EBRAINS Technical Coordination Guidelines (D5.3 – SGA3)

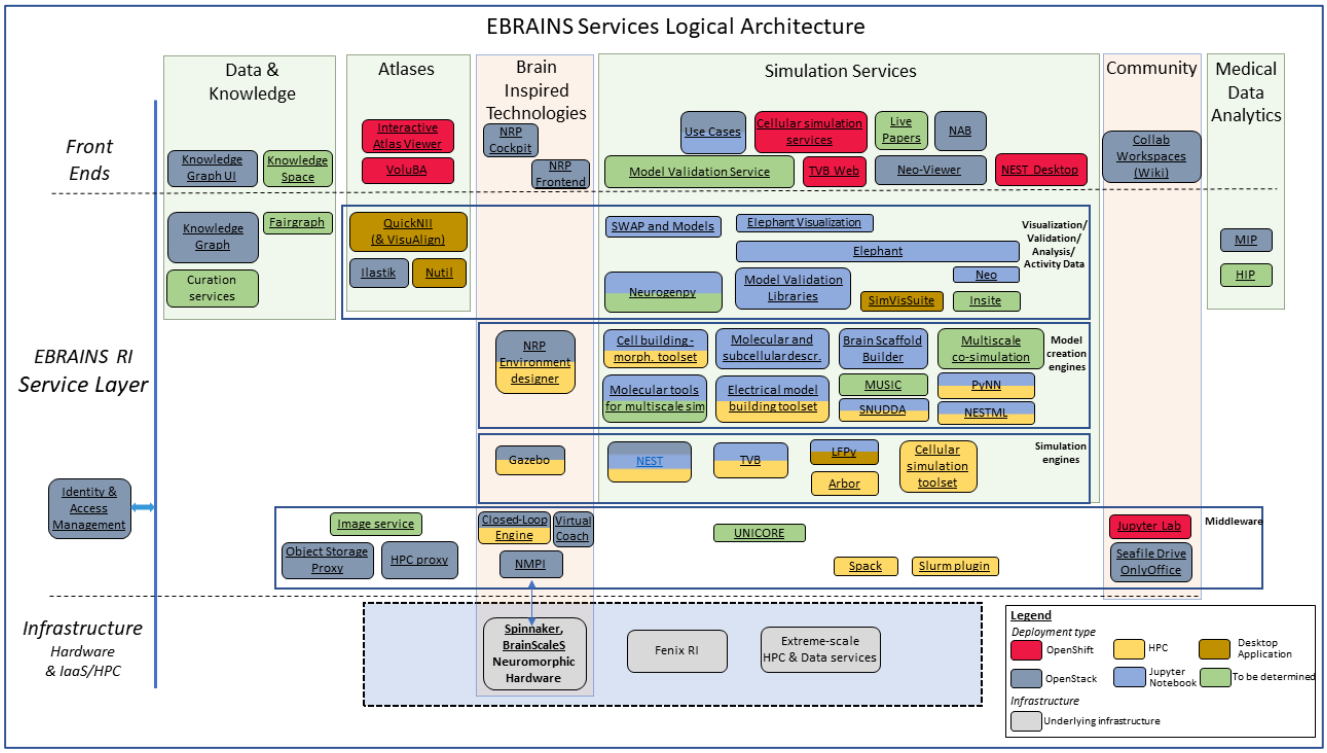

Figure 1: EBRAINS Services Logical Architecture (Section 3)

Project Number:	945539	Project Title:	HBP SGA3

Document Title:	EBRAINS Technical Coordination Guidelines
Document Filename:	D5.3 (D50) SGA3 M10 ACCEPTED 220520.docx
Deliverable Number:	SGA3 D5.3 (D50)
Deliverable Type:	Report
Dissemination Level:	PU = Public
Planned Delivery Date:	SGA3 M10 / 31 Jan 2021
Actual Delivery Date:	SGA3 M13 / 29 Apr 2021; accepted 20 May 2022
Author(s):	Amaryllis RAOUZAIOU, ATHENA (P133), Thanassis KARMAS, ATHENA (P133), Rowan THORPE, ATHENA (P133), Spiros ATHANASIOU, ATHENA (P133), Sofia KARVOUNARI, ATHENA (P133), George ANDREOU, ATHENA (P133), Evita MAILLI, ATHENA (P133), Orfeas AIDONOPOULOS, ATHENA (P133)
Compiled by:	Amaryllis RAOUZAIOU, ATHENA (P133)
Contributor(s):	Wouter KLIJN, JUELICH (P20), contributed to Sections 2 and 3 and Annex X and VI. Alan B STOKES, UMAN (P63), contributed to Annex IX. Daniel KELLER, JUELICH (P20), contributed to Annex IX. Daviti GOGSHELIDZE, JUELICH (P20), contributed to Annex IX. Dennis TERHORST, JUELICH (P20), contributed to Annex IX. Eric MULLER, UHEI (P47), contributed to Annex IX. James Gonzalo KING, EPFL (P1), contributed to Annex IX and Annex X. Sandra DIAZ, JUELICH (P20), contributed to Annex IX. Thorsten HATER, JUELICH (P20), contributed to Annex IX. Marmaduke WOODMAN, AMU (P78), contributed to Annex X. Oliver BREITWIESER, UHEI (P47), contributed to Annex X. Petra RITTER, CHARITE (P122), contributed to Annex X.
WP QC Review:	Evita MAILLI, ATHENA (P133)
WP Leader / Deputy Leader Sign Off:	Yannis IOANNIDIS, ATHENA (P133)
T7.4 QC Review:	Tech: Steven VERMEULEN, EBRAINS (P1). Edit: Guy WILLIS, EPFL (P134)
Description in GA:	Report presenting (a) the EBRAINS Infrastructure user requirements, its detailed logical architecture, deployment planning, integration approach and guidelines, (b) the delivery processes, quality assurance procedures, mechanisms and metrics/KPIs, software and software production management tools and dependency/licensing policies. The report will be continuously updated throughout the duration of the project to reflect the evolution and alignment of the EBRAINS infrastructure.
Abstract:	Deliverable **D5.3** lays the foundations for the **design**, **integration** and **delivery** of the EBRAINS infrastructure, presenting the **requirements elicitation process**, **conceptual architectural diagrams**, **guidelines** for the components' integration, as well as establishing **rules** for **software quality** in SGA3. D5.3 is the first deliverable necessary for the fulfilment of Workpackage Objective 5.3 (***WPO5.3 EBRAINS software components integration, testing and delivery***), leading us to the first milestone of WP5 (*MS5.1 PoC EBRAINS infrastructure*).
Keywords:	Architecture, guidelines, requirements, software quality, software delivery, infrastructure, integration, services, use cases, user panels, conceptual architecture, physical deployment, components, API, KPIs
Target Users/Readers:	computational neuroscience community, computer scientists, consortium members, HPC community, platform users, neuroinformaticians, neuroimaging community, neuroscientific community, neuroscientists, researchers, scientific community

Table of Contents

Table of Tables

Table of Figures

Executive Summary

Deliverable D5.3 lays the foundations for the **design**, **integration** and **delivery** of the EBRAINS infrastructure; presenting the **requirements elicitation** process, **conceptual architectural diagrams** and **guidelines** for the components' integration, as well as establishing rules for **software quality** in SGA3. D5.3 is the first Deliverable needed for the fulfilment of Work Package Objective 5.3 (WPO5.3 EBRAINS software components integration, testing and delivery), leading us to the first Milestone of WP5 (MS5.1 PoC EBRAINS infrastructure).

EBRAINS is a distributed digital infrastructure 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. The **final EBRAINS infrastructure** (D5.7 EBRAINS Infrastructure), produced during SGA3 and delivered at the end of the Project, will include all the components and services developed in HBP.

D5.3 is (SGA3 description in bold) a report presenting**:**

- The **EBRAINS Infrastructure user requirements**: the methods for the elicitation of the requirements in accordance with ISO/IEC/IEEE 29148:2018 [1], the use cases used during this process and a first set of requirements for the technical part of EBRAINS RI are available in Section 2, Annex VI and Annex VIII respectively;
- Its **detailed logical architecture**:- Section 3.3 describes the Logical Architecture with diagrams
- The **deployment planning:**- the Physical Deployment with diagrams of present and projected states is presented in Section 3.4;
- Its **integration approach and guidelines:** "How to Integrate Components" (with a diagram for each deployment type) is the topic discussed in Section 3.8.2;
- **The delivery processes**: Section 4.1.5 describes all the Developer Services and Tools including project management services, how they connect with continuous integration and how it provides delivery pipelines, and deployment environments;
- The **quality assurance procedure**: like the formally scheduled "TC Weeklies" and "TC Task Force" meetings described in Annex I, and regular ad-hoc "debrief" meetings for synchronizing about KPI outputs;
- The **mechanisms**: Section 4.1.2 describes Collaboration Tools and Services, particularly Kanban boards provided by Gitlab and Wiki pages provided by Collaboratory;
- The **metrics/KPIs**: Section 4.3 includes a KPIs table;
- The **Software and software production management tools**: Section 3.8 presents the main Deployment Types on EBRAINS RI, the various IaaS and PaaS technologies from an architectural point of view along with the various tools available to EBRAINS Component Developers that assist in software development and software production management as well as day to day operations. Moreover, as mentioned above, Section 4.1.2 describes Collaboration Tools and Services; and
- **Dependency/licensing policies**: a first approach of which can be found at Section 3.8.3.

The report will be continuously updated throughout the duration of the project to reflect the evolution and alignment of the EBRAINS infrastructure. EBRAINS Phases of Integration are described in Section 3.1. Once officially delivered, the report will be available in the Technical Collaboration collab (Annex I: Technical Coordination).

Collaboration at the heart of SGA3 Technical Coordination

The content of the current Deliverable has been framed, influenced and co-developed through a series of **joint activities and collaborations** engaging the entire SGA3 Consortium in a concerted

manner, via all established Technical Coordination (TC) instruments and decision-making bodies (see Annex I: Technical Coordination). In summary:

Our work begun with the creation of a comprehensive, **EBRAINS RI components catalogue**, with the initial input of all SGA3 Partners delivered during our 3-day TC Kick-off meeting (M2). The catalogue has been created as a **stable and constantly updated** knowledge base for the collection, disambiguation, and dissemination of technical knowledge across the Consortium. The original input from partners was assessed and cross-referenced across all available distributed sources of current and historical information (i.e. SGA3, Collaboratory 1 and 2, HLST, PLUS) under a **critical perspective.** A comprehensive reporting template was prepared, and information was transferred into the joint **TC Collab,** while partners were instructed to update/complete their respective contributions and ensure the provided information remains updated during the **entire duration** of the project. This catalogue is the foundation for **sharing** and **monitoring** up to date information per EBRAINS component, promoting **synergies** and reuse with other activities (e.g. front-facing public roadmap, HLST), while also guiding the planning of the different Phases of development, integration and delivery activities. A concise version of this catalogue is available in **Annex IV** of the current deliverable.

Further, three working groups (WGs), i.e. thematically driven teams comprising members of the SGA3 Consortium providing **directions**, **discussion points**, and **critique** on a series of cross-RI areas, have been established:

- EBRAINS Architecture Infrastructure WG, 'EAI-WG',
- EBRAINS Software Quality WG, 'ESQ-WG' and
- EBRAINS Software Delivery WG – 'ESD-WG')

These provided valuable feedback, that informed the guidelines of the respective sections of this deliverable. The outputs of these working groups are available in **Annexes V**, **IX,** and **X**, respectively. The WGs will remain active throughout SGA3, offering their continuous feedback and support, complemented by two additional WGs:

- EBRAINS Security WG – 'EIS-WG', and
- EBRAINS APIs (EBRAINS-APIs WG – 'EA-WG'.

In full alignment with the SGA3 provisions and following a series of discussions and consultations in the context of the TC-TF (Technical Coordination Task Force), working groups and higher-level decision-making bodies of SGA3, a **pragmatic high-level roadmap** towards the delivery of EBRAINS has been established, agreed up and enacted. This roadmap will be applied to deliver, monitor and convey the shared responsibility for the delivery of EBRAINS, while specialising and integrating all technical guidelines identified within this report.

The roadmap comprises **four phases**, spanning the course of SGA3, presented in Annex II: TC planning:

- Phase 1 (Duration: M4-M11; Output: MS5.1 Proof of Concept EBRAINS RI).
- Phase 2 (Duration: M12-M18; Output: MS5.2 Beta EBRAINS RI).
- Phase 3 (Duration: M19-M30; Output: RC EBRAINS RI).
- Phase 4 (Duration: M31-M36; Output: EBRAINS RI).

Currently, **Phase 1** is ongoing, with its early output already integrated in the current guidelines. Four components, **Arbor**, **NEST**, **SimVisSuite** and **The Virtual Brain (TVB),** have been selected in order to provide a manageable, yet representative, assembly of technical maturity, established development and testing processes, and ongoing deployment practices. This phase will allow us to streamline the integration practices for all EBRAINS components, identify potential issues and blocking points, test the guidelines presented in this report and, ultimately, inform and improve the EBRAINS technical guidelines to be applied and enforced across the board in Phase 2.

Furthermore, 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, in response to

the need to prioritise EBRAINS-wide technical aspects and issues of critical nature for the delivery of SCs and ICEI output. The prioritisation process, where different technical priorities are 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.

Report Structure

D5.3 includes, apart from introductory and concluding parts, **three main sections** and several **annexes**, assembling and presenting all relevant information for the design, integration, and delivery of EBRAINS.

Section 2

In Section 2, we present the **requirements elicitation process**. Several methods were combined in order to gain a more "all-round" and representative view of the EBRAINS complex requirements, with each subsection presenting a different part of this process, also explaining the different types of **stakeholders** of the platform. The extracted requirements concern the technical part of the infrastructure, its general functionalities, as well as approach for our development, integration, testing, and delivery processes. Depending on their nature, these requirements are presented 'as-is' (**Annex VIII: User Requirements**), portrayed and embedded in scientific use cases EBRAINS must support (**Annex VI: Use Cases)**, or fully integrated across all respective sections of this document regarding the overall architecture, development, integration, testing, and deployment of EBRAINS. Our approach is designed to be **iterative** and **continuous**, with flows of additional information and thus requirements stemming from HLST and SLU.

In this context, the user requirements of the individual **components** to be integrated in EBRAINS, are outside the scope of this document; they continue to be defined and implemented according to scientific priorities established by SGA3. However, the EBRAINS-level requirements and technical guidelines presented in this report must be respected and fully adhered to. In this collaborative process, the SGA3 TC assumes a proactive role in monitoring, consulting and accepting each component in terms of its conformance with the provided guidelines. Furthermore, the SGA3 TC ensures coordination, reuse of existing solutions, and overall an optimal utilisation of available human and computing resources.

The different subsections of Section 2 present the requirements engineering method, the levels of requirements specification and their elicitation process, including the benefits and drawbacks of every method. Section 2 also describes the key stakeholders of EBRAINS RI and presents the user panels created in the framework of SGA3. The users, through these panels, participate in both the elicitation of requirements and the evaluation of the system. Finally, the use cases that helped to extract the requirements, along with the methods used for their formalisation, are also presented.

Section 3

In Section 3, we present and elaborate the **EBRAINS Architecture**. EBRAINS is built on top of complex, heterogeneous and federated technical foundation, which features a very wide range of capabilities and offerings at many levels. 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/communication 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.

This section includes the high-level **Conceptual Architecture** of EBRAINS, meant to assist the understanding of EBRAINS by all 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. Further, we provide an overview of supported deployment options both for **current** and **future** EBRAINS component developers, aiming to educate and support them in the operationalisation of their services in EBRAINS. We place special emphasis 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 realization of the long-term vision for EBRAINS, as a constantly evolving marketplace of services and offerings for neuroscience and the brain.

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 Technical Coordination (TC) team responsible for detailing the infrastructure 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. The various views of the EBRAINS RI that are presented in this section of D5.3, will be continuously iterated, reviewed and updated to accurately reflect the current and projected status of the EBRAINS RI.

Section 4

In Section 4, we provide comprehensive directions for SGA3 developers, component owners and scientists. Its main goal is to provide technical information regarding the offerings of EBRAINS to each target group and present the requirements and recommendations for a component to be deployed and integrated in the EBRAINS RI.

A great variety of Developer and Runtime services are demonstrated and summarized in order to help developers understand the complex EBRAINS RI Architecture. Scientists, developers and component owners will benefit from the collaboration tools and services that EBRAINS offers and from the ability to use Knowledge Graph as a multi-modal metadata store for storing, fetching, depositing data and metadata. In order to create a streamlined and effortless procedure for integrating components to EBRAINS RI, guidelines for component integration to EBRAINS RI are presented. The included guidelines depict our current holistic directions and may be revised for case-by-case components. Further, we present our approach regarding the indicators to be used for quality control.

This section has been developed under the guidance, feedback, contribution, and review of the EBRAINS Software Quality (ESQ-WG) and EBRAINS Software Delivery (ESD-WG) Working Groups.

Annexes

Finally, several **annexes** are provided in this report, offering additional insights and details to the reader. **Annexes I** and **II** detail the approach of the SGA3 Technical Coordination team and its current progress; **Annexes V, IX** and **X,** present the outputs of the different Working Groups contributing to this report, **Annexes III** and **IV** present the current list of EBRAINS components and Web-APIs; **Annexes VI, VII** and **VIII** present the EBRAINS Use Cases, questionnaire used in requirements elicitation process, and the established set of requirements respectively.

NOTE:

This Deliverable is considered as a live document, co-developed and revised in the TC Collaboratory. This version is the latest snapshot of our complete body of work, reflecting our current vision and directions regarding the development, integration, and delivery of the EBRAINS infrastructure.

