of the SC prioritisation issues being active more than 3 weeks. In addition to that, SC prioritisation issues are “in progress” soon after their creation by SC members (i.e. no SC prioritisation issues are open, with only one issue in the backlog).

Table 5: KPIs - Administration and Management-values

Administration and Management	KPI name	All components - M21	Phase 1 components.- beginning Phase 1 (M4)	Phase 1 components.- beginning Phase 2 (M13)	Phase 1 components.- M21	Phase 2 components.- beginning Phase 2 (M13)	Phase 2 components.- M21
Collaboration indicators	Participation in TC Weekly meetings	68.18%	87.50%	87.5%	87.5%	93.75%	93.75%
	Wiki pages update response time (weeks)	4.85	1.13	1.25	1.25	11.09	11.06

Table 6: KPIs - Development and Operation - values

Development and Operations	KPI name	All components - M21	Phase 1 components.- beginning Phase 1 (M4)	Phase 1 components.- beginning Phase 2 (M13)	Phase 1 components.- M21	Phase 2 components.- beginning Phase 2 (M13)	Phase 2 components.- M21
Documentation	Developer Documentation	67.27%	75%	75%	75%	87.5%	87.5%
	Maintainer/Operator Documentation	29.09%	75%	75%	75%	25%	25%
	End-User documentation	81.82%	100%	100%	100%	100%	100%
Service Operations	Deployment to testing	42.31%	75%	75%	75%	69.23%	69.23%
	Deployment to staging	17.31%	75%	75%	75%	15.38%	15.38%
	Deployment to production	65.38%	75%	75%	75%	92.31%	92.31%
Component maturity	EBRAINS APIs adoption	62.22%	50%	50%	50%	92.31%	92.31%
	VMs Deployment	26.53%	50%	50%	50%	58.33%	58.33%
	Containers Deployment	36.73	50	50	50	66.67	66.67%
Responsiveness/ Readiness	Turnaround of TC Gitlab active issues.	69%					
	TC Gitlab active issues	62					
	Turnaround of TC Gitlab active SC prioritization issues.	58%					
	TC Gitlab active SC prioritization issues.	12					
	TC Gitlab open issues	18					
	TC Gitlab backlog issues	27					

	TC Gitlab in-progress issues	17					
	TC Gitlab open SC prioritization issues	0					
	TC Gitlab backlog SC prioritisation issues	1					
	TC Gitlab in-progress SC prioritisation issues	11					
	Automated tests	43.64%	50%	50%	50%	43.75%	43.75%

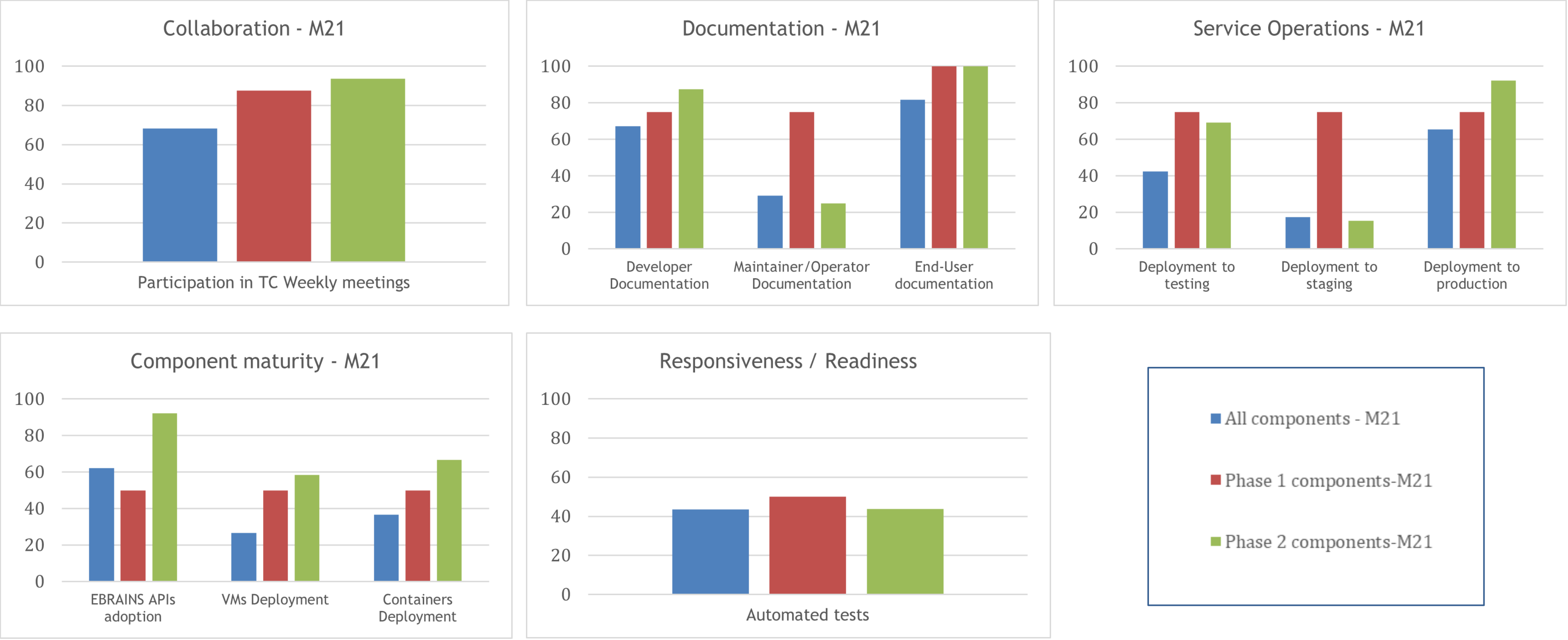


Figure 17: Components in M21

5.2 Current status

Currently, our main focus is the integration of the EBRAINS components. **Phase 2** is about to end (M21), and **Phase 3** is around the corner (Section 2.2). A final assessment of **Phase 2 components** is taking place, before they continue their journey in **Phase 3**. All remaining components are expected to be integrated during Phase 3. – but not all of them from the beginning of the Phase. New components will gradually join after assessment and communication with the TC team.

Moreover, all **Working Groups** are active, offering their feedback and support, with TC closely collaborating with **HLST** - especially for component support and documentation -, **SLU** -for a better understanding of the showcases and of the scientific part of the project in general-, and with **Fenix Infrastructure** for the necessary resources. In addition, and in collaboration with **SLU** and **WP1-3**, define **new use cases** and new technical requirements for the EBRAINS RI.

In parallel, besides the current collaboration tools and technical instruments to EBRAINS component owners and users (e.g. Collab, Gitlab, VM services, communication services etc.), we have initiated the development and gradual provision of a series of new tools and offerings, as described in Section 3.

5.3 Delays and Risks

During the first half of SGA3, there were slight delays and risks we had to mitigate.

In many cases, the component owners lacked information about TC approach, so they were not following the defined integration guidelines. The most important missing piece of information for TC was the obsolete description of some components in TC collab. Updated information is necessary for the right assessment of a component and its participation in the integration process, so TC is actively reminding all the components owners to update their wiki pages, while it arranges meetings and adapted presentation for the component owners who have specific questions or are not familiar with the process.

Another risk for TC, already mentioned in Section 2.6, is the fact that many base infrastructure facilities are still slowly coming online to full capacity. There are some sites that have almost all of their infrastructure resources readily available, some that have a good percentage of their resources available and some that are in the final stages of making available the full extent of their offerings. The EBRAINS RI must be able to fully utilise the dedicated resources and operate in a multi-site environment in order to be able 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). TC is collaborating with Fenix Infrastructure to avoid perpetuation of this issue.

Scientists and developers are two different worlds that should collaborate and understand the different needs and requirements. Bridging the gap between these two worlds is not always easy, with the scientists ignoring useful tools for their research and the developers not knowing the services that can be developed based on the research conducted in SGA3. TC, in close collaboration with SLU, through different meetings and presentations and through the definition of newly defined, small use cases, is trying to understand the scientific approach and demonstrate to HBP scientists what they can find in EBRAINS RI and how they can use the different services.

5.4 Planning for next months

Phase 3 of Integration is starting next month, and our aim is to integrate 100% of current components into the EBRAINS RI as it is described in Sections 2.2 and 2.6. The architectural diagrams will be further updated in order to provide all the necessary information.

As far as standardised computational workflows are concerned, our goal for the coming months is to provide EBRAINS users with functional, tested options for defining their workflows, executing them on EBRAINS infrastructure and publishing them in an EBRAINS registry. Our main focus is finalising the options that will be available to EBRAINS users for executing their workflows. Depending on their dedicated resources, users will in the future be able to run their workflows either on HPC systems, using the UNICORE workflow manager, or on the OKD cluster, via a dedicated endpoint. Another essential aspect of the Technical Coordination efforts to establish standardised workflows as an essential part of EBRAINS RI is creating a searchable EBRAINS workflow registry, where standardised workflows defined in CWL as workflow recipes will be published along with their metadata, in order to be easily accessible and findable. To that end, there is an ongoing discussion with the Knowledge Graph team, and it has been decided that the metadata schema (OpenMINDS[11]) will be modified to also include descriptions of workflows and tools able to be used as workflow steps. This way, any workflow created by EBRAINS users will be able to be published in the Knowledge Graph, given a unique identifier and linked to all the related existing software, models and data. Integration with the Provenance API (Section 3.6.1) is also planned for the next months, so that the workflows provenance information can be tracked and stored.

Of course, Technical Coordination team will also continue assisting component owners to proceed with the integration of their components and scientists to familiarize with the processes of workflow definition, execution and publication, and alleviating any misconceptions that may occur. Documentation, guidelines, tutorials and support will be provided. Additionally, our collaboration with the Showcases, as well as other EBRAINS teams that work on valuable use cases, will also continue in the coming months.

6. Looking forward

Three are the main axes of our work. The first one is Users: we want EBRAINS RI to fulfil the needs of users of all types, so we are trying to follow suggestions and needs, bridging the gap between developers and scientists. The second axis is Tools: we want to facilitate results, not just provide tools, and make the research activity easy, quick, and accessible. The third one is Engineering: we want our platform to be scalable, secure, maintainable, affordable

We want EBRAINS to be part of the future, designed to adapt to new technologies and to changing needs of scientific community and evolving user needs We know that our expertise is on the IT side, not on the scientific side which it serves, so we are trying to learn from our collaboration with the different types of users.

We would like to help the showcases evolve to highlight how much easier, quicker, more expressive, and more accessible the usage becomes as the platform and ecosystem mature, compared to independent tools and toolsets. At the same time, we are trying to remain focused on "what results I need" rather than "what tools I run to get results".