The Deliverable will be continuously updated throughout the duration of the Project to reflect the evolution and alignment of EBRAINS with the scientific, technical, and sustainability requirements of SGA3 and the EBRAINS AISBL.

1. Introduction

The "Final Proposal" submitted by the future Human Brain Project to the European Commission in October 2012 stated that: *"The goal of the Human Brain Project (HBP) is thus to build an integrated ICT infrastructure…"*[1]

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[2]) 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 D5.3 lays the foundations for the design, integration and delivery of the infrastructure, presenting the requirements elicitation process, conceptual architectural diagrams, guidelines for the components' integration, as well as establishing rules for software quality in SGA3. The approach, the procedures, the planning, the activities and the collaborations of TC that informed this document are presented in **Annex I: Technical Coordination.** D5.3 is the first Deliverable necessary for the fulfilment of WP Objective 5.3 (WPO5.3 EBRAINS software components integration, testing and delivery), leading us to the first Milestone of WP5 (MS5.1 PoC EBRAINS infrastructure).

The deliverable is considered as a **live** document, with this version reflecting our current vision and directions regarding the delivery of the EBRAINS RI. As such, the deliverable will be continuously updated throughout the duration of the project to reflect the evolution and alignment of the EBRAINS infrastructure with the scientific, technical, and sustainability requirements of SGA3 and the EBRAINS AISBL.

[1]Special FET Flagship Call, "The Human Brain Project", October 2012 – Proposal Abstract, p.1. The objective was stated as: "The goal of the Human Brain Project is to build a completely new ICT infrastructure for future neuroscience, future medicine and future computing that will catalyse a global collaborative effort to understand the human brain and its diseases and ultimately to emulate its computational capabilities." – Section 1.1.1, The objective, p.9.

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

1.1 Document Structure

This document includes, apart from introductory and concluding parts, **three main sections** and several **annexes**, assembling and presenting all relevant information for the design, integration, and delivery of EBRAINS.

1.1.1 Section 2

Section 2 presents the **requirements elicitation process**. Several methods were combined in order to gain a more "all-round" and representative view of the EBRAINS complex requirements, with each subsection presenting a different part of this process, also explaining the different types of **stakeholders** of the platform. The extracted requirements concern the technical part of the infrastructure, its general functionalities, as well as approach for our development, integration, testing, and delivery processes. Depending on their nature, these requirements are presented 'as-is' (**Annex VIII: User Requirements**), portrayed and embedded in scientific use cases EBRAINS must support (**Annex VI: Use Cases)**, or fully integrated across all respective sections of this document regarding the overall architecture, development, integration, testing, and deployment of EBRAINS. Our approach is designed to be **iterative** and **continuous**, with flows of additional information and thus requirements stemming from HLST and SLU.

In this context, the user requirements of the individual components to be integrated in EBRAINS, are outside the scope of this document, as they continue to be defined and implemented according to scientific priorities established by SGA3. However, the EBRAINS-level requirements and technical guidelines presented in this report must be respected and fully adhered to. In this collaborative process, the SGA3 TC assumes a proactive role in monitoring, consulting, and accepting each component in terms of its conformance with the guidelines put into place. Further, the SGA3 TC ensures coordination, reuse of existing solutions, and overall an optimal utilization of available human and computing resources.

The different subsections of Section 2 present the requirements engineering method, the levels of requirements specification and their elicitation process, including the benefits and drawbacks of every method. Section 2 also describes the key stakeholders of EBRAINS RI and presents the user panels created in the framework of SGA3. The users, through these panels, participate in both the elicitation of requirements and the evaluation of the system. Finally, the use cases used in the extraction of requirements, along with the methods used for their definition are also presented.

1.1.2 Section 3

Section 3 presents and elaborates the **EBRAINS Architecture**. EBRAINS is built on top of a complex, heterogeneous and federated technical foundation, which features a very wide range of capabilities and offerings at many levels. 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 back end 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/communication 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.

This section includes the high-level **Conceptual Architecture** of EBRAINS, meant to assist the understanding of EBRAINS by all 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 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.

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 Technical Coordination (TC) team responsible for detailing the infrastructure 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. The various views of the EBRAINS RI presented in this section of D5.3 will be continuously iterated, reviewed and updated to accurately reflect the current and projected status of the EBRAINS RI.

1.1.3 Section 4

Section 4 provides comprehensive directions for SGA3 developers, component owners and scientists. Its main goal is to provide technical information regarding the offerings of EBRAINS to each target group and present the requirements and recommendations for a component to be deployed and integrated in the EBRAINS RI.

A great variety of **Developer** and **Runtime services** are demonstrated and summarised in order to help developers understand the complex EBRAINS RI Architecture. Scientists, developers and component owners will benefit from the collaboration tool and services that EBRAINS offer and from the ability to use the Knowledge Graph as a multi-modal metadata store for storing, fetching, depositing data and metadata. In order to create a streamlined and effortless procedure for integrating components to EBRAINS RI, guidelines for component integration to EBRAINS RI are presented. The guidelines included depict our current holistic directions and may be revised for case-by-case components. We also present our approach regarding the indicators to be used for quality control.

This section has been developed under the guidance, feedback, contribution and review of the EBRAINS **Software Quality** (ESQ-WG) and EBRAINS **Software Delivery** (ESD-WG) Working Groups.

1.1.4 Annexes

Finally, several **annexes** are provided in this report, offering additional insights and details to the reader. **Annexes I** and **II** detail the approach of the SGA3 Technical Coordination team and its current progress; **Annexes V, IX** and **X** present the outputs of the different Working Groups contributing to this report, **Annexes III** and **IV** present the current list of EBRAINS components and Web-APIs; **Annexes VI, VII** and **VIII** present the EBRAINS Use Cases, questionnaire used in requirements elicitation process and the established set of requirements, respectively.

1.2 How to read this Deliverable - Relation to other Deliverables and documents

This Deliverable is considered as a live document co-developed, revised and available in the **TC Collaboratory**. This version is the latest snapshot of our complete body of work, reflecting our current vision and directions regarding the development, integration, and delivery of the EBRAINS infrastructure.

The current Deliverable is addressed to all the SGA3 Partners. There are sections related to management issues (e.g. Section 1) and others that contain information relevant to the Project's

scientists (e.g. Section 2), while the main part (Sections 3 and 4) is mainly addressed to the **developers**, including useful guidelines about the delivery of their work. This document, properly adapted, will also constitute the main guidelines for **external users** who want to contribute to the EBRAINS RI. 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 presentation and the purpose of D5.3 is to collect, evaluate and present it, in order to serve as a reference point for every reader. Having this in mind, all information referenced in this Deliverable, including documents embodying the output of the SGA3 Working Groups, are referenced or provided as Annexes for the convenience of the reader.

Besides the documents of the different Working Groups (Architecture Infrastructure, Software Quality and Software Delivery Working Groups), D5.3 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 D5.3 refers to **QC/KPIs on technical level** (component-level, up to EBRAINS). While certain D5.3 KPIs could also be reported in D7.1, their technical nature (development, testing, integration, operations) should be framed, presented and assessed on a technical level alone.

D5.3 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**) serve as an information source on possible new RI requirements related to the Showcases, while **D4.1**, **D4.2**, **D5.1** and **D5.2** 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)** (Section 4.1.2.2) in response to the need to prioritise EBRAINS-wide technical aspects and issues of critical nature for the delivery of the SCs' outputs.

D5.3 will be continuously updated throughout the duration of the Project to reflect the evolution and alignment of EBRAINS with the scientific, technical and sustainability requirements of SGA3 and the EBRAINS AISBL. The information included in this document, especially the sections targeting developers (e.g., technical guidelines), will be maintained and updated outside this report and in the TC Collab, as separate documents, shared with all SGA3 Partners.

Important updates will be disseminated through WP managers or the EBRAINS list. The executive summary has been structured in a way that facilitates the reader and provides an overview of our work.

D5.3 represents the first step towards **WPO5.3** (EBRAINS software components integration, testing and delivery). Two more deliverables are connected to this WPO: D5.4 - Interim EBRAINS Infrastructure Implementation – M18 and D5.7 - EBRAINS Infrastructure – M36.

1.2.1 EBRAINS RI: Terminology

Please note that in this text, and 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 wrapping EBRAINS public APIs • internal scientist/ internal developer/component owner = contributor of EBRAINS components • EBRAINS itself = as user of core services & Fenix resources 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.

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
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)
AISBL	Association International Sans But Lucrative (Legal Entity under Belgian Law)

2. EBRAINS RI Requirements

In this section, we present the requirements elicitation process. Several methods were combined in order to gain a more “all-round” and representative view of the EBRAINS complex requirements,

with each subsection presenting a different part of this process, also explaining the different types of platform stakeholders. The extracted requirements concern the technical part of the infrastructure, its general functionalities, as well as approach for our development, integration, testing, and delivery processes. Depending on their nature, these requirements are presented 'as-is' (Annex VIII: User Requirements), portrayed and embedded in scientific use cases EBRAINS must support (Annex VI: Use Cases), or fully integrated across all respective sections of this document regarding the overall architecture, development, integration, testing, and deployment of EBRAINS. Our approach is designed to be iterative and continuous, with flows of additional information and thus requirements stemming from HLST and SLU.

In this context, the user requirements of the individual components to be integrated in EBRAINS, are outside the scope of this document, as they continue to be defined and implemented according to scientific priorities established by SGA3. However, the EBRAINS-level requirements and technical guidelines presented in this report must be respected and fully adhered to. In this collaborative process, the SGA3 TC assumes a proactive role in monitoring, consulting, and accepting each component in terms of its conformance with the provided guidelines. Further, the SGA3 TC ensures coordination, reuse of existing solutions, and overall an optimal utilization of available human and computing resources.

Section 2 is outlined as follows:

- Section 2.1 presents the requirements engineering method and the levels of requirements specification.
- Section 2.2 presents the elicitation process, including the benefits and drawbacks of every method.
- Section 2.3 describes the key stakeholders of EBRAINS RI.
- Section 2.4 presents the user panels created in the framework of SGA3. The users, through these panels, participate in both the elicitation of requirements and the evaluation of the system.
- Finally, Section 2.5 presents the use cases used in the extraction of requirements and the methods used for their definition.

2.1 Requirements Engineering Method

Following ISO/IEC/IEEE 29148:2018 [1], the requirements engineering method of EBRAINS includes two main processes, the Stakeholder Requirements Definition Process and the Requirements Analysis Process. Both processes are executed in an iterative and recursive way throughout SGA3. During the first process, the defined requirements are based on the services needed by users and other stakeholders of the platform (output: Stakeholder Requirements Specification – StRS, Annex VIII: User Requirements), while during the latter, the stakeholder view of desired services are transformed into a technical view of the system that could deliver those services (output: System Requirements Specification – SyRS and Software Requirements Specification – SRS, Annex VIII: User Requirements).

The output of the requirements engineering method are requirements specification at three different levels, serving as input not only to different stages of the architectural design process, but also to architectural diagrams at different levels. Table 3 describes the different levels of Requirements [1] along with their expected role in the different levels of architectural design.

Due to the early stage of the SGA3 in which this deliverable is produced, the complexity of the envisioned architecture and the wide variety of technologies involved, this first iteration of requirements analysis focuses on use case (=stakeholders') requirements and software technology requirements separately.

The interaction between the different requirements levels and the transformation of the stakeholder, requirement-driven view of desired services into a technical view of a required product that could deliver those services are expected to be presented in the next iterations of this process. SLU will serve as the link between the initial architectural design (presented in Section 3) and the

feedback from the members of the user panels. EBRAINS business model will further contribute to contextualise this process.

Table 3: Levels of requirements specification

Level of requirements	Description	Architectural design
Stakeholder Requirements Specification (StRS)	During this process, the stakeholders of EBRAINS (Table 5), who are involved with the system throughout its lifecycle, are identified along with their needs and desires. These needs are further analysed to form a set of requirements that guide the intended interaction between EBRAINS RI and its operational environment. This set of requirements will provide validation guidelines and will be used as the reference against which the technical offerings and functionalities of the platform are validated.	EBRAINS RI capabilities
Systems Requirements Specification (SyRS)	This level defines the technical specifications for EBRAINS RI. It includes the future system requirements from the domain perspective, background information about the overall objectives for the system, possible constraints, and non-functional requirements, defining the characteristics that will satisfy stakeholder requirements.	EBRAINS RI services
Software Requirements Specification (SRS)	SRS defines the interfaces between EBRAINS RI and a software component and place external performance as well as functionality requirements upon it, including also the conditions under which the software component must perform, while it also provides means of verification for the requirements.	EBRAINS RI Technologies

The table describing every requirement of **Annex VIII: User Requirements** contains, along with its description, the following attributes:

- **Level**: the levels to be used are Stakeholder, System and Software (see Table 3).
- **Type**: FUNC (functional) and non-functional: USE (usability), PERF (performance), ENV (operational environment), SUP (maintainability and support), and SEC (security and privacy).
- **Priority**: MAN (mandatory), DES (desirable), OPT (optional), FUT (possible future enhancement).

2.2 Elicitation Process

The broad range of stakeholders to the EBRAINS infrastructure represent needs and challenges to be mitigated. EBRAINS promises an infrastructure with a very significant potential for tackling complex research and innovation challenges. Members of HBP and external users can use the EBRAINS infrastructure as contributors or as simple users. The range of stakeholders also includes Partnering Projects that have proven the value of wider collaboration on use and further development of the infrastructure.

One of the critical success factors in the development of any system is the deep understanding of its requirements. Once identified, the user requirements effectively lay the foundation for developers, testers, and implementers to begin implementing its required functionality, responsiveness, and interoperability.

Methods such as document analysis and analysis of users' interviews/surveys are being used for the elicitation of user requirements in combination with scenarios and Showcases. Different requirements analysis methods are being applied in parallel to complement each other, in order to yield more effective results.

The identification of needs and requirements of different groups is based on the combination of information from the **Showcases** defined in SGA3 (see Section 2.5.5), the **use cases** created during

the project (see Section 2.5), the **user panels** we established (see Section 2.4) and the preliminary analysis in preparation of SGA3.

The development of EBRAINS RI uses different methods in a complementary way, always considering the vision and scope of EBRAINS. The benefits and the drawbacks of each method are presented in Table 4.

Table 4: Requirements Elicitation Process

Method	Description	Benefits	Drawbacks
Showcases of SGA3 - Use cases	Detailed realistic examples of how users can use EBRAINS in a specified context.	• User requirements brought to life. • Vision clearly explained.	• Oversimplification of the needs. • Fragmentary approach.
User Panels (questionnaires, interviews etc.)	Sample population of users. Surveys, questionnaires, and interviews are used to determine needs, current work practices and attitudes to the new system ideas.	• Relatively quick methods of determining preferences of large user groups. • Allows statistical analysis. • Allows quick elicitation of ideas & concepts.	• Difficult to capture in depth comments. • May not permit follow-up. • Recruitment effort to assemble groups. • The group may not have the right questions to lead to "innovation" instead of "incremental improvement".

2.3 Stakeholder categories

The following key stakeholders are the main categories of actors involved with the EBRAINS RI: End-users, Data generators and providers, Technology and service providers. These categories are described in the following table.

Table 5: Stakeholders

Category	EBRAINS Side	Description
End-users	Demand side	Scientists of different fields or developers who will use the EBRAINS RI, and the tools and services included, in order to support and progress their research or create their own project with the advanced functionalities of EBRAINS. This category will be further segmented, as it is expected that different types of end-users will lead to different requirements.
Data generators and providers	Supply side	The scientists who provide the platform with the necessary knowledge, derived from their expertise. Currently and until the official release of EBRAINS RI, the scientists are members of the EBRAINS project, but, in the future, researchers and institutions external to EBRAINS are also expected to be data providers, contributing to EBRAINS and taking advantage of its central position in the research community.
Technology and service providers	Supply side	The developers of EBRAINS who create the different tools and services that are part of EBRAINS platform. In the future, external developers are also expected to contribute, developing external applications wrapping EBRAINS public APIs.

We should also mention that EBRAINS itself could also be considered as stakeholder, as a user of core services and Fenix resources.

2.4 User Panels

Elicitation and analysis of the requirements have been performed considering the needs and concerns coming from the different groups of EBRAINS users.

Three user panels (**Scientits-Internal**, **Developers-Internal**, **External Users**) – in alignment to the Stakeholders categories of Table 5 - are currently under formation, while a fourth one (**User Acceptance Panel**) will be formed around M18 of the SGA3; all of them are described in the following sections. Two of the current user panels involve users who are members of HBP, having an important role in the definition of EBRAINS specifications: the first one consists of scientists, while the second one includes developers. The third user panel brings together possible future EBRAINS end-users from different fields and with different levels of expertise.

Participants of the three above-mentioned user panels will be involved in the fourth user panel that will evaluate EBRAINS progress regarding the defined requirements and will continue to meet until the end of the project.

2.4.1 Scientists - Internal

The first of the internal user panels involves the **scientist members** of the HBP. The creation of this group involves both SLU (Scientific Liaison Unit) and TC. The impact of the delayed forming of SLU is not considered as substantial at this stage of the project, since the formation of the panel would be planned for M12, even if SLU was available, with Phase 1 (see Section 3.1) being "introverted" in nature (SGA3/TC bootstrap, internal use cases, Showcases). The technical offerings/functionalities are communicated to the scientists of HBP, aiming at seeing the technical details from a scientific, more high-level point of view. Questionnaires, meetings and discussions with the members of this group are some of the methods to be used to fill the gap between the different aspects of the RI: the technical and the scientific one. "Scientists – Internal" includes the scientists of HBP and it will be active until the creation of the User Acceptance Panel (see Section 2.4.4).

2.4.2 Developers - Internal

The second internal user panel involves the **developer members** of the HBP. Developers who are participating in the development of the different components form this group, in order to communicate the difficulties they are confronting not only during the development and integration of the component, but also during the close collaboration with TC for the different phases of components' integration. Questionnaires, meetings, discussions, but also a special session during Codejam #11 [2] about developers' onboarding were some of the methods we used to get familiar with the developer's point of view, improve the different procedures and get informed about their requirements. This panel will be active until the creation of the User Acceptance Panel (see section 2.4.4). One of the questionnaires used can be found at Annex VII: Developer's Questionnaire, while the document that constitutes the outcome of Developers' Onboarding Session is presented in Section 4.1.1 and can be found at [5].

2.4.3 External Users

The third panel involves different types of end-users who are **not members of the HBP**, but wish to **use** the EBRAINS offerings for their research or for **developing** their own applications and tools. EBRAINS aspires to become a research infrastructure that will address the needs of different communities, based on the tools and services it offers, but also on the technologies used, available later to all the interested end-users.

The external panel will comprise scientists and developers from (a) the neuroscience community, (b) neighbouring fields, (c) completely external fields (e.g., AI/ML, Data Scientists). It will be voluntary, with pan-European representation, and periodic user meetups to discuss and assess end-user perceptions/use cases/insights/ideas. Due to the expansive nature of the panel, the recruiting process (open calls, info days, comms) will be performed with the collaboration of SLU (neuroscience) and AISBL (neighbouring/external domains), leveraging their existing networks. The members of the external User Panel will also take part in the User Acceptance Panel.

2.4.4 User Acceptance Panel

After M18, a User Acceptance Panel will be formed, with members from all the above-mentioned panels. Scientists and developers who participate in HBP, along with external stakeholders, will take part in the activities of this panel, validating EBRAINS RI in a real setting. The aim will not be limited to a simple check of the defined requirements, but rather ensure that the EBRAINS RI satisfies the needs of the different categories of stakeholders. Evaluation will be performed on a frequent, ongoing basis to ensure that the RI remains in line with its scope. With the support of the User Acceptance panel, any necessary adjustments will be made while EBRAINS RI is still in development, since such a panel can capture new user requirements in a direct way, can identify issues that the testing might has missed, and provide an overview of the RI progress.

2.5 Use Cases

A use case can be a specific scientific or technical challenge as encountered in the HBP or EBRAINS and describes the current or expected ICT needs. The use cases are created in collaboration between domain scientists and technical coordinators. Different use cases elicitation processes have been applied, to ensure that the EBRAINS Architecture addresses the requirements of all different stakeholders. A first set of use cases has been already created, while more use cases are expected to be created throughout the Project.

The process consists of:

1) Co-design of use cases/research project proposals/engineering component descriptions.
2) Creation of an infrastructure architecture from bottom-up requirements.
3) Complementing the infrastructure architecture with top-down infrastructure requirements.
4) High-level validation by infrastructure coordinators, science Work Package and Service Category stakeholders.
5) Ongoing reviews and updates by the architecture working group consisting of science and infrastructure experts.

The description of the already defined use-cases can be found in the **Annex VI: Use Cases**. More information is available to HBP members in the related wiki page in EBRAINS Collaboratory [17].

More details about the whole process and its results can be found at the EAI-WG output (Annex V: High-level EBRAINS architecture).

2.5.1 Use case co-design

The information that guided the infrastructure architecture in the initial phase was the breakdown of HPC neuroscience use-cases as detailed in ICEI D3.6 ‘Scientific Use Case Requirements Documentation’[16]. In the co-design approach of this initial phase, 16 neuroscience cases were formalized. Each use-case completed a template document breaking down the science case into a science description, an overarching architecture diagram, detailed requirements of the components in this diagram and a first assessment of ICEI service or HPC requirements. This work was focused on scientific needs first with no overarching design limiting the scope of the use cases.

2.5.2 Bottom-up requirements

The second major input in the architecture was a semi-structured text analysis of the SGA3 science Work Package descriptions, as available in March 2019. This analysis entailed extraction of significant science, software and hardware keywords. These keywords were grouped based on level of abstraction, input/output relationship and science versus technology. The groupings were given a placeholder name, and major relationships were marked. The semi-structured text analysis resulted in three diagrams for each science Work Package.

In the second phase, tools, components and workflow fragments implementing the different components in the SGA3 WP diagrams were taken from the ICEI Deliverable D3.6 [16]. In a new diagram, these tools were placed in relationship to each other. At this stage, front-ends and back-ends were separated into two distinct layers. Additionally, a preliminary mapping to ICEI services was performed. Finally, the three SGA3 WP diagrams were merged into a single diagram: an infrastructure architecture from bottom-up requirements.

2.5.3 Top-down requirements

Up until this point, the architecture was only informed by science and infrastructure offerings, excluding components outside the domain/scope of these stakeholders. Shifting the focus from specific scientific workflows to a large-scale infrastructure added additional requirements. In this third phase of the architecture design, such components (e.g. operations, resource management, administrative tools) were included in the architecture. Additionally, we further abstracted the components into higher level components, avoiding where applicable HBP-specific tool names (e.g. 'Collaboratory' becomes 'Collaborative work environment').

2.5.4 Feedback from stakeholders

In the fourth stage, the architecture was presented to Architecture Working Group (EAI-WG) and TC weeklies. An explanation was given regarding the goal of the current activity and explicit feedback to the architecture diagrams was requested. More specifically, instructions were given to elucidate missing components and functionality. Only small problems were identified, resulting in the addition of two components: Neural Activity Resource and Machine Learning & AI.

2.5.5 SGA3 Showcases

The HBP science-driven SGA3 Showcases (Section 1.5.2.5 HBP Scientific Showcases in the SGA3 Grant Agreement) demonstrate the EBRAINS infrastructure role in addressing scientific challenges. EBRAINS RI requirements will be informed by the showcase breakdown into technical requirements and specifications. Workflows described in each of the five Showcases highlight the software components, tools and processes needed for their materialisation, and relevant use cases are derived to guide the integration of these components to the EBRAINS RI. The process of breaking down each Showcase is ongoing with the help of the Technical Engineering and Scientific integration Tasks (T1.11, T2.11, T3.10) and the SLU, with relevant RI requirements continuously added in future versions of this Deliverable.

3. EBRAINS RI Architecture

In this section, we present and elaborate the **EBRAINS Architecture**. EBRAINS is built on top of a complex, heterogeneous and federated technical foundation, which features a very wide range of capabilities and offerings at many levels. 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/communication 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 (primarily vertically-stacked) priorities of each component team, and the need to address continuously evolving requirements and adhere to large-scale infrastructure best practices, we have opted for loosely-coupled integration 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 the current section. The architecture design, especially for the physical deployment diagrams, was performed with the aim to achieve demand-aware scaling capabilities, cost-effective deployments and sustainable operation.

Considering the complexity of the RI, as well as the various involved stakeholders, different architectural diagrams needed to be created featuring appropriate levels of abstraction for distinct purposes and audiences. Focus was helped by not only accurately picturing the different conceptual levels of the RI, but also in facilitating external service developers that want to integrate in the EBRAINS RI to grasp an extensive overview of the platform level services and offerings.

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

This section is outlined as follows:

- Section 3.1 presents the development/implementation phases that have been established to keep track of the evolution and progress during SGA3 and will also shape the iteration cycles for the mapping of the infrastructure architecture.
- Section 3.2 includes the high-level **Conceptual Architecture** of EBRAINS, meant to assist the understanding of EBRAINS by all stakeholders.
- Section 3.3 presents the **Logical Architecture** of the EBRAINS RI at two levels of abstraction and detail.
- Section 3.4 includes the current and planned **Physical Deployment** of EBRAINS over the available computing, storage, and networking resources.
- Section 3.5 illustrates an overview of a particular Service Category (SC) and serves as an example of the types of overviews that will be documented in due course for the other SCs as well.
- Section 3.6 elaborates on EBRAINS APIs.
- Section 3.7 focuses on the concept of EBRAINS components that ultimately form the majority of the EBRAINS RI.
- Finally, Section 3.8 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.

3.1 EBRAINS Phases

Design, development and implementation activities during SGA3 that will ultimately lead to the official release of EBRAINS will be tracked against specific time intervals that have been established by EBRAINS TC (see **Annex II: TC planning**) 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 will be 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:

3.1.1 Phase 1 (Duration: M4-M11; Output: MS5.1 Proof of Concept EBRAINS RI)

EBRAINS Phase 1 aims to lay the foundation for the intense integration efforts that are going to take place through the course of SGA3 and will lead to the delivery of EBRAINS. The aim is 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 represent the most important architectural and operational decisions established for the EBRAINS RI thus far. 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:

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

Setup facility for EBRAINS CI runs (EBRAINS Gitlab platform) so that EBRAINS Phase 1 components have their official code repositories mirrored in the EBRAINS Gitlab and the EBRAINS CI runs for building container images, integration tests and performance tests on HPC have a central initialisation point. 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. The latest implementation status is available in the TC Kanban board at cards #201 (closed, CI/CD EBRANS Phase 1), #239 (closed, mirror code repositories for EBRAINS CI runs).

3.1.1.2 Container Orchestration with OpenShift

The idea was to prioritise containerised deployment instead of VMs where applicable and apply this rule to the components participating in Phase 1. Containerised (in OpenShift) 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 OpenShift, 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). Currently, EBRAINS is benefited from a second production OpenShift cluster that was recently installed, that allows for more workloads and provides load-balancing, and fail-over capabilities to EBRAINS applications running on top of it like the Collaboratory notebooks. The latest implementation status is available in the TC Kanban board at card #212 (closed).

3.1.1.3 Multi-scale, multi-site workflows

A small set of 2-5 of workflows implemented and working across two of the five Fenix RI sites. Any technical interventions required to deliver them will be addressed formally in the next phase. We are currently identifying the broad term “workflows”, since within EBRAINS it is used in a lot of different ways (computational workflow vs scientific workflow, workflow recipe vs workflow execution). At this stage, we are also implementing a first exercise where the main goal is to provide ready to use templates so users can retrieve a dataset, or a model stored in the Archival storage, by getting the exact location from the EBRAINS Knowledge Graph. After this step is completed, a simulation could run in an HPC system and the output will be stored back in the Archival storage, after informing Knowledge Graph with the important metadata of where the output is located, so it can be later retrieved. The latest implementation status is available in the TC Kanban board at card #218 (PoC EBRAINS workflow, parent card of six other tasks).

3.1.1.4 Monitoring & Observability

Work is ongoing towards establishing all the available and necessary component-, platform- and infrastructure-specific sources of information, assess completeness of the collected information, and prototype implementations for automated sharing (push/pull) of information to a centrally managed monitoring service. Currently at a PoC level, an ELK stack [41] has been installed and is available 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 appropriate dashboards. The evolution of the PoC to a prototype implementation will continue in the next phases.

3.1.1.5 Collaboratory/Notebooks

Work is ongoing to provide 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). 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) is actively pursued. The latest implementation status is available in the TC Kanban board at cards #185 (closed, make available a Docker image registry and for the Collaboratory images), #243 (update the base Docker Collaboratory 2 image used by Jupyter Notebooks), #200 (enable modern software packages to be installed in the Collaboratory spawned containers), #234 (choose the execution site for Collaboratory 2 Jupyter notebooks), #178 (CI for headless browser e2e testing of Collaboratory notebooks).

3.1.1.6 Provenance

Identify all sources for information that need 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). Prototype

assembling the available metadata under one record (with PID (persistent identifier)) and providing a simple API. Limited interventions in EBRAINS components, where needed, establish a roadmap for formal PROV-O metadata/service in Phase 2. At this stage, a provenance metadata schema that will be stored inside the EBRAINS Knowledge Graph will be created from information provided by different use cases. With the implementation of an API, an automated way of gathering this kind of metadata will be available in a later step. Also, an automated way of storing captured provenance metadata inside the Knowledge Graph will be in place for EBRAINS components to follow. The latest implementation status is available in the TC Kanban board at card #225 (a parent card of 8 other cards relating to the aforementioned activities, i.e. API implementation, metadata storage, capturing provenance in workflows, and Jupyter notebooks, creation of metadata schemas, UI for searching for provenance information).

3.1.1.7 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 next phases. Currently, and directly relating to the PoC for Container Orchestration with OpenShift a secondary production OpenShift 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 (Section 0) 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 missing items and new requirements that are sure to facilitate both the TC's and the CO's activities in the subsequent phases, working towards the sustainable delivery of EBRAINS.

Before we present the next phases, it is worth mentioning that the exact scope of the subsequent phases will be established after Phase 1, depending on progress, insights generated and results. The following descriptions are only indicators and include a limited selection of high-level actions.

3.1.2 Phase 2 (Duration: M12-M18; Output: MS5.2 Beta EBRAINS RI)

In Phase 2, the aim is to integrate nearly 50% of current components to the EBRAINS RI and, at the same time, ensure integration practices' full conformance with Deliverable "D5.3 EBRAINS Technical Coordination Guidelines". A key action item will be the expansion of the EBRAINS stack of services, currently provided by two ICEI/Fenix facilities (CSCS and JSC - see Section 3.5), to the remaining three ICEI/Fenix facilities, to enable multi-site deployments and identify site-specific considerations according to the EBRAINS Architecture. Intense work will be performed to formalise the EBRAINS APIs (*de facto*, documentation, new) and establish API management as well as gradually integrate middleware components across the RI. It is intended that EBRAINS RI-wide Monitoring, Accounting and Observability services, will be in operation.

3.1.3 Phase 3 (Duration: M19-M30; Output: RC EBRAINS RI)

In this phase, the aim is to integrate 100% of current 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 need to be elaborated, bootstrapped and tested in this phase, to prepare for the final phase and last refinements.

3.1.4 Phase 4 (Duration: M31-M36; Output: EBRAINS RI)

In Phase 4, everything should be ready for the delivery of the envisaged EBRAINS RI and the handover of its operations to EBRAINS AISBL.

3.2 Conceptual Architecture

In this section, the Conceptual Architecture of EBRAINS is presented. The work is the direct output of the EAI-WG, which monitors the architecture infrastructure mapping efforts of EBRAINS (full document available in Annex V: High-level EBRAINS architecture).

This high-level view of EBRAINS architecture is based on the architecture section of the SGA3. 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 3.2.1 provides an outline of the architecture at the highest level. In Section 3.2.2, the design principles are introduced. Finally, Section 3.2.3 provides the next level of detail for the components of the EBRAINS infrastructure. It is worth mentioning that Section 3.2.3 is quite extensive and describes in detail all boxes of the architectural diagram presented in Figure 2.

3.2.1 Conceptual architecture outline

The EBRAINS Architecture comprises of the four layers depicted in Figure 2. The architecture is completed with the operations teams (on the right), along with administration and democratisation components exposing the infrastructure 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).
- Service Offerings. 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).
- Infrastructure Middleware. This contains all components responsible for managing, allocating, and monitoring the use of all underlying computing resources provided by “Hardware & IaaS/HPC” layer, affording flexibility, scalability, efficient use of resources, fault-tolerance and dynamic demand-aware provision of services to users.
- Hardware & IaaS/HPC. This contains: (a) the external 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 grand basis via EuroHPC, Prace or potentially other 3rd parties; as well as (c) the Neuromorphic Hardware provided by the HBP Facility Hubs, SpiNNaker and BrainsScaleS.

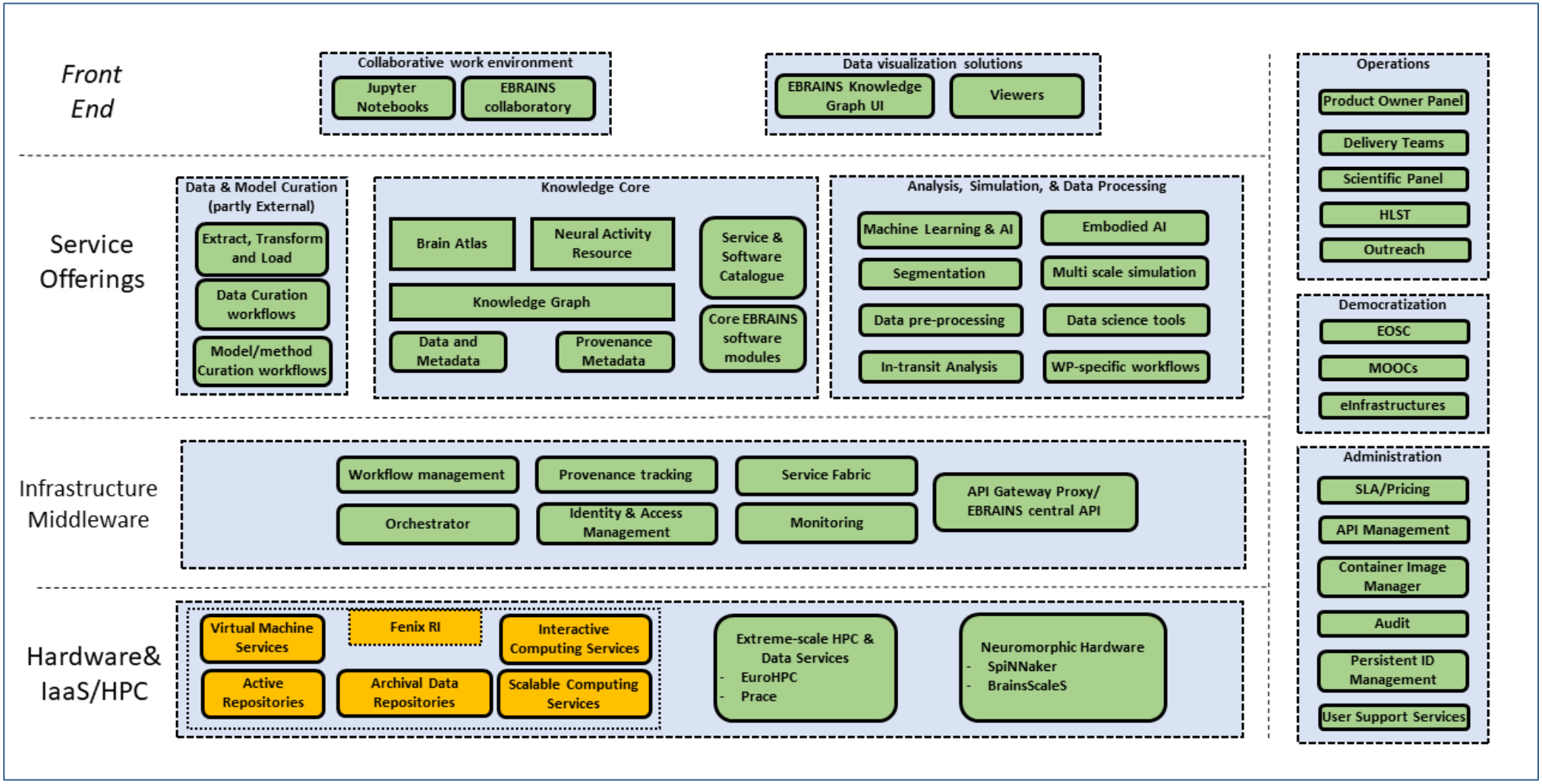

Figure 2: EBRAINS High-Level Architecture.