Finally, the Technical Coordination team has identified the need to have a central Dashboard that will allow users to import and share their workflows, monitor their execution progress, collect logs and results, through an intuitive, easy to use Graphical User Interface (GUI). For the time being, we are in the process of identifying user requirements, mainly through our work for components' integration and our collaboration with HBP instruments.

[11] https://github.com/HumanBrainProject/openMINDS

7. References

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

[2] https://c4model.com/

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

[4] https://www.elastic.co/what-is/elk-stack

[5] https://spack.io/about

[6] https://spack-tutorial.readthedocs.io/en/latest/

[7] https://computing.llnl.gov/projects/spack-hpc-package-manager

[8] https://www.slideshare.net/insideHPC/spack-a-package-manager-for-hpc

[9] https://www.alcf.anl.gov/support-center/training-assets/software-deployment-spack

[10] https://oaciss.uoregon.edu/NSFDOE19/talks/spack-nsf-doe-deployment-intro.pdf

[11] https://www.elixir-europe.org/

[12] https://www.eosc-life.eu/

[13] https://www.ga4gh.org/

[14] https://workflowhub.eu/workflows

[15] https://www.myexperiment.org/workflows

[16] https://github.com/ga4gh/task-execution-schemas

[17] https://github.com/ohsu-comp-bio/cwl-tes

[18] https://github.com/elixir-cloud-aai/TESK

[19] https://drive.ebrains.eu/lib/5ff9d79d-711f-4fba-953f-0cf83bab7f0b/file/HBP_SGA3_D5.3_Annex_IX.docx

[20] https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

[21] https://tools.ietf.org/html/rfc2119

[22] https://robotframework.org/

[23] https://matomo.org/

[24] https://uptimerobot.com/

[25] https://www.elastic.co/guide/en/welcome-to-elastic/current/welcome-to-elastic.html

[26] https://www.elastic.co/beats/

[27] https://www.elastic.co/beats/metricbeat

[28] https://www.elastic.co/beats/filebeat

[29] https://www.elastic.co/guide/en/apm/get-started/current/components.html

[30] https://nmpi.hbpneuromorphic.eu/dashboard/

8. Annexes

In this section, we collected updates and documents that can offer additional insights and details to the reader of this deliverable. For the long documents, a small description is provided along with the URL to the respective document.

8.1 TC instruments and main achievements

EBRAINS Technical Coordination (TC) is responsible for the planning, design, integration and delivery of the EBRAINS RI.

TC work began with the creation of a comprehensive, EBRAINS RI components catalogue, based on the initial input of all SGA3 partners delivered during our 3-day TC Kick-off meeting (M2). This catalogue (available in TC Collab) is the foundation for sharing and monitoring up to date information per EBRAINS component and for planning the different phases of development, integration and delivery activities.

Further, five working groups (WGs), providing directions, discussion points, and critique on a series of cross-RI areas, have been established (EBRAINS Architecture WG – ‘**EAI-WG**’, EBRAINS Software Quality WG – ‘**ESQ-WG**’, EBRAINS Software Delivery WG – ‘**ESD-WG**’, EBRAINS Security WG – ‘**EIS-WG**’ and EBRAINS-APIs WG – ‘**EA-WG**’), providing valuable feedback. The WGs will remain active throughout SGA3, offering their continuous feedback and support.

The major meeting of EBRAINS TC is **TC weekly**. TC weeklies are open meetings being held once a week, monitoring technical progress at the RI level; they have already become a meeting point of all the technical people of the project. TC weeklies are open to all members of SGA3 and bring together more than **60 participants**. a **GitLab Kanban** is used for creating issues, keeping track of progress and resolving any blocking points. All members of the SGA3 are welcome to create issues – cards. Until now, more than two hundred issues have been closed. Eight months ago, we introduced the definition of **SC priorities**. Every **Service Category** (SC) define 2-3 priorities as issues for TC weekly. New priorities are defined as soon as another priority issue is closed.

All the information available for the Technical Coordination, the different files that contribute to a successful collaboration and the minutes of the meetings are available in the collaboratory maintained by TC (**Technical Collaboration collab**), while the information regarding TC weeklies is available in the respective collab (**Tech Coordination weeklies**). TC Collab is the central collaboration point on all TC aspects and is expected to be used by all TC instruments to facilitate coordination and planning activities, flow of information and knowledge transfer.

As presented in Section 2.2, TC has defined a pragmatic high-level roadmap towards the delivery of EBRAINS, comprising **four phases of integration**. During **Phase 1** (M4 -M12), four components have been selected in order to provide a manageable, yet representative, assembly of technical maturity, established development and testing processes, and ongoing deployment practices. We used Phase 1 to identify potential issues and blocking points and to test and improve the defined guidelines. In **Phase 2** (M13- M21), 16 components are participating (plus 4 from Phase 1). For Phase 2, we used the output of Phase 1, in order to improve integration requirements and tackle possible issues. Several discussions and meetings took place with Phase 2 components and in some cases, we have also extracted component-specific guidelines. Phase 2 components will continue their journey in **Phase 3** (M22-M33).

Further, a closer and structured collaboration with the **Service Categories (SCs)** and the **ICEI** has been designed and implemented, further expanding the TC instruments of SGA3, and in response to the need for prioritizing EBRAINS-wide technical aspects and issues of critical nature for the delivery of SCs and ICEI output. As already mentioned, the prioritisation process, where different technical priorities are being identified by SCs, evaluated by TC, integrated in the TC Weeklies context for monitoring and coordinating their implementation, is already in place.