A clear layer structure has become apparent. The lowest level supplies the resources for the Infrastructure Services. The layer above this constitutes the Resource Management layer, exposing the wide range of lower-level services in a unified manner. At the Front End, there is a collection of user-facing components. The Science and Service Categories are embedded in between these two layers and constitute the science functionality as offered by the HBP. The architecture is completed with, on the right side, the operations teams along with administration and democratisation components exposing this infrastructure to the world.

3.2.2 Architecture design principles

The EBRAINS architecture abstracts and collates the internal operational and computing requirements of each Service Offering 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 on an EBRAINS infrastructure. Each service offering is to be instantiated in the proposed infrastructure 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 infrastructure, 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 & scaling.** The physical deployment and operation of EBRAINS will be supported by the IaaS cloud and HPC services of Fenix. 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 automatically 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.

Detailed explanations of the EBRAINS components illustrated in **Figure 2** 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.

3.2.3 Architecture components

3.2.3.1 Front End

A non-exhaustive set of end-user components, primarily at the user-facing end of the EBRAINS infrastructure, serves to introduce potentially new or adapted end-user interfaces with relatively small effort during the infrastructure'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 the evolving user needs and

pertinent scientific output by repurposing existing components and service end-points, 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 service offerings. Furthermore, they support all types of users (from new, to advanced), allowing a user to increase their knowhow and sophistication/complexity of EBRAINS interactions, within the same UI, thus adapting to their needs. Currently, the group comprises the following:

- The **EBRAINS Collaboratory** is 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 serves as first point of contact with EBRAINS for groups and communities of new 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.
- **EBRAINS 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).
- **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 generic-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.
- **Viewers**: A collection of specialised comprehensive, stand-alone, web-based, domain-specific/specialised 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.

3.2.3.2 Service Offerings

This is the second layer of the EBRAINS infrastructure (illustrated in Figure 2), closely linked to and powered by the third layer (Infrastructure 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 Service Fabric, 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 an series of *ad hoc* components to instantiate open-ended scientific research. For example, the Curation Workflows directly provide the EBRAINS Knowledge Graph with the output of data curation workflows, while the EBRAINS Knowledge Graph provides links to its data assets (stored within ICEI/Fenix) via the Service Fabric to the Data pre-processing component.

3.2.3.2.1 Curation Workflows

These comprise a collection of internal and external components and workflows that implement EBRAINS curation workflows for data assets, methods and models. Regardless on 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.

- Extract Transform and Load. This includes an extensible suite of standard extract-transform-load (ETL) processing operations, applied in multiple types and sources of data assets. Operations can be assembled in 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-process data to prepare the input for a given workflow).
- Curation: This comprises an extensible collection of formalised data and model/method curation workflows, with their final output being data assets forward to the Knowledge Core for indexing, management, linking, and provision.

3.2.3.2.2 Knowledge Core

The Knowledge Core 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 all other EBRAINS components.

- EBRAINS 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 ICEI/Fenix ('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: 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 at which computation run, started by agent or person, hardware system, configuration files. Different use cases within EBRAINS may need further information to be captured, in order for the overall process to be reproduced (Reproducibility) – note that the set of described functionalities is currently not fully implemented.
- Brain Atlas: Brain atlases 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 organises data in 2D maps of the surface of the Earth.
- Neural Activity Resource: Tools and services to provide in-depth (Tier 3) curated metadata for activity data, making the data interpretable and discoverable through more advanced EBRAINS Knowledge Graph searches.

3.2.3.2.3 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 standardized. 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. Listed here a number of non-exhaustive 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.
- Segmentation: Component for post-processing and segmentation of data stored in the Brain Atlas and other Knowledge Core 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 interactive 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.
- WP-specific 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.
- Data science tools. Generic tools needed to perform data analysis (e.g. Python, R, TensorFlow).

3.2.3.3 Infrastructure Middleware

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.

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 intra-component communication, invocation and provenance tracking. In a typical large-scale system, such operations (message queues/micro-services) are handled by a service bus and 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).

Please note that the term 'workflow' is used in different parts of the proposal to refer to dataflows, workloads or job orchestration (depending on the respective context). In the case of the Workflow Manager (WM), we are referring to the software responsible for invoking, scheduling, monitoring, and allocating the corresponding computing resources for the execution of analysis-processing-simulation pipelines involving one or more EBRAINS components. For example, the WM is responsible for instantiating the efficient execution of an analysis algorithm according to its context (e.g. access rights, quotas). In another example, the WM is responsible for executing a DAG of analysis operations defined as part of a broader (e.g. multi-scale) analysis pipeline.

The Infrastructure Middleware comprises the following 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.
- Service Fabric. This is responsible for the efficient communication and message exchange of inner-EBRAINS services. As such, it allows the disciplined, traceable and secure service invocation, data exchange, event handling, and infrastructure-wide monitoring of its operation. The Service Fabric implements (in standard Service-Oriented Architecture terms) an "Enterprise Service Bus" (ESB) for the EBRAINS infrastructure, facilitating the integration of its various components, and inherently addressing their *de facto* heterogeneity in implementation, deployment and operation details.
- API catalogue. All software artefacts, service offerings, internal services, workflows, and resources are uniquely identified, versioned, catalogued and presented to internal components, services, users and third-party systems (as appropriate via the "APIs" component of the "Front-Ends" layer) via the API catalogue. The catalogue is provided as a metadata-source for searching, browsing and programmatically discovering all APIs in the infrastructure, including component-provided APIs and the EBRAINS central API (see below). 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. An option that could be examined for the hosting of the API catalogue is that of the EBRAINS Knowledge Graph.
- API gateway & web reverse-proxy. This provides a single point-of-entry (and therefore singular point of administration & maintenance) for any APIs which are REST-on-HTTP conformant enough to able to be gateway-proxied, and for any web/HTTP resources which are appropriate for basic proxying. These are both provided by the same service, 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.

- EBRAINS central API. This 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.
- Workflow management: This component 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.
- Orchestrator. Development and deployment of a central Orchestrator that prioritises and monitors the execution of VMs, containers and jobs, enforcing policies for maximising the efficiency in using the available computing resources, and in accordance to the required SLA per given user/service offering.
- Provenance tracking: This is a centralised service 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.
- Monitoring: A unified service collecting, storing, sharing, and selectively informing in real time (e.g. to end-users) monitoring information from integrated EBRAINS components via the Service Fabric. In addition, it is responsible for the management of an internal unique identifier scheme for all data assets, services, workflows and resources, with linking to external global persistent identifier schemes for data assets and services.

3.2.3.4 Hardware & IaaS/HPC

EBRAINS relies on this final and lowest layer to provide compute and storage resources. It comprises of (a) the external 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 grand basis via EuroHPC, Prace or potentially other 3rd parties, as well as (c) the Neuromorphic Hardware provided by the HBP Facility Hubs, SpiNNaker and BrainsScaleS. The availability of the planned full set of five Fenix RI sites can is publicly available[3].

- 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.

[3] https://fenix-ri.eu/infrastructure/resources/planned-resources

- 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, Monitoring Services.
- Extreme-scale HPC & Data Services: Tier 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 BrainsScaleS.

3.2.3.5 Other facilities

3.2.3.5.1 Administration

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

- SLA (Service Level Agreement) / 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 owners.
- API Management: Provides detailed information regarding API use/rights according to established licenses/privileges.
- Container Image Manager: handles the lifecycle and provision of production-ready containers (for containerised services, models, algorithms, clients) to be deployed on top of the available IaaS resources, thus allowing the flexible and dynamic scaling and re-configuration of EBRAINS.
- 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.
- Repos/Issue tracking. A source code repository and issue tracking system for all software expected to be developed/extended/reused/deployed by EBRAINS. Aims to reduce development and maintenance effort, increase code quality, and promote intra-scientific collaboration at the software development level.
- User Support Services. Back end for the management of support tickets from EBRAINS users.

3.2.3.5.2 Operations

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.

- 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 Service Team. Decisions are taken by the EBRAINS Product Owner and implemented by the SCRUM Masters (if available) and Delivery Teams.

- Delivery Teams: Collection of all Delivery Teams developing the EBRAINS infrastructure and its components.
- 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).
- High Level Support Team (HLST), including Helpdesk: The HLST team 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.
- Outreach Team: Responsible for inviting users/organisations to EBRAINS, disseminating its service offerings, exploring new synergies/collaborations, and pursuing the established business model/revenue streams.

3.2.3.5.3 Democratisation

External key policy and educational activities pursuing the democratisation of the EBRAINS output across the scientific community and society.

- EOSC: (European Open Science Cloud). Ensures conformance to Open Access and FAIR data policy guidelines, streamlining the exposure and inclusion of EBRAINS-produced scientific output.
- MOOCs (Massive Open Online Courses): Provides support to e-learning curricula and training courses to educate, train and inclusively engage young scientists in EBRAINS offerings and output.
- e-infrastructures: Open-ended collection of aligned/complementary e-Infrastructures consuming data/services from EBRAINS and vice-versa.

3.3 Logical architecture

In this section, the Logical Architecture of the EBRAINS RI is presented. The Conceptual Architecture (see Figure 2) 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 in 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.

The EBRAINS high-level Logical Architecture displayed in Figure 3 can be abstracted into three layers. Starting from the top layer:

- **Front ends**. This layer abstracts 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). This includes web interfaces (web applications, public APIs) and downloadable executables (desktop, mobile, IoT etc.).

- **EBRAINS RI Service Layer**. This includes all software artefacts responsible for the provision of EBRAINS services (e.g. analysis, simulation, data curation) as well as the Infrastructure Middleware that abstracts all components responsible for managing, allocating, and monitoring the use of all underlying computing resources provided by the "Infrastructure" layer.
- **Infrastructure (Hardware & IaaS/HPC)**. Abstracts the infrastructure services (presented in Section 3.2.1).

As a sidenote: due to a history of independent conception and development, many components are highly siloed. This means that, although there have been efforts towards consensus and/or middleware/interoperability on the "data plane" (from the user perspective), there are almost none yet on the "control plane". Such "control plane middleware" though, already exists (UNICORE) for interaction with low-level resources (like ICEI/Fenix infrastructure). It is expected that, over time, as EBRAINS and its components become more sophisticated and less siloed, particularly by deduplicating and componentising non-"business-logic" like network/auth/resource-handling to their correct places, "control plane middleware" will become increasingly important for interservice cooperation. "Data plane middleware" will also be important for interservice communication (at least until standardisation of channels, protocols & data formats reduce the need for that). This is clarified here to pre-empt future confusion when conflating control plane/data plane, core-infra/component-to-component, etc while talking about "middleware".

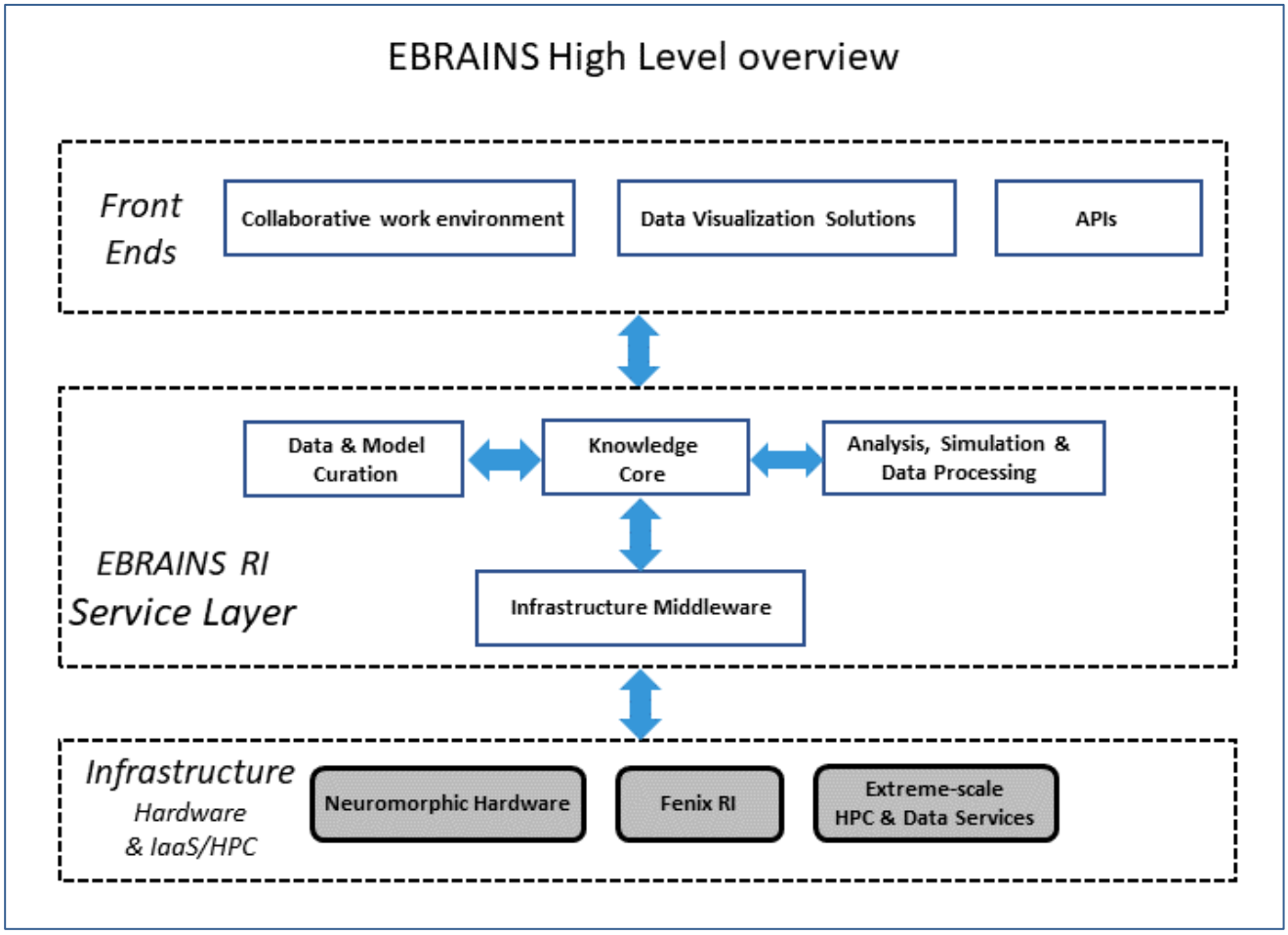

Figure 3: EBRAINS High-Level Logical Architecture

Based on the presented high-level overview, our efforts were concentrated on mapping all SGA3 components in this architecture towards creating a logical view of EBRAINS by imprinting the way components are relating/interfacing with each other and the different processing levels.

It needs to be noted that, at the start of SGA3, an important objective was identified and pursued by the EBRAINS TC. This 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 with 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 activities context of the EBRAINS TC and the EAI-WG.

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 start elaborating a concrete logical architecture design. By considering all the aforementioned material and keeping it in sync with the 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 diagram presented in Figure 4, which will remain a live diagram as iterations continue.