Finally, collaboration with the **SLU** and **HLST** has intensified, ensuring complementarity and best use of resources for cross-cutting areas, thus ensuring that software development, integration, and support are in line with the needs of the EBRAINS user community. At the same time, we are collaborating with the scientific WPs of HBP in order to identify scientists' needs and bridge the two worlds of scientists and developers.

8.2 Phase 2 requirements

In Phase 2, the aim was to integrate nearly 50% of the currently listed components to the EBRAINS RI and, at the same time, ensure integration practices' full conformance with Deliverable "D5.3 EBRAINS Technical Coordination Guidelines" ([1]). Significant progress has been achieved in all components that participated in Phase 2 regarding testing, automation, development, documentation, and deployment (where applicable). To achieve this in a standardised manner and be able to track the 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. In essence this integration requirements list concretizes the "EBRAINS Technical Coordination Guidelines" as presented in D5.3. This is the same approach that was successfully adopted during Phase 1. The integration requirements is a densely formatted list that can be easily referenced by the component teams without much overhead. For Phase 1 the integration requirements were broken into three levels, "Required", "Welcomed" and "Suggestions". During Phase 2 this structure was reformatted to two levels, namely, "Baseline" and "For applications/services". This was considered more 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. As with Phase 1 requirements, again, with respect to the requirements' list, the ordering is random and does not imply that one requirement is more important than others.

Phase 2 integration requirements that governed this integration phase can be retrieved from the following link: https://drive.ebrains.eu/f/cea0c03e0c43465a8215/

8.3 Software Quality guidelines

The document of Software Quality WG, mentioned in Section 4.2, including guidelines with requirements and best practices for EBRAINS developers, together with a checklist for the convenience of component owners, can be found at:

https://drive.ebrains.eu/f/93cc55bb92b2433eae68/

Please note that this is the version of the document available in M21.

8.4 Software Delivery and Installation for the Lab

Towards seamless integration of the various components to the EBRAINS platform, there is a particular Use Case concerning the "Build, Delivery and Activation of software for direct use in Jupyter Notebooks in the Collaboratory Lab environment (from now, the Lab)". More specifically, how to efficiently and optimally build, deliver and activate the various simulation engines and other software that participate in the EBRAINS integration activities and make them available to Collaboratory users in the Lab, without requiring from them to perform any action to install the tools, and at the same time offer a streamlined process to the various Component Owners (COs)/software developers, through which they can make their software available to EBRAINS users in the Lab.

A delivery strategy was implemented during Phase 2 that addresses the requirements of the aforementioned Use Case. The full solution for the developed delivery strategy for "software delivery and installation for the Lab" is described in dedicated documentation material. All the aspects of the delivery strategy, namely, design, implementation, all involved processes, and maintenance are included. There are several actors that participate in the processes, namely, the end-users, testers, developers, and DevOps/Administrators and so a dedicated section for each actor exists in the documentation to provide all the required information and facilitate the integration activities.

The full documentation of the delivery strategy can be retrieved from the following link: https://drive.ebrains.eu/d/050db3f541684ea0ae4e/. The material will be continuously updated to reflect the latest development status.

8.5 Workflows documents

During Phase 2 the work with respect to standardised scientific computational workflows that is referenced in Section 3.2 can be retrieved from the folder:

https://drive.ebrains.eu/d/f19c6cb6e6e94eafb3ed/.

The first document is dedicated -but not limited- to scientists of the consortium and can be found under https://drive.ebrains.eu/f/63647b7c0bb44675864b/.

The second one is dedicated –but not limited- to developers of the consortium and can be found under https://drive.ebrains.eu/f/49f5aa3cecbe4a89be27/.

8.6 Integration assessment template for components progress

Phase 2 requirements provided a list of specific requirements that need to be fulfilled, to streamline the integration activities and centre them around common action items. With the purpose of evaluating the component teams progress against the integration requirements, a template was created based on the requirements list. This template was named "integration assessment template" and 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. The template holds several questions that can be answered clearly with "yes" or "no," greatly simplifying (offline) communication between numerous teams. In that way unresolved items can be identified quickly, and mitigation measures can be applied effectively.

The integration assessment template is broken into three levels. These are the following: "Requirements", "Extras", and "Future". "Requirements" level which directly relates to points in the integration requirements list is further expanded to "General" and "Services" to match the categorisation of Phase 2 integration requirements ["Baseline" (=General) and "For applications/services" (=Services)]. The provided questions in the template in the "Requirements" level, aid in determining mostly the technical progress performed by a team at any given time. "Extras" level concerns either points relevant to very specific EBRAINS offerings (like Desktop applications) or general governance points (not technical) but important to check whether they are fulfilled. "Future" level concerns information on the vision of the components' development after the end of SGA3 and attempts to assess the components' long-term commitment to the RI, and the community in general, in an effort 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/f92faa736b4a48ceb465/

8.7 Service Category Illustration

In this Annex we include the illustration of a Service Category as presented in the (D5.3, Section 3.5, [1]) for completeness and as an example of the diagrams for the various SCs that need to be created.

It is obvious that EBRAINS is far too wide in scope to approach it in its entirety with only the set of diagrams presented in the Architecture's section. Different audiences "demand" respective point of views, that will enable them to grasp the RI at the appropriate level of abstraction. This is achieved by detailing only the "in scope" architectural elements and as a result creating various views of the RI.

The different audiences (within SGA3 consortium) are in essence the different **Service Categories** (SCs) and a required exercise is to illustrate each SC and its understanding of the EBRAINS RI architecture to ensure that all aspects of the architecture are adequately informed in the appropriate manner.

This section serves as a kind of preamble of the next phases of mapping the EBRAINS RI architecture. The target is to create **one diagram per SC** to best inform the target user communities about the architecture. Towards this objective, a diagram (Figure 18) for Service Category #3 (SC3) "Simulation" is presented to illustrate the scope of the current work and give an idea of what is expected to follow for all SCs. The diagram in this section is a "work in progress" and is not finalised. It has been created through an iterative process between the architecture team (EAI-WG) and the stakeholders and it displays the understanding of the current status. This co-design process has been started for the other SCs and some science cases but is not yet at a level that it can be presented here. In a synergistic process capitalising on the shared visual language and description of the different components in the architecture, individual Service Categories, Work Packages and core infrastructure functionality, EBRAINS science cases have several extreme requirements, such as co-deployment of multiple applications with human interaction; real-time requirements when interacting with robots or future hardware; and provenance tracking, monitoring, user access control and SLA (Service Level Agreement) / Pricing requirements which are abstracted by the "Resource Management and administration" layer at the diagram. The requirements of this layer are unique at the size of EBRAINS, and dedicated effort has been spent on finding solutions to this challenge. What is presented here is the current proposition for (a part of) the resource management layer. Figure 18 and future diagrams will be further formalised.

EBRAINS science cases have several extreme requirements, such as co-deployment of multiple applications with human interaction; real-time requirements when interacting with robots or future hardware; and provenance tracking, monitoring, user access control and SLA (Service Level Agreement) / Pricing requirements which are abstracted by the "Resource Management and administration" layer at the diagram. The requirements of this layer are unique at the size of EBRAINS, and dedicated effort has been spent on finding solutions to this challenge. What is presented here is the current proposition for (a part of) the resource management layer.