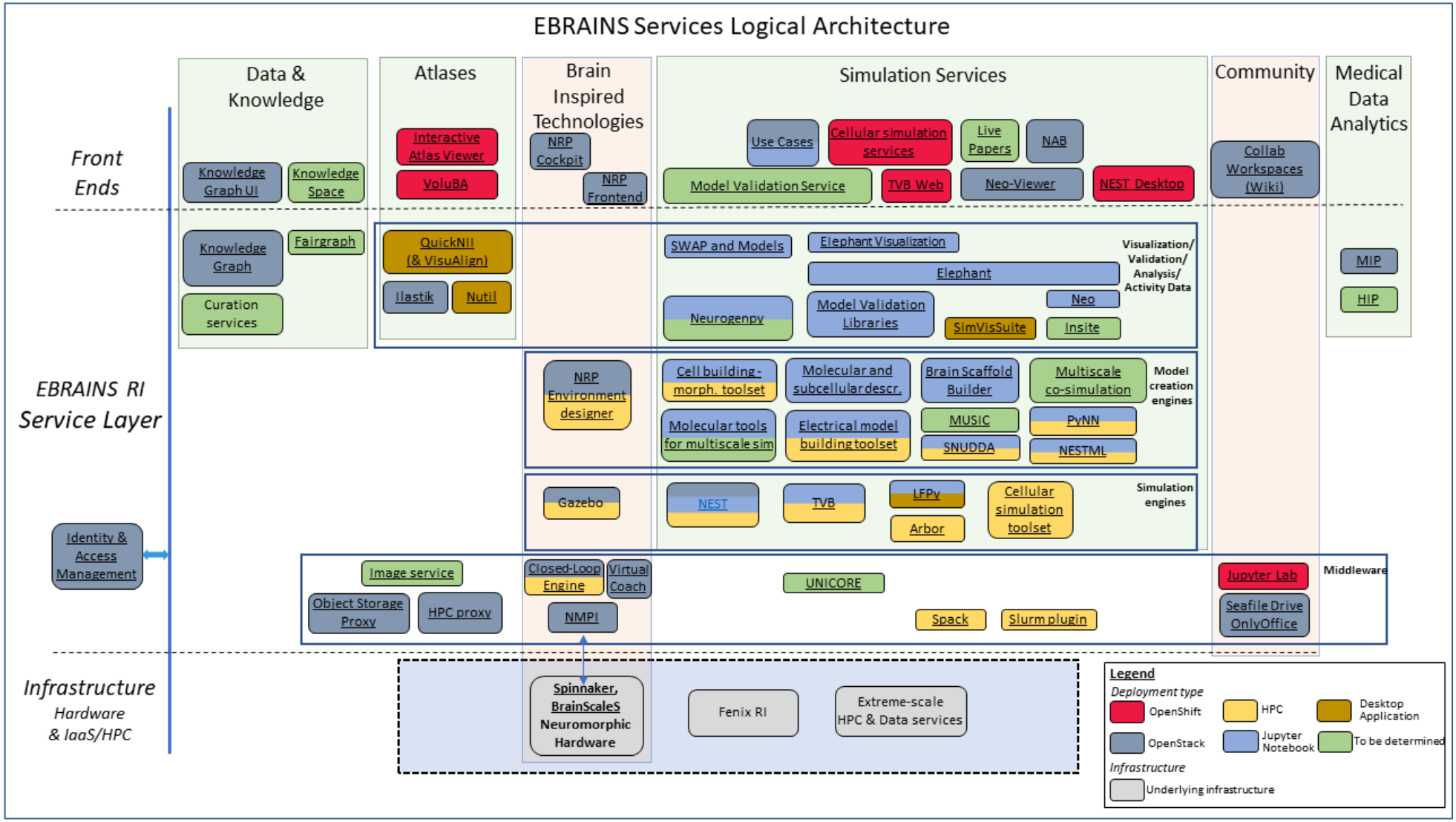

Figure 4: EBRAINS Services Logical Architecture

The diagram presented places all EBRAINS components (for further details, see Section 3.7) 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 Core RI services and the EBRAINS front ends. EBRAINS Core RI services can be further decomposed in four layers, namely, (a) the Visualisation/Validation/Analysis of Activity Data, (b) Model creation engines, (c) Simulation engines and (d) the Middleware layer.

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 different ways and at different processing stages and this would create distraction in such a dense diagram.

Another way to approach the diagram is from **left to right**. By observing "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 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 Services, Community and Medical Data Analytics, are also clearly visible with solid 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 OpenShift 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 3.5. For further information about the Deployment types available at EBRAINS see Section 3.8. Information about the Physical deployment of EBRAINS and how it enables the underlying infrastructure can be found in Section3.4.

3.4 Physical deployment

Following the presentation of "EBRAINS RI services 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 a first version of the Physical deployment diagrams had an extra layer of complexity due to EBRAINS federation and the fact that many facilities (EBRAINS is a distributed platform) that form the base infrastructure layer will come online at a later stage. The need to exploit the resources available to EBRAINS from ICEI/Fenix is self-evident, as otherwise the SGA3 Consortium would not be able to efficiently and effectively use the ICEI/Fenix 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 **current status** of EBRAINS physical deployment as it has been identified during EBRAINS Phase1 (Section 3.4.1) and ii) the **projected state** of physical deployment as it is envisaged for upcoming phases (Section 3.4.2). Information that supplemented their creation was extracted from the latest updates to the "TC Collab" wiki pages for the EBRAINS components (Annex IV: EBRAINS RI Components), the timeline for ICEI services [3], ICEI available resources [4], and Fenix AAI overview and updates (presentation slides from 1st FURMS workshop, SGA3 T6.6 Fenix AAI/FURMS Workshop).

The idea was to present the different available facilities (different locations) for EBRAINS as distinct slots and display in each slot the installed/available infrastructure/middleware services as well as the existing applications. 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 (ICEI/Fenix services) 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 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
  - **CSCS** (Centro Svizzero di Calcolo Scientifico/Swiss National Supercomputing Centre): Geneva, Switzerland
  - **JSC** (Jülich Supercomputing Centre): Jülich, Germany
  - **CINECA** (Consorzio Interuniversitario del Nord Est italiano per il Calcolo Automatico): Bologna, Italy
  - **CEA** (Commissariat à l'Energie Atomique): Paris, France
  - **BSC** (Barcelona Supercomputing Centre): Barcelona, Spain
- 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 **Annex IV: EBRAINS RI Components**. 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.

3.4.1 Present

In this section, the Present status of the Physical Deployment Diagram is presented, as it has been identified during EBRAINS Phase1. It can be clearly observed that **not all Fenix RI sites are yet fully operational**, as more and more of the base infrastructure services are coming online gradually. It is

evident, that currently **CSCS** is considered as the de facto main site (supports more workloads and the base infrastructure services are all online) and **JSC** as the de facto secondary site.

Most of EBRAINS middleware currently is installed **only on one site** and as a result key platform and simulation services **run only on one location**.

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

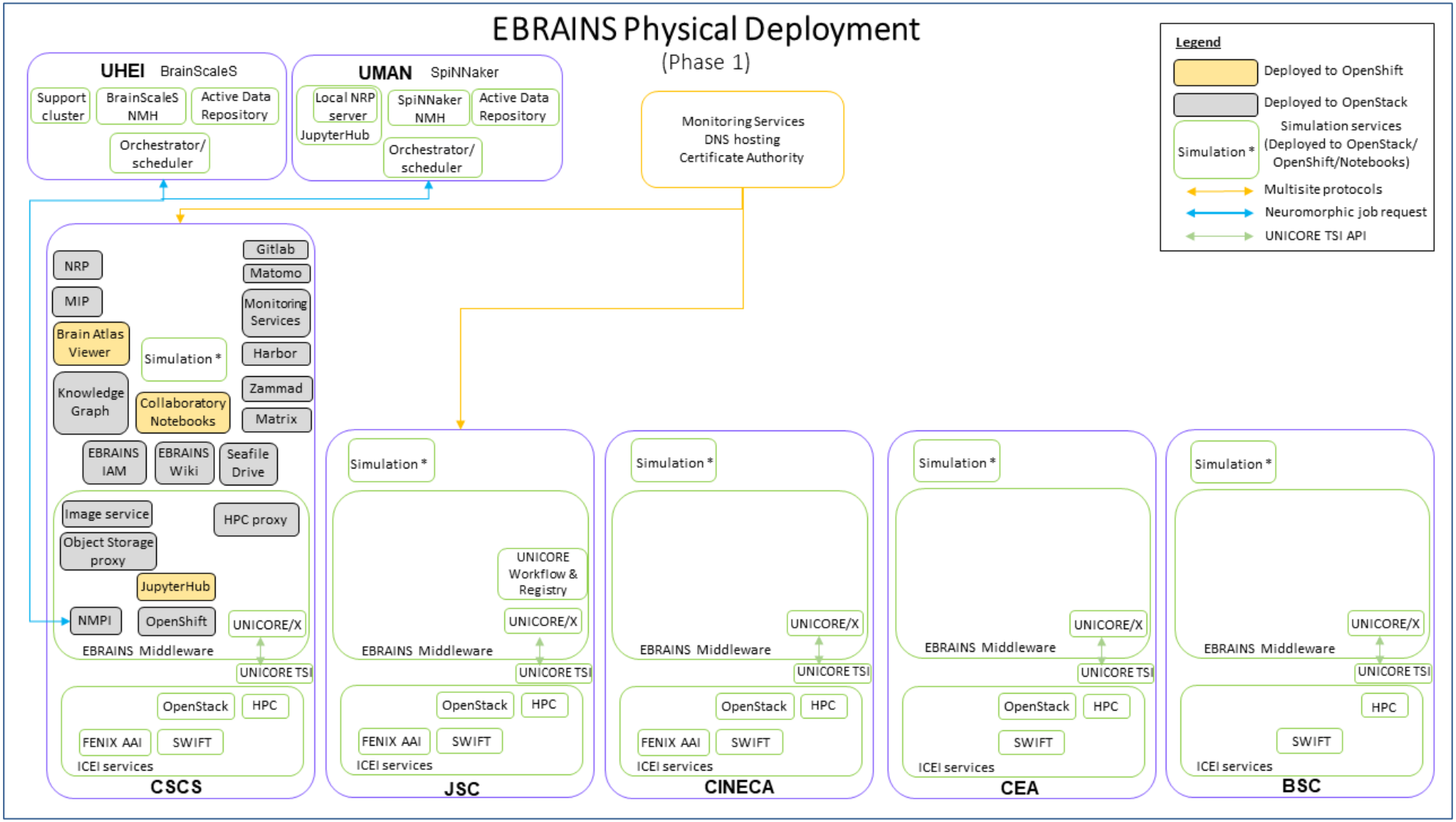

Figure 5: Physical Deployment (Phase 1)

3.4.2 Projected

In this section, the projected state of physical deployment, as it is envisaged for upcoming phases, is presented. 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 be deployed and scaled anywhere in the federation. Critical EBRAINS applications as well as infrastructure 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 OpenShift not only as clusters on multiple sites, but ideally supporting multi-cloud functionality across sites. If some 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.

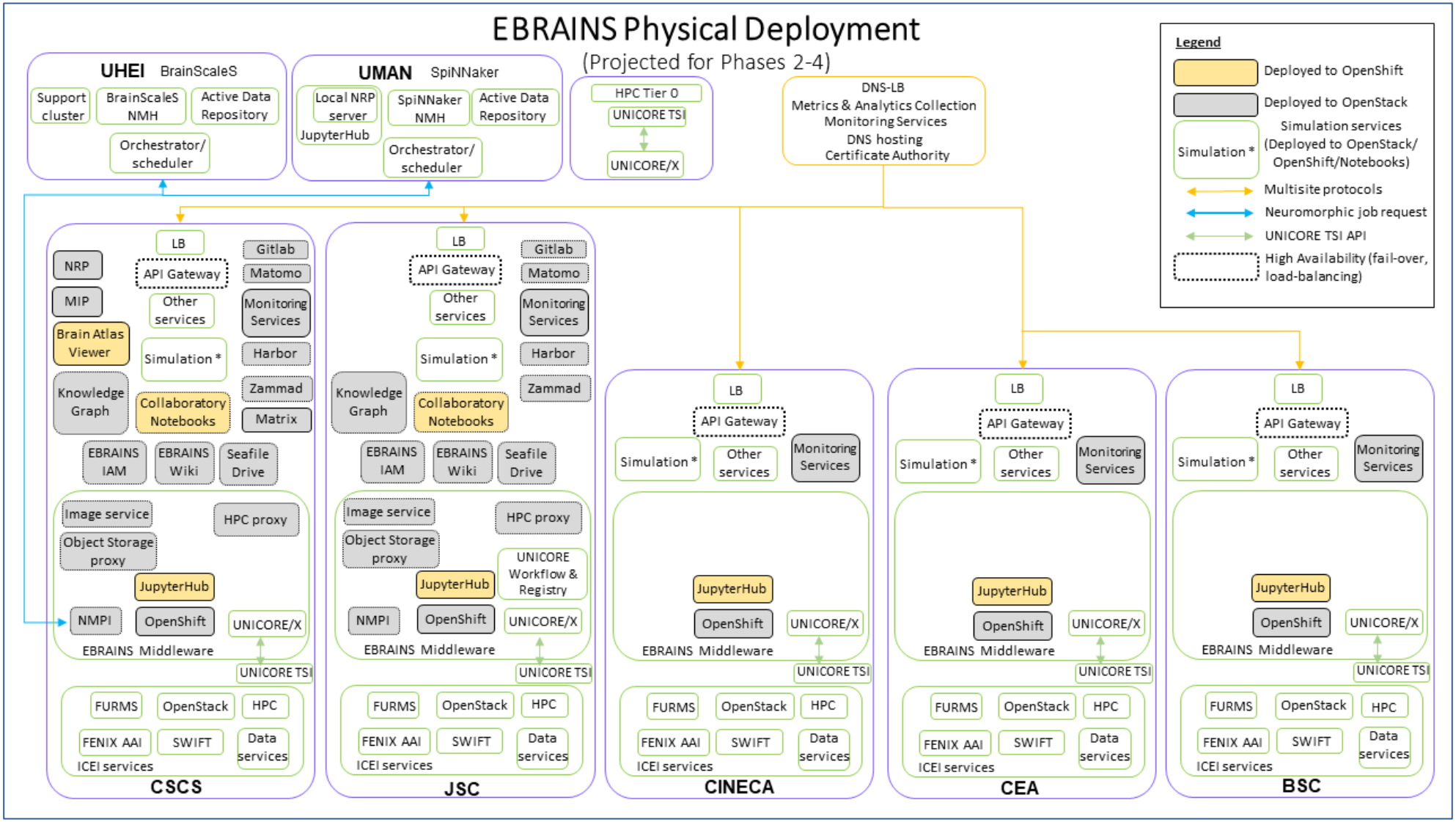

Figure 6: Physical Deployment (projected for Phases 2-4)

3.5 Service Category Illustration

As it is evident, EBRAINS is far too wide to approach it in its entirety with only the set of diagrams presented in the previous sections. Different audiences "demand" respective point of views, that will enable them to grasp the RI in the appropriate level of abstraction. This is achieved by detailing only the "in scope" architectural elements and as a result create various views of the RI.

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

This section serves as a kind of a preamble of the next phases of mapping the EBRAINS infrastructure architecture. The target is to create **one diagram per SC** to best inform the architecture to the targeted user communities. Towards this, a diagram (Figure 7) for Service Category #3 (SC3) "Brain modelling and simulation workflows" is presented to illustrate the scope of the current work and give an idea of what is expected to follow for all SCs. It is worth mentioning that 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, Figure 7 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.

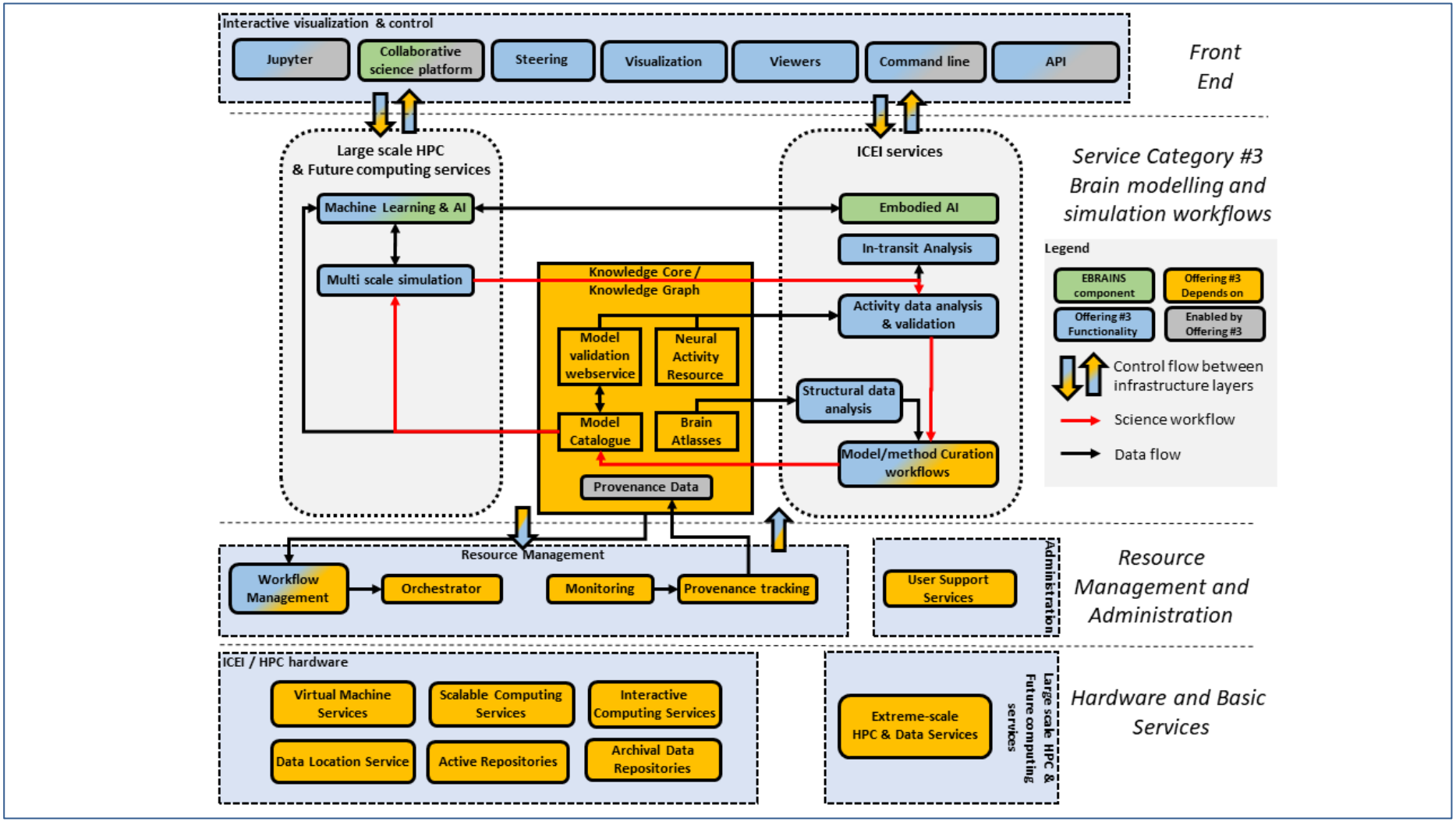

Figure 7: The current shared understanding of SC3 functionality and its relationship with the infrastructure architecture[4].

[4] 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.

3.6 APIs

3.6.1 Current status

Currently, the EBRAINS API ecosystem is at an "initial" stage, but with a clear roadmap. A basic collection of APIs ("API Catalogue") is under construction, with links to existing APIs and documentation (Annex III: EBRAINS components Web-APIs list).

At present, there is not a singular hard requirement from component authors in terms of API format. Several components provide only document developer-interfaces (e.g. a Python library with public functions) for performing high-level actions involving chains of requests against the actual application-interface endpoint (the API). These component owners will be encouraged to also document the actual API endpoints, to accommodate all use-cases (e.g. command line testing with tools like Curl, developers working with languages other than the supplied developer-interface implementations). In practice, nearly all component-provided APIs are REST web-services, several specified as OpenAPI/Swagger, with the added benefits of auto-documentation and client-code generation. At least for the near future, components which provide graph interfaces (like GraphQL/Cypher) can be presented semantically equivalently in the form of REST via an API gateway by appropriate formatting of parameters in the requests (whether in the URL, headers, or body-data).

All centrally provided infrastructure (for example OpenStack, OpenShift, Gitlab, etc) already provides mature REST APIs. Additionally, minimal proof-of-concept instances have been setup for:

- a centrally provided API-gateway and web reverse-proxy;
- a high-level "coordination & backend-workflows" API;

as precursors to real-world versions.

3.6.2 Roadmap

3.6.2.1 API Gateway

This will be the point of entry and exit (ingress & egress) for all external programmatic access to EBRAINS APIs, and its use will be encouraged for all non-trivial and non-critical-path intercommunication between different internal services too (both Enterprise-System services and as many as possible of the System-of-Systems components' services which are willing to be standards-compliant (see Section 3.8.1 for details). The three primary reasons for this are:

1) to maximise decoupling of the platform for transparency and for a meaningful platform-wide dependency-graph;
2) to help component owners reduce their cognitive-load to actual domain/business-logic while helping platform-maintainers to automate many of the other tasks;
3) to provide as much of the programmatic surface as possible under a unified (proxied) interface.

Point 1 reduces maintenance-overhead for the platform as a whole and would reduce "action at a distance" problems, while also enabling greater platform-wide debugging-power via all the extra metrics and analytics generation of proxied traffic that comes for free, and by limiting the "combinatorial explosion" of traffic format, and quirks that would otherwise need navigating when diagnosing system-issues.

Point 2 increases the quality and uptime of the platform by centralising and minimising maintenance impacts, including coordination of updates for security-vulnerabilities, etc. It also means that efforts to optimise the infrastructure are deduplicated, so that for example response-caching can be switched on and finetuned for most/all services with a single toggle, including at runtime per-service (by the service) by providing the usual HTTP response-headers for proxy-control (Vary, etc).

Point 3 makes it easier for humans to visualise and "mentally map" the platform in a contextually rich way, while also inherently encouraging developers to expose their services within the "principle of least astonishment" paradigm wherever possible. Lastly, having as integrated as possible (and as uniformly REST as possible) interface to the infrastructure means that integrated user-interfaces and portals (whether web-frontends, mobile clients, IVR, IoT, etc) can strive for 100% separation of frontend/backend concerns, and the different channels can reuse much of the API-handling logic - especially for example reusing JavaScript initially written for the browser in the other contexts).

Through simple low-code/no-code plugin configurations, the API gateway can centralise and deduplicate non domain-logic overhead (such as routing, load-balancing, authentication, authorization, sessions, request/response data/header transformations, ACLs, caching, metrics/analytics generation, URL rewriting, logs routing, cluster ingress controller automation, continuous-deployment strategies such as canaries/blue-green/etc, and all the security tasks such as TLS termination, bot detection, IP blocklisting, request/response size-limiting, rate-limiting, request validation, CORS, etc). Note that we would only use the “continuous deployment strategies” mentioned here in the context of the testing environment because we will use staged/packaged platform-releases for production.

As the API gateway can process all HTTP methods for REST, its basic functionalities like routing, caching, etc can also be re-used for reverse-proxying of miscellaneous web-traffic (primarily GETs, with POSTs for forms) "for free". For such payloads, it is recommended to not include URI-rewriting in the routing though (just routing by Host-header, SNI, etc) because also heuristically rewriting all URIs in the unstructured requests and responses is fragile at best. Providing gateway handling for APIs can be routed by “Host-header/SNI” or for those who wish to appear as integrated into a single API by “URI rewriting” (for example [source]/api/v1/xx -> [destination]/xx), but they would need to ensure relative-links-only (and no hardcoded hostname) in the response-bodies. Because a single instance would be a bottleneck and a "single point of failure" (SPOF), this would ideally be deployed highly-available and load-balanced, and ideally geographically redundantly (e.g. via DNS load-balancing). When combined with methods like "sticky sessions", this would be the easiest way to also provide the web-portal and frontends from multiple sites across Europe (achieving geo-redundancy and geo-optimisation for dynamic traffic, going beyond the static-content optimisation provided by CDNs).

The proof-of-concept was setup on Kong, which is a stack built on PostgreSQL, Nginx, Openresty, and extra Lua functionality. Alternatively, Cassandra can be used in place of PostgreSQL if greater “eventual consistency” flexibility is desired. There exists a Kong Ingress Controller which assists Kong to declaratively control OpenShift/Kubernetes routing, health-checking, etc. as well. The intention is that this PoC will be developed into a combined production service (API Gateway, web reverse-proxy, & central coordination API provider) for Phase 2 for the offered APIs and will be available as a (preferred where feasible) alternative for the onboarding services.

3.6.2.2 Central coordination & backend-workflows API

This can be provided with minimum effort from the same stack that runs the API gateway (as it already includes a reverse-proxying web-server with a built-in application-server and backing database). By embedding readily-available libraries in the webapp-configuration and triggers in the database, it is possible to host an auto-generated REST interface on the underlying database, providing a no-code (and no daemon-process) interface to central data, and to backend-workflows (for locating, triggering, monitoring, etc). Backend-workflows would be chains of calls to other services, with possible retention of ephemeral state, to realise high-level tasks without reimplementing functionality already provided by lower-level services. These workflows could even be contributed by component-owners as "plugins" for execution by this API, and are not the same entity as is referred to elsewhere in this document as (end-user) "Workflows". An extra point under investigation is whether it would be best to utilise a dedicated namespace within the “EBRAINS Knowledge Graph” to host the RI and /or backend-workflows data/metadata as the knowledge-store behind this API. Either way, the API should make such implementation-distinctions invisible for consumers (except that it might be feasible to provide additional “graph semantics” if backed by a graph rather than classical relational database).

3.6.2.3 Evolution/addition of component/gateway APIs

Possible future decisions to also (or instead) unify different API formats or protocols (e.g. GraphQL, REST-over-ZeroMQ, SQL-over-Rsocket, etc.) behind a central gateway is an open topic, presently with no specific goals or timeline. This will probably be guided by emergent requirements and changing needs during the RI's evolution (and to attempt to pre-empt such things risks the "evil of premature optimisation"). In the meantime, component owners who have such special requirements must bear the operational burden (and the need for especially thorough documentation) of not having those gateway-proxied yet.

3.6.2.4 API/ABI of non-service products (libs/applications)

Even library/app deliverable products have a (non-runtime service) functional API, and in many cases (especially include-files for compiled code) a set Application Binary Interface (ABI). Even though it would be far too restrictive, opinionated and unproductive to standardise these in a small list, it will however always be preferred when code is maintained in an "API friendly" (and "code-reuse friendly" way), avoiding heavily coupled spaghetti code, modularising and encapsulating in future-proof ways, etc. The RI will not "only be as good as its contributed components", it will "only be as good as its maintainable, shareable, and reusable components". This also applies to the auxiliary aspects of the code; in particular, how easily they can be mechanised. For example, maintaining informative inline documentation (for automatic docs-generation using Doxygen, JavaDoc, etc), or even literate-code when appropriate (using pyWeb, Eve, etc.), enables extremely intuitive documentation for easy-onboarding and code-reuse, with minimal maintenance after the initial overhead. A form of literate-programming more familiar to EBRAINS users is of course a Jupyter notebook.

Various other forms of "auxiliary API/automatable methodologies" also apply in code products (delivered for reuse) in a way that is even more important than in code for running services. Examples are inline unit-tests, design-by-contract clauses, standardised-style pragmas for editors/linters, static-analysis hints for supporting tools, source-maps to aid debugging at runtime, license-tags (e.g. SPDX) for mechanised compliance-checking, etc.

3.6.2.5 API Versioning

As is best-practise for any form of software delivery, when all of the components contributed to a specific version of an infrastructure ("distribution") for the production release have their own specific version (especially in an easily parsable/mechanisable versioning format like SemVer) it simplifies the release process considerably. For example, releasing platform version 1.1 becomes a mathematical exercise of resolving all operational and dependency conflicts between component A version 1.2, component B version 1.3, and Component C version 2.1 (and then of course "all system/solution-tests passing green"). This can be improved dramatically for the release of an API-driven service platform when also using API versioning. For example, versions 1.3 through 1.9 of component A may provide API version 2, and versions >= 2.0 provide API version 3. Then for interoperability component B can just depend on component A's API version 2 and automatically expect to interoperate with all component A versions which conform to that API version.

3.7 EBRAINS RI Components

EBRAINS components (in the context of this section) are the (science) products, services and infrastructure middleware offerings developed and/or updated through the course of SGA3. EBRAINS components are participating in the integration process that leads to the release of the EBRAINS RI.

At the start of SGA3, an important objective was identified and pursued by the EBRAINS TC team: identify, disambiguate and setup a comprehensive, authoritative and always up-to-date knowledge base of EBRAINS RI components. To meet this objective, the TC team compiled an EBRAINS RI components catalogue that led to a shared knowledge base (**EBRAINS Components Bluebook**). The

EBRAINS Components Bluebook is intended to be used as a central source of information for TC purposes regarding the various EBRAINS RI components and their life cycle.

The TC team aimed to gather comprehensive content that spans both static and dynamic aspects of an EBRAINS RI component's life cycle. Moreover, it was considered important to feature in one easy to read and update wiki page all necessary information that both a technical and non-technical user would require to have a clear overview. The requested information for the EBRAINS Components Bluebook concerns the following fields:

- Golden Paragraph - Short description for the component being developed.
- End Users – List of End User Classes as well as End User Groups in order to be able to accurately determine use cases and organize in an ad-hoc way communication necessary for the validation of the component/service. End User Classes consist of scientific or commercial fields that the component/service targets. End User Group of a component is made up of people who have expressed an interest in using this component.
- Application category – The Application category/Type of the component with respect to the wide-ranging functionality available throughout the EBRAINS ecosystem.
- License - The license governing the use or redistribution of software developed for the component.
- Documentation – Links to the latest documentation.
- Architecture - Detailed information regarding the architecture of the component.
- Roadmap - A high-level visual summary that maps out the vision and direction of the component's offerings over time—conveying the strategic why and what behind what it is built.
- Releases & Schedule – Table featuring releases (versions & release dates).
- Repository – Link to the source code repository of the component.
- Issue tracking - Link to the issue tracker of the component.
- Integration & Testing – Established Integration and Testing practices for achieving delivery of code changes more frequently and reliably.
- Deployment status – Physical deployment status (location, technologies involved).
- Dependencies – Dependency lists for i) software dependencies, ii) other depending EBRAINS components, iii) from tasks/activities within the component's WP or Task.
- Communication – Technical communication channels.
- Points of Contact – Contact points (component's owners).

The EBRAINS Components Bluebook (currently provided in best-effort form) aims to authoritatively document the exact development, testing, delivery, and deployment status of all EBRAINS RI components. As such, it provides clarity and transparency, streamlines communication, reporting and intra-SGA3 collaboration, and facilitates TC activities across all levels. Where complementary information is maintained (e.g. PLUS, HLST, Knowledge Graph) or must be compiled and reported (e.g. EBRAINS RI Roadmap) the corresponding links to the above activities, as well as flow of information, have been considered to avoid duplication of effort. The EBRAINS Components Bluebook was initially populated with information submitted by the SGA3 Consortium in the form of short 2-page structured reports requested and delivered before the TC Kick-Off virtual three-day meeting (29 May, 2-3 Jun 2020).

This information was assessed by the TC team for completeness, technical maturity, SGA3 conformance and cross-checked with all available background material (SGA3, SGA3 background/internal documents, architectural diagrams, past Deliverables). With the planned availability of EBRAINS RI-wide technical instruments to automatically monitor and export testing and integration progress (e.g. CI/CD), content included in the EBRAINS Components Bluebook is planned to be populated and exported in an automatic and semi-automatic manner, as appropriate.

The TC team processed and integrated all available information under the TC Collab, a Collaboratory hosting up to date information and guidelines for all TC activities and accessible by all SGA3 members. For all components (currently 52 in total) information was organised following a common reporting template and initialised with the content prepared by the SGA3 Partners. The template contained specific guidelines for all factors to be monitored (e.g. development roadmap and deployment status), as well as component-specific guidelines. After that, all responsible SGA3 partners were individually contacted and instructed to complete/update and constantly revise their respective component page in the TC Collab as a formal continuous activity. In parallel, the TC team initiated a series of coordination activities with all relevant SGA3 organisation groups (e.g. HLST, EBRAINS RI Coordinator) to avoid duplication of efforts, ensure complementarity with ongoing/planned actions, and maximise the added value of the EBRAINS Components Bluebook for the entire SGA3 project, beyond the strict confines of the TC. As such, information may diffuse from the EBRAINS Components Bluebook to the various HBP SGA3 reporting lines:

- the Knowledge Graph, as a reliable source of information to be curated and imported in the graph database.
- the PLUS platform: for example, information on stable releases can be propagated to update respective PLUS pages, as well as to update the desired KPIs that monitor software components status, based on KPIs derived from EBRAINS Components Bluebook's information.
- EBRAINS website front-facing roadmap; components' dedicated roadmaps can be aggregated to an EBRAINS RI roadmap.

Finally, the TC team has planned a monthly periodic review of each component's page in order to offer comments, instructions, and guidelines based on the course and mandates of SGA3.

As the EBRAINS Components Bluebook is a living organism, all involved stakeholders can benefit from its maintenance as it provides a clear, quick and unified overview of the development efforts and status towards the release of EBRAINS RI.

In Annex IV: EBRAINS RI Components a partial view of EBRAINS Components Bluebook is presented with a list of the currently tracked EBRAINS components along with relevant accompanying information to provide the reader with a quick overview of the current development efforts.

The intention is to transfer the content and functionality of the EBRAINS Components Bluebook from the static wiki pages where it currently resides, to a more appropriate structured and programmatically accessible solution which can leverage Continuous Integration features for automatic updates to its content (and for extraction of semantic data for other uses like analytics, visualisations, triggers for external alerts, etc.).

3.8 Guidelines for Component Developers

This section provides EBRAINS application/service developers with an overview of the deployment types that are possible 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 purely for optional use, while others constitute required or recommended methods for interacting with the platform. See Section 4.2.3 for those details. Please also note, that upcoming iterations of the RI are intended and expected to include resources which are not present in this iteration. For example a "universal service mesh" (spanning services across container & VM clusters, including multi-site) would be of benefit if/when it becomes feasible and there is enough time to implement it and onboard component-owners.

3.8.1 Overview of the 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 back end 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 of 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 4.2.2), but these inherently exclude some of the requirements of ES components. For an independently hosted "encapsulated" service many platform-level library-dependency/packaging issues become irrelevant, but as a balancing factor this also means the responsibility for continuity of that service remains on its component-owners.