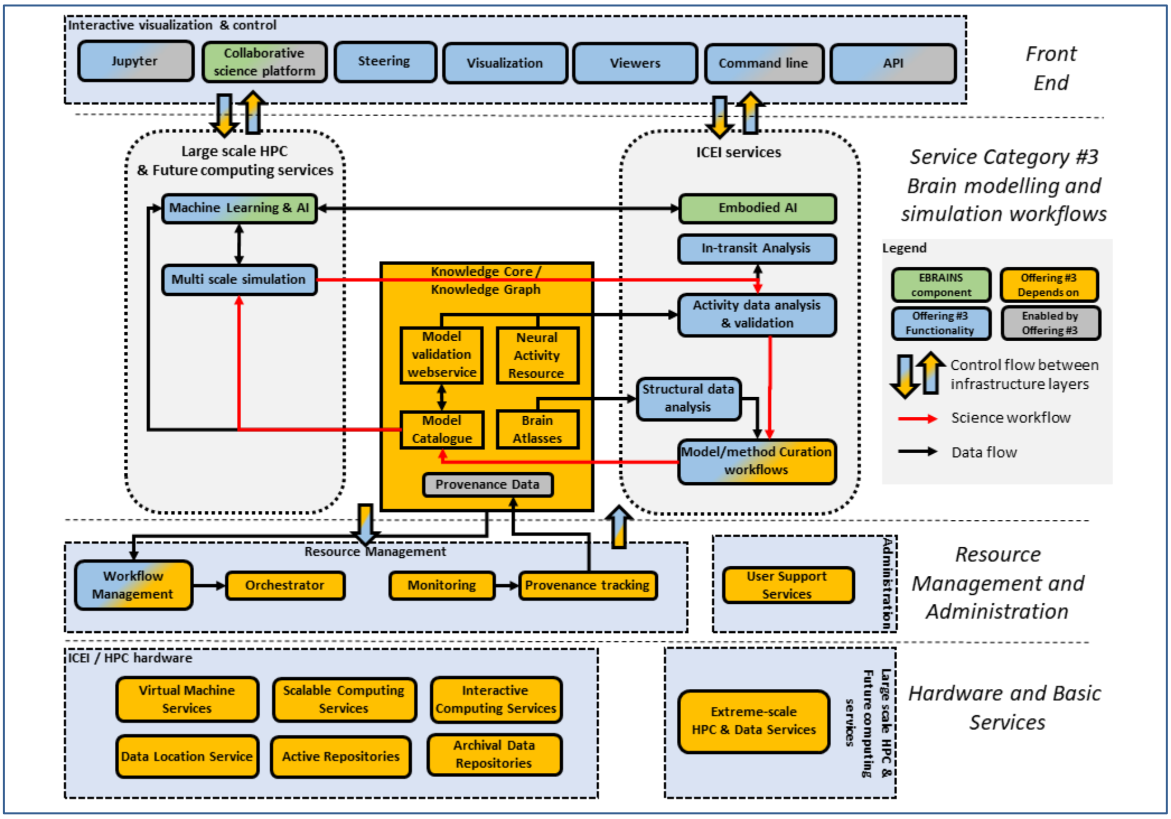

Interactive visualization & control
Jupyter
Collaborative science platform
Steering
Visualization
Viewers
Command line
API
Front End
Large scale HPC & Future computing services
Machine Learning & AI
Multi scale simulation
ICEI services
Embodied AI
In-transit Analysis
Activity data analysis & validation
Structural data analysis
Model/method Curation workflows
Knowledge Core / Knowledge Graph
Model validation webservice
Neural Activity Resource
Model Catalogue
Brain Atlasses
Provenance Data
Service Category #3 Brain modelling and simulation workflows
Legend
EBRAINS component
Offering #3 Depends on
Offering #3 Functionality
Enabled by Offering #3
Control flow between infrastructure layers
Science workflow
Data flow
Resource Management
Workflow Management
Orchestrator
Monitoring
Provenance tracking
User Support Services
Administration
Resource Management and Administration
ICEI / HPC hardware
Virtual Machine Services
Scalable Computing Services
Interactive Computing Services
Data Location Service
Active Repositories
Archival Data Repositories
Extreme-scale HPC & Data Services
Large scale HPC & Future computing services
Hardware and Basic Services

Figure 18: The current shared understanding of SC3 functionality and its relationship with the EBRAINS RI architecture[12]

8.8 Monitoring

In Section 8.8.1 information is provided regarding the service level monitoring. In Section 8.8.2 the Web Analytics platform currently used in EBRAINS RI is introduced. Finally, in Section 8.8.3 the prototyped back-end monitoring services are described.

8.8.1 Service level monitoring.

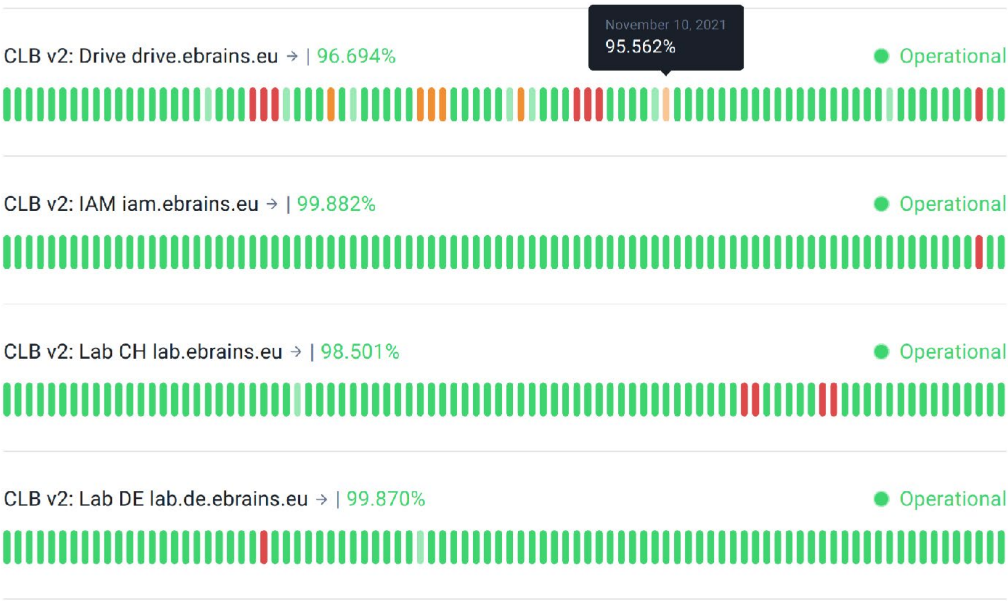
External monitoring at the service level is applied to determine the availability of EBRAINS services to the end-users in real-time. Using the UptimeRobot service [24], alerting mechanisms are in place to notify of service unavailability. As displayed in Figure 19, the operational status and uptime of the various EBRAINS services is measured.

Global status page (publicly available): https://stats.uptimerobot.com/6WNp1IoGo2

List of the monitored services:

https://wiki.ebrains.eu/bin/view/Collabs/infrastructure/platform%20monitoring/external%20monitoring/ .

[12] An important wish of the stakeholders was the explicit inclusion of their science workflow, marked with red arrows. The Knowledge Core/ Knowledge Graph box is not defined in this offering and should be seen as illustrative at this stage.



November 10, 2021
95.562%

CLB v2: Drive drive.ebrains.eu → | 96.694% ● Operational

CLB v2: IAM iam.ebrains.eu → | 99.882% ● Operational

CLB v2: Lab CH lab.ebrains.eu → | 98.501% ● Operational

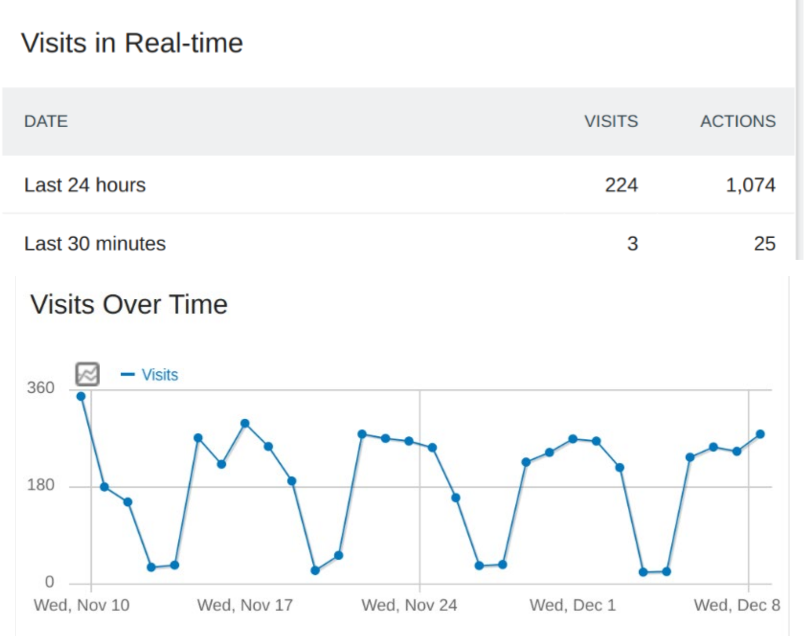
CLB v2: Lab DE lab.de.ebrains.eu → | 99.870% ● Operational