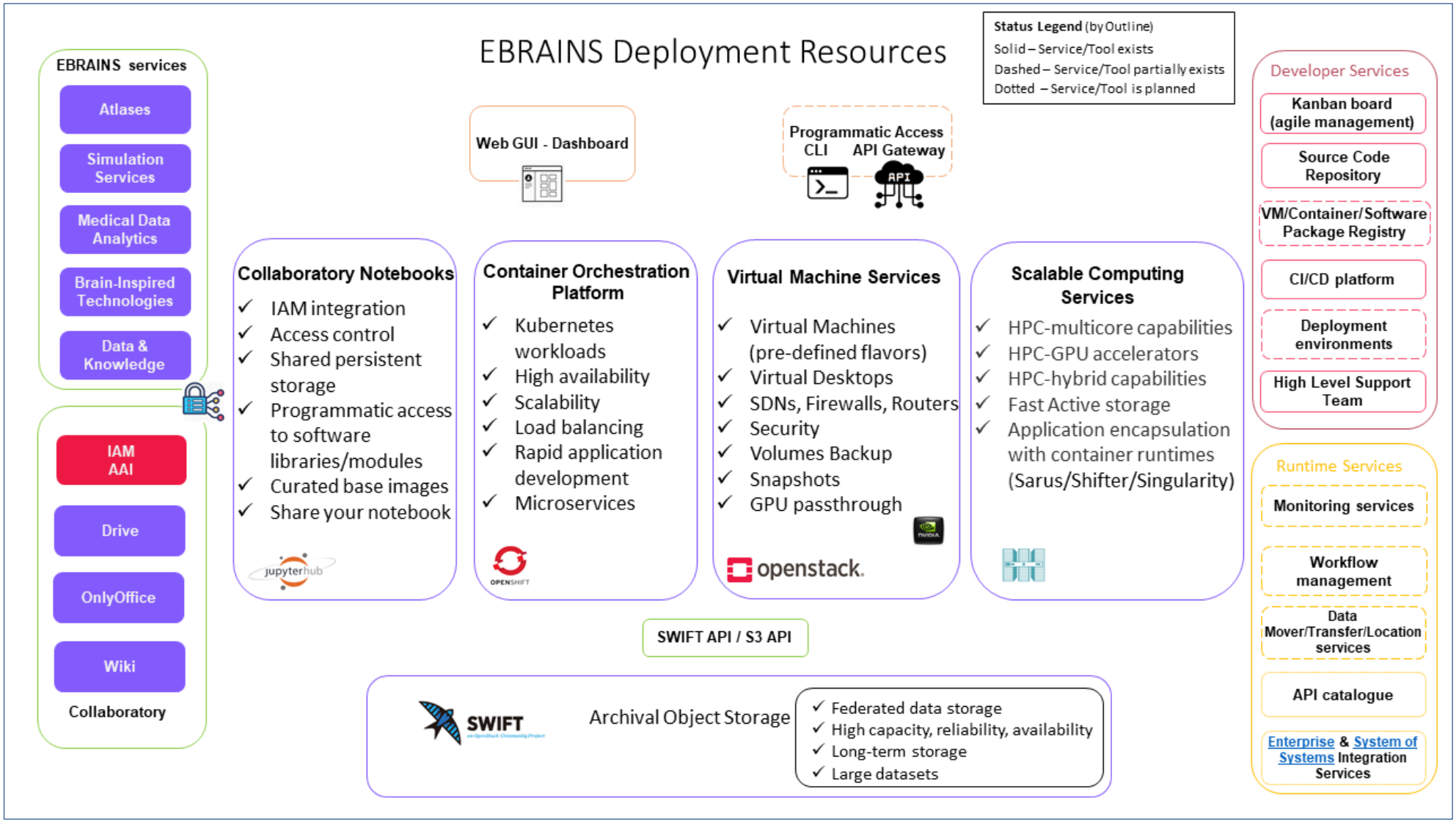

Figure 8: EBRAINS Deployment Resources

3.8.2 How to integrate components to EBRAINS RI

Except for edge-cases[5], there are **four deployment types** 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 **OpenShift**.
- **Jupyter notebooks** as ephemeral, on-demand services in the **Collaboratory** environment.
- **HPC libraries and applications** as reusable products within **HPC systems**.

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.

3.8.2.1 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. 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.

[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

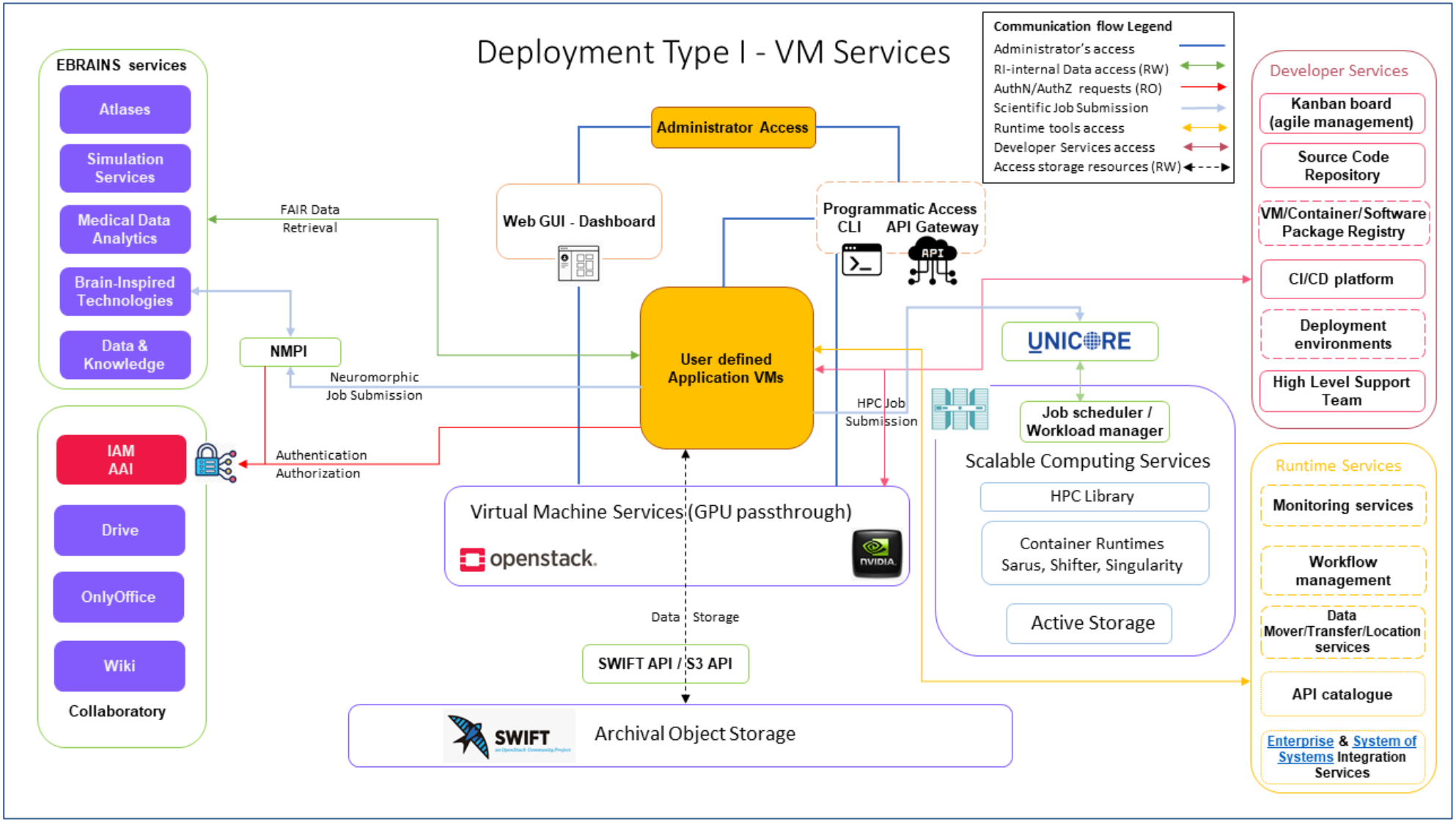

Figure 9: Deployment Type I

3.8.2.2 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, OpenShift 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 OpenShift Container Orchestration Platform. This format is recommended especially for stateless applications, or for stateful applications where high-availability, configuration-management, dynamic horizontal-scalability, or sandboxing is worth the extra abstraction-layer. Containerised applications deployed by accredited members can access all EBRAINS services, the DevOps tools, can submit jobs to HPC systems and can have access to object storage for long-term, safe, and secure storage. Administrator access to OpenShift for managing the containers can be accomplished either via web dashboard, CLI, or programmatic access.

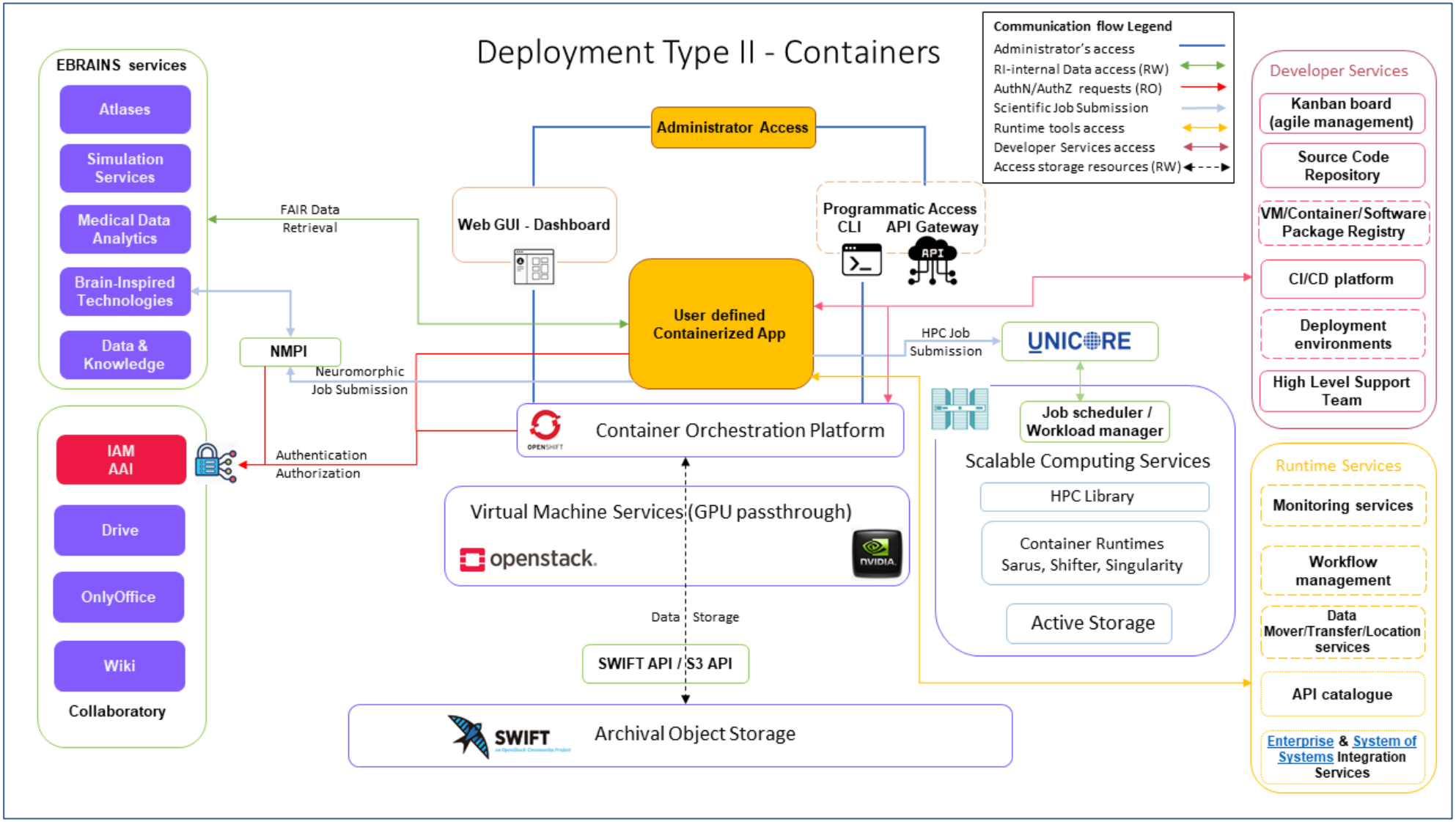

Figure 10: Deployment Type II

3.8.2.3 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.

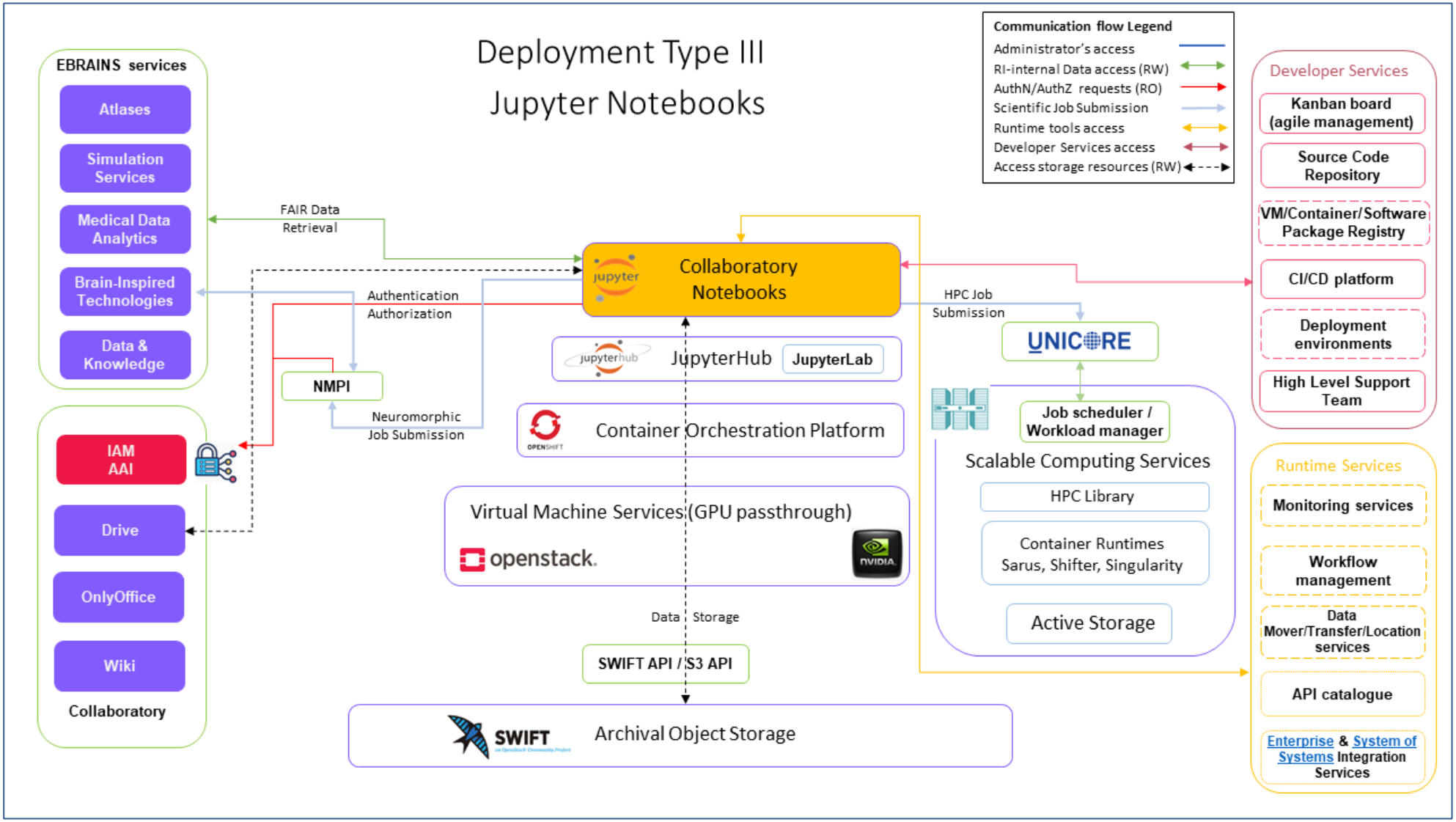

Figure 11: Deployment Type III

3.8.2.4 Deployment type IV – HPC Library / Container runtimes

Executable code can be “deployed” (released and installed, but not 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 into a container image can be easier than providing all dependencies for the application via operating system packages.

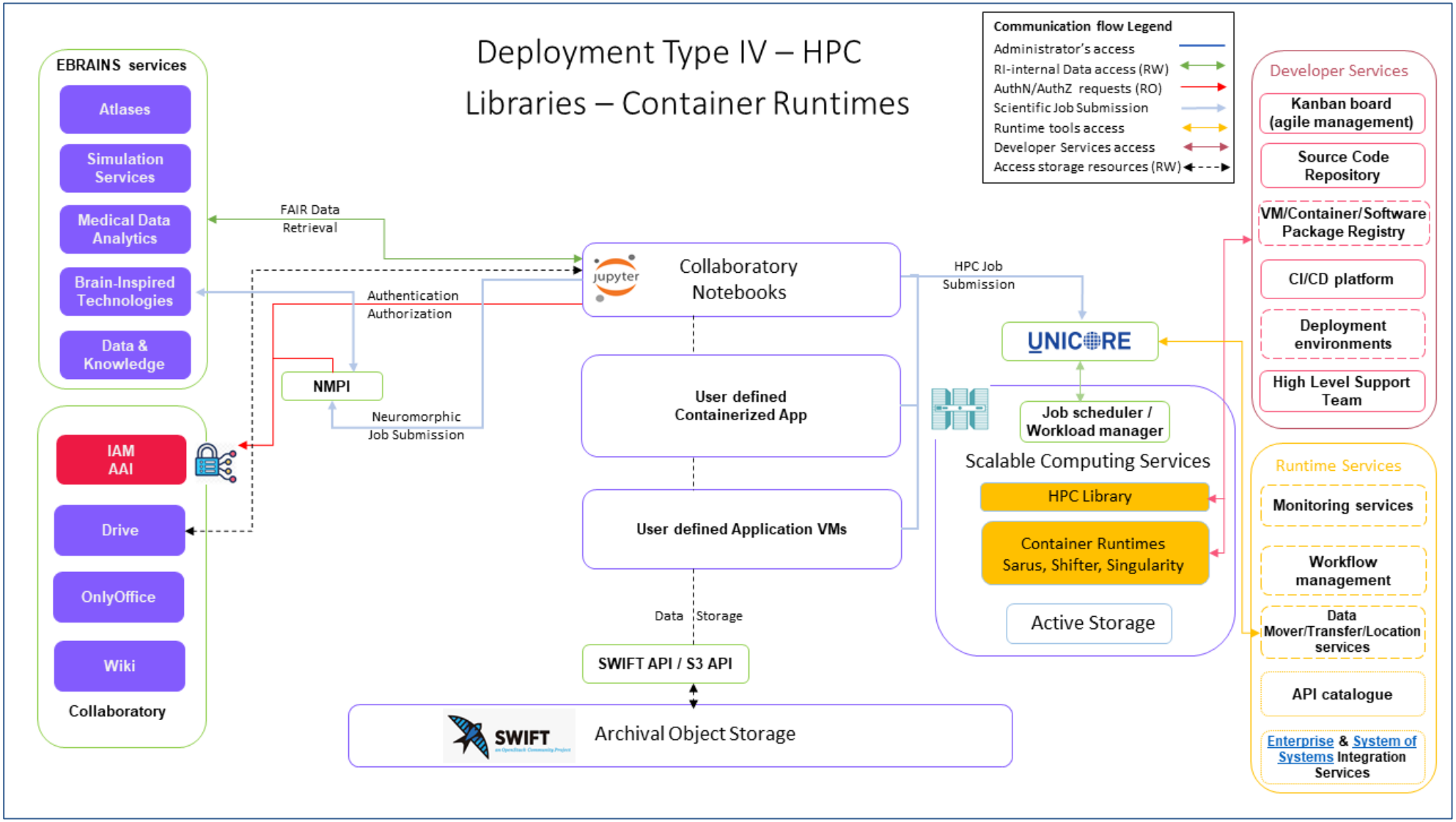

Figure 12: Deployment Type IV

3.8.2.5 Notes common to all deployment types

In addition to VMs, cloud containers, HPC containers and software-bundles being provided in their respective "packaging-source form" (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 encouraged, there wherever feasible that is necessarily more of a request for "best effort".