Figure 19: Service level monitoring example

8.8.2 *Web Analytics Platform*

Matomo Web Analytics Platform is currently installed for calculating web metrics for the various EBRAINS websites. It provides detailed web analytics reports about each site and its visitors. Visualizations (Figure 20 and Figure 21), provide real-time and cumulative user visits statistics with real-time geolocation identification of each visit as well. For the various pages of a particular site, user behaviour is also tracked by enumerating page views, bounce rate, average time on a page, etc. (Figure 22).

Matomo will be one of collected sources of data that will be aggregated in a central data store as mentioned in the introduction. The central data store offers the possibility to apply further analytics on the collected metrics by aggregating them centrally with data from other sources. Consequently Platform-level Administration Dashboards can be created to provide a holistic overview of the EBRAINS RI.



Visits in Real-time

DATE	VISITS	ACTIONS
Last 24 hours	224	1,074
Last 30 minutes	3	25

Visits Over Time

Figure 20: Real-time user visits and cumulative visits

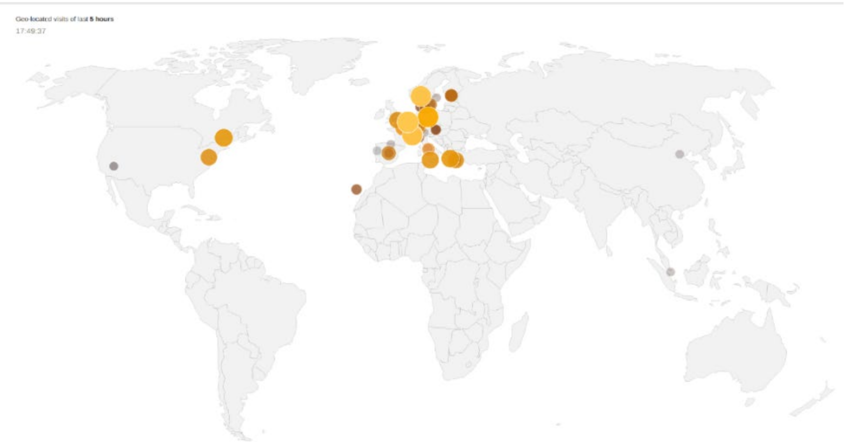
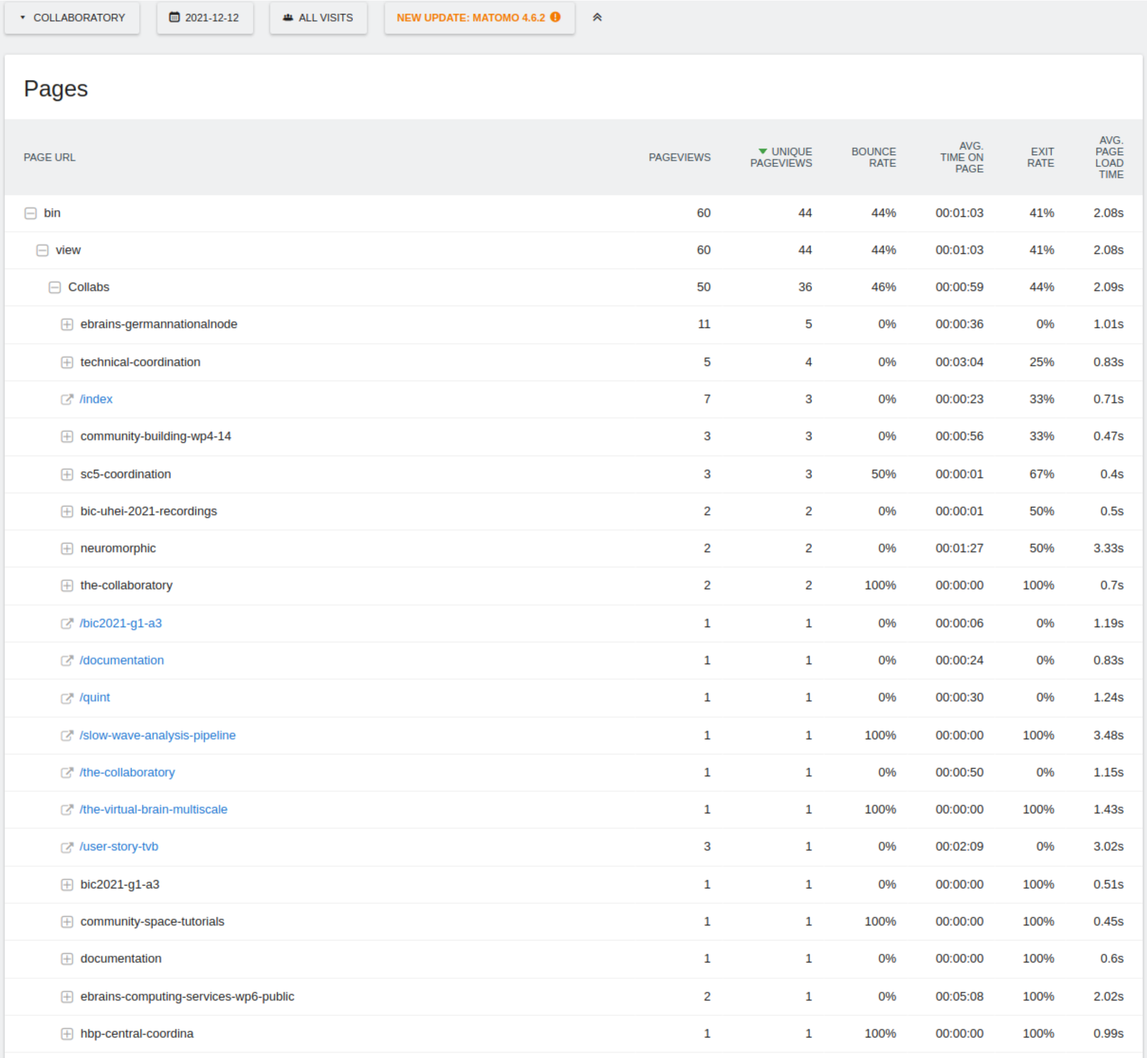


Figure 21: map for real-time geolocation tracking



Pages

PAGE URL	PAGEVIEWS	UNIQUE PAGEVIEWS	BOUNCE RATE	AVG. TIME ON PAGE	EXIT RATE	AVG. PAGE LOAD TIME
bin	60	44	44%	00:01:03	41%	2.08s
view	60	44	44%	00:01:03	41%	2.08s
Collabs	50	36	46%	00:00:59	44%	2.09s
ebrains-germannationalnode	11	5	0%	00:00:36	0%	1.01s
technical-coordination	5	4	0%	00:03:04	25%	0.83s
/index	7	3	0%	00:00:23	33%	0.71s
community-building-wp4-14	3	3	0%	00:00:56	33%	0.47s
sc5-coordination	3	3	50%	00:00:01	67%	0.4s
bic-uhei-2021-recordings	2	2	0%	00:00:01	50%	0.5s
neuromorphic	2	2	0%	00:01:27	50%	3.33s
the-collaboratory	2	2	100%	00:00:00	100%	0.7s
/bic2021-g1-a3	1	1	0%	00:00:06	0%	1.19s
/documentation	1	1	0%	00:00:24	0%	0.83s
/quint	1	1	0%	00:00:30	0%	1.24s
/slow-wave-analysis-pipeline	1	1	100%	00:00:00	100%	3.48s
/the-collaboratory	1	1	0%	00:00:50	0%	1.15s
/the-virtual-brain-multiscale	1	1	100%	00:00:00	100%	1.43s
/user-story-tvb	3	1	0%	00:02:09	0%	3.02s
bic2021-g1-a3	1	1	0%	00:00:00	100%	0.51s
community-space-tutorials	1	1	100%	00:00:00	100%	0.45s
documentation	1	1	0%	00:00:00	100%	0.6s
ebrains-computing-services-wp6-public	2	1	0%	00:05:08	100%	2.02s
hbp-central-coordina	1	1	100%	00:00:00	100%	0.99s

Figure 22: Metrics from tracking a particular web application

8.8.3 *Back-end monitoring*

This refers to the services that can be installed and configured in an application's back-end and can be used to monitor several crucial issues. The central data store (i.e. Elastic Stack) ingests data from back-end services to collect and visualize performance metrics from EBRAINS components/applications/services. During Phase 3, activities will focus on two main levels from which performance metrics can be collected. Namely, the Operating System and the Application level.

8.8.3.1 Operating System (OS) performance monitoring

This level of monitoring is done at the Operating System (OS) level, providing basic system performance metrics (e.g. CPU performance, disk usage, memory usage, network I/O) and ensuring that some important OS processes are running. Metrics are collected by dedicated data shippers (from now on "agents") which are installed on a component's server, provide the metrics to Elastic Stack, and offer monitoring insights at the operating system level. In EBRAINS, specific lightweight agents named "beats" [26] have been selected to connect and send performance data to the central data store. Specifically, metricbeat [27] is utilized for collecting basic system-level performance metrics while filebeat forwards logs and files to the central data store. The logs can then be visualized in the dedicated Logs User Interface (UI) (i.e. Kibana dashboards) and can also be filtered by several criteria (service, app, host) to track down non-standard behaviour [28].

8.8.3.2 Application Performance Monitoring (APM)

APM refers to monitoring each application uniquely and must be designed in cooperation with the corresponding component owner(s). It also depends on the technology stack used by each component. APM gives the possibility to instrument the application code and collect performance data and errors at run time. This data is collected via the dedicated APM agents which are open-source libraries (like the beats agents used in OS level monitoring) but written in the same language as their target service. For instance, in case the application code of an EBRAINS component is written in Java, dedicated Java agents must be installed and configured to collect application-specific performance metrics.

Since ELK Stack is intended to be the monitoring stack in EBRAINS, Elastic APM agents will be used. They can automatically pick up agent-specific metrics (e.g. JVM metrics in Java Agent) and identify specific errors at runtime in production. The APM agents will then send the metrics to an APM server which transforms the data into Elastic Stack documents and stores them in the corresponding indices [29]. The data can finally be visualised in dedicated (Kibana) dashboards.