3.8.3 Common Licensing Guidelines

In this section, we present some initial guidelines for licensing. Licensing is part of the guidelines, since it is connected to EBRAINS platform provisioning cost. We are trying to avoid depending on closed licenses, as the increase of cost will hinder scaling in case of a closed license application. Nevertheless, further discussions with the component owners are necessary for the clarification and analysis of these guidelines, consisting just our proposal, since every component owner is free to choose their own license:

- All software components should be licensed with a permissive[6] license (preferably the Apache License[7]), which allows proprietarisation, except for cases where:
  - The software is already proprietary;
  - The software is an extension of existing software already licensed; all SGA3 contributions must conform to the software's original license;
  - The software applies non-permissive software; these cases need to be discussed with the TC to avoid potential future problems related to commercialisation.

These guidelines are an attempt to address the problem of combining components with different and possibly incompatible legal licenses. Apache license is mentioned only as a suggestion, since it explicitly defines all the terms and concepts that it uses, avoiding ambiguity, and it is more specific about certain circumstances than the other permissive licenses, so this can be helpful in case of a dispute.

EBRAINS RI should ensure that the combination of the different components will not violate any of the involved licenses, with EBRAINS RI having its own.

[6] https://en.wikipedia.org/wiki/Permissive_software_license

[7] https://en.wikipedia.org/wiki/Apache_License

4. EBRAINS RI Technical Guidelines

This section provides comprehensive directions to SGA3 developers, component owners and scientists. The main goal is to provide technical information regarding the EBRAINS RI offerings and present the requirements and recommendations for a component to be integrated and deployed in the EBRAINS RI.

The "Developer and Runtime services" are demonstrated and summarised in order to facilitate application/service developer's familiarisation with the set of offerings of the EBRAINS RI (Section 4.1). Scientists, developers and component owners will benefit, not only from the Collaboration tools and services that EBRAINS offers, but also from the capability to use the EBRAINS Knowledge Graph as a multi-modal metadata store for storing, fetching, depositing data and metadata. In order to create a streamlined and guided procedure for integrating components to the EBRAINS RI, "Technical Guidelines" for a component's integration to the RI are presented (Section 4.2). These guidelines aim to present standard, best-practice tactics to achieve smooth integration of applications/services to EBRAINS, featuring a minimum set of common, yet critical requirements that are intended to raise the technical foundation bar across the SGA3 board. The guidelines may be adapted on a case-by-case basis depending on the actual needs of each component/service on the path of integration. Further, our approach regarding the indicators to be used for quality control for the entire RI is presented (Section 4.3).

This section has been developed under the guidance, feedback, contribution, and review of the EBRAINS Software Quality (ESQ-WG) and EBRAINS Software Delivery (ESD-WG) Working Groups.

4.1 What EBRAINS RI offers

In this section, we present what EBRAINS RI can offer to the Consortium, supplementing the presentation made in Section 3. Developers who are already part of EBRAINS, component owners and scientists can find descriptions for some useful collaboration tools and services (Section 4.1.2), such as the Collaboratory, the Kanban Board and the Instant Messaging. Especially for new developers contributing to EBRAINS, a whole section is dedicated to onboarding material (Section 4.1.1) that Technical Coordination gathered from different sources of truth. In this way, it is easier for a new developer to capture the technical essence of EBRAINS and its underlying Infrastructure than before, whem information existed but was spread across many different places. Developers can also read more about the authorisation and authentication (Section 4.1.3) procedures for services at EBRAINS. Finally, they will find useful information and supplementary material for developers (Section 4.1.5) and runtime services (Section 4.1.6). Scientists can read about the use of Knowledge Graph as a metadata management system (Section 4.1.4), and how they can share and find FAIR (Findable, Accessible, Interoperable, Re-usable) data.

4.1.1 Developers' onboarding guide

In such a diverse ecosystem, like the EBRAINS RI, there is a need for a "Single Point" of reference, available not only for the new members of the various projects, but also for those who are interested to integrate their tool with another one or, for those curious enough to expand their field of interest, even to different ones. Part of the Technical Coordination work was to identify different tools and services, identify their technical aspects and draw a common approach for all tools services that need to be integrated with the EBRAINS RI. Therefore, a single point of reference for technical aspects was needed, since valuable information constantly flows from many different sources. Other than technical aspects, the Single Point of reference needed to host important information for various other aspects including administrative procedures, code sharing capabilities, documentation sources, data sharing places and online collaborative tools that facilitate interactive work among different collaborators. Hence, a unified document with the respective guidance information was assembled; the ***Developers onboarding guide*** is available at [5].

4.1.2 Collaboration tools and services

The EBRAINS RI features a broad community, ranging from experienced application/service developers, to neuroscientists, researchers and even students. The inherent heterogeneity, that stems from the variety of the community's branches is reflected in the heterogeneity of the tools that EBRAINS RI offers. It is important to have easy-to-use tools and services that can facilitate collaborative work and are available to the entire Consortium. Note that this section only provides a summary of tools/services provided to all developers contributing to EBRAINS and does not aim to offer a comprehensive presentation of each tool/service given that detailed documentation is publicly available for each one of them.

4.1.2.1 The Collaboratory

To fulfil the abovementioned needs, EBRAINS offers "Collaboratory", a community platform for collaboration built upon an integration of JupyterHub (notebooks), OnlyOffice (collaborative editing), Seafile (shared drive), and XWiki (wiki). In that sense, collaboration has become easier between individual people and teams. The Collaboratory creates an environment where scientists can identify fellow researchers and developers, form teams, coordinate collaborations, develop live code via Jupyter Notebooks, document methodology and results in Wiki pages and Office files, store data, and publish work in a safe environment, which is entirely self-hosted. The Collaboratory is a valuable source of various information, hence all EBRAINS members have access to. Since EBRAINS community is a large community, there was a need to have organisation structure. That was accomplished by units and groups. Units are static structures that identify the position of Project members and links all users to their institution. They represent the organisation structure of the HBP, EBRAINS and Partners. Task leaders and project managers are responsible for keeping track of people being involved in EBRAINS and requesting support team to remove them from respective units upon departure. Groups on the other hand are more dynamic structures with group admins choosing who can be a member or not. For more information related to Collaboratory you can refer to The Collaboratory wiki page [6].

4.1.2.1.1 What The Collaboratory Offers

The Collaboratory, as described above, is a tool for collaboration, a way to do interactive work and share data with EBRAINS users. The entry point for making that happen is known as a "collab" which is the workspace inside the Collaboratory. A collab extends across multiple services to share its content with a team of users who have set permissions within that collab [7]. From this page, users can create, edit, browse wiki pages. They are a convenient way of publishing content in a specific target group by choosing if the collab will be public or private and by managing permissions for people inside the EBRAINS Consortium. A collab also gives the opportunity to users to interactively work on Office documents via OnlyOffice and store them inside a dedicated collab's Drive. All files inside Drive are version controlled and users can roll back to a previous one quite easily. Inside Collab users can access JupyterLab notebooks and spawn instances of them on a hosted JupyterHub, in a reliable and fast way. Not only are JupyterLab notebooks one of the most promising interfaces for interactive work (Section 4.1.6.1) among developers and neuroscientists but they also provide great benefits like an easy way for a user to run large jobs in an HPC system or Neuromorphic hardware. Finally, developers can contribute their applications as community apps, make them available, and provide end-users with these applications (as extensions to the Collaboratory capabilities) to be used in their private collabs. The procedure that a developer must follow is described in the dedicated collab [8]. At this point, some applications are already available like for example Simulation Services, Neuromorphic Computing and Neurorobotics Platform. In a nutshell, developers can register their application in the Community Apps Catalogue so that users can find the application. The "OpenID Connect" Identity Provider (part of the Identification and Access Management system described in the "Authentication and Authorisation" subsection) reuses the Collaboratory permissions for authorisation.

4.1.2.2 Kanban Board

A critical requirement in such a large project, where different, parallel, distributed development efforts occur, is to minimise potential bottlenecks and quickly broadcast implementation solutions of general interest across the board. In that sense, a central coordination point for developers was needed. A dedicated GitLab Kanban board is currently used by EBRAINS TC for creating issues, keeping track of development progress and effort, and resolving any blocking points that relate to the development of the EBRAINS RI, as well as the EBRAINS components integration process. Not only developers, but also neuroscientists, researchers, and component owners, who are members of the SGA3 Consortium, are welcome to create issues - also known as cards- in the GitLab Kanban board that EBRAINS provides. The possibility to create new cards at the board and consequently request for dedicated development effort and technical solutions, is available to all EBRAINS accredited members.

Moreover, a prioritisation process, where 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, is already in place.

4.1.2.3 Instant Messaging

The benefits of instant communication channels across a board that consists of hundreds of members are quite obvious. Having public and private discussion rooms where information can be quickly circulated across a great number of people can be very powerful and leverages the speed at which informal, yet immediate technical support may be offered. With that in mind, instant messaging functionality provided by EBRAINS to its users (both internal and external) was a functionality that was missing for a long time.

Currently, EBRAINS offers to its users an instant messaging service based on Matrix [40] Open-Source Platform for secure, decentralised real-time communication. It is an alternative to products like Slack or Mattermost. Any EBRAINS-accredited member can connect to the Matrix service as it is integrated with EBRAINS Identity and Access Management system. It offers end-to-end encryption by default and recovery keys for checking history conversations. Although EBRAINS now offers an instant messaging service, it is up to the Consortium development teams to use it. If a development team wishes to use an external instant messaging service, it is free to do so. EBRAINS only provides an instant messaging facility, but does not require developers to use it (opt-in).

Multiple clients are available for the ‘Matrix’ service. For the HBP, Element client was used. From the official documentation [9] “Element is the flagship secure collaboration app for the decentralized Matrix communication network. Element lets you own your own end-to-end encrypted chat server, while still connecting to everyone else in the wider Matrix network”. ‘Matrix’ for the HBP is hosted under element.humanbrainproject.eu. Users can also download the client in their devices, since web client comes with limited functionalities. Every member with EBRAINS credentials can log in to both Web and Client. Users in ‘Matrix’ can send direct messages to other users with default end-to-end encryption in place. Encryption may be optional, only if the user wants to create a room with other users. When end-to-end encryption is enabled, either by default or by selecting that option when creating a room, users will have to create a recovery key so they will be able not to lose any history conversations. In that way, each time users log out and log back in, they will have to add their recovery key to be able to see previous discussions and conversations. In rooms that were created without end-to-end encryption, every member can at any time check the history without providing a recovery key, since those rooms are public to EBRAINS users. More detailed information for EBRAINS members interested in ‘Matrix’ can be found in a dedicated wiki page [10] under Collaboratory. As of January 2021, 40 EBRAINS users have already signed in to ‘Matrix’ with their EBRAINS account. Some teams have already migrated from Slack to ‘Matrix’ too, or will do so soonThere are already some public rooms (e.g. HBP-TC, Townhall) available for the users. In most cases, users used those public rooms to ask questions regarding an issue they were facing, to bring to our attention a possible failure of the system, or to ask for documentation and more. Some further adaptations for ‘Matrix’ instant messaging and possible fixes throughout the Project are expected to be needed with respect to user needs that may arrive.

4.1.3 Authentication and authorisation

Authentication and authorisation are critical when it comes to security. In general, authentication is the procedure where the system identifies a user. Authorisation, on the other hand, is the act of granting or denying the right of a user to complete an action. The centrally provided Identification and Access Management (IAM) service for EBRAINS runs on Keycloak[8], which acts as Identity Provider Broker (IdP Broker) to independently supplied IdPs and also provides an OpenID Connect (OIDC) IdP itself for authentication and authorisation (based on OAuth2.0 authorisation). Practically, it authenticates EBRAINS users and authorises them for the different services. Components of EBRAINS that fall in the "service" or "application" category must have centralised authentication and authorisation via the EBRAINS IAM service. As ofJanuary 2021, all services authenticate EBRAINS users via that service. For services to be integrated with EBRAINS, they also need to authorise their users via the EBRAINS IAM. More specifications for that requirement can be found in Section 4.2. A powerful aspect of the IAM service is that it provides Single-Sign-On across all integrated services. In that sense, users can log into different tools/systems by providing the same EBRAINS credentials. Finally, the IAM service provides means for services to connect to each other. Applications or services that will authenticate and authorise their users via the EBRAINS IAM service need to register for an OIDC client. A procedure is already available to the interested EBRAINS developers and a dedicated collab exists [11]. Components that used another OIDC provider in the past, like MitreID, need to migrate to the new OIDC provider by following the instructions that have already been presented [12].

4.1.4 Metadata Management

The EBRAINS Knowledge Graph is used as a multi-modal metadata store that combines information from different fields on brain research, data/models, software that exist in EBRAINS. Neuroscientists who wish to share their data in Findable, Accessible, Interoperable and Re-usable (FAIR) way can apply for user support to have their data and models curated by a group of specialised curators. In that way, they can make their data and models easy to be discovered and reused by other researchers. The Knowledge Graph also offers rich metadata annotation, an important aspect of finding using and re-using actual data and models. Researchers may be interested to know that they can identify their work and at the same time increase visibility of it, by mapping the data with a persistent identifier, for example a Digital Object Identifier (DOI). The procedure of integrating data and at the same time map the data to a DOI, consists of three steps, 1) request to submit data 2) curation team to review the submitting 3) curation team to accept the submission. For journal authors there is a variety of options that they can select of regarding when the data used in a publication will be publicly available. First option gives the opportunity to allow public visibility and access to the dataset -a DOI will be in place- before the paper has been submitted. Second option is to allow the public to view the metadata and not the actual data before the paper has been published. Last option is for neither the dataset nor the metadata to be visible to the public until the paper has been published. One feature of great importance is the new metadata models that are already released under the umbrella of the open Metadata Initiative for Neuroscience Data Structures (openMINDS) and are adopted by the EBRAINS Knowledge Graph. openMINDS gathers a set of metadata models describing heterogeneous neuroscience data. The migration from the old metadata models (MINDS, uniMINDS) to the new openMINDS metadata models (core, SANDS and controlledTerms) as well as the integration of further in-depth metadata models (e.g. for electrophysiology) into openMINDS have already started. For more information about openMINDS, you can refer to the main GitHub repository [14] and the press release [15].

Detailed documentation for developers that want to integrate with the EBRAINS Knowledge Graph can be found at [13].

8 https://www.keycloak.org/

4.1.5 Developer Services and Tools

EBRAINS, as a fully-fledged digital research infrastructure, provides a set of tools and services that support developers throughout the Software Development and Delivery lifecycle. These tools create a complete environment that can support all development activities, by providing all the required means to securely host part or all aspects of the development effort. This leverages the function of the development teams by facilitating their day-to-day operations. The available tools and services span the areas of source code repository hosting, build, test and deploy on the EBRAINS RI across different deployment environments. Public-facing APIs, Infrastructure as a Service, Platform as a Service and storage capabilities are also available. Finally, a support channel providing a dedicated ticketing system guarantees end-user satisfaction.

In the following sections, developers can find information on the available deployment types and how they can take advantage of middleware and APIs for their use case. These also describe where and how data can be stored for long-term storage purposes and how to request for frontline support to timely resolve any issues that may arise. A brief overview of the CI/CD system and the planned deployment environments is also provided.

4.1.5.1 UNICORE

UNICORE (Uniform Interface to Computing Resources[21]) offers a ready-to-run system, including client and server software. UNICORE makes distributed computing and data resources available in a seamless and secure way in intranets and the internet. As presented in the Architectural diagrams (Section 3.8.2), UNICORE consists of four components. The UNICORE/X server is the central component of a UNICORE site. It hosts services such as job submission, job management and storage access, and provides the bridge to the functionality of the target resources, e.g. batch systems or file systems [18]. UNICORE/X is deployed in each Fenix site. UNICORE TSI is a daemon running on the frontend of the target resource (e.g. a cluster login node). It provides a remote interface to the operating system, the batch system and the file system of the target resource. It is used by the UNICORE/X server to perform tasks on the target resource, such as submitting and monitoring jobs, handling data, managing directories etc. The TSI, which is also deployed in all Fenix sites, performs the work on behalf of UNICORE users [19]. The UNICORE Registry server provides information about available services to clients and other services. It is deployed only in one Fenix site. Last but not least, the UNICORE Workflow service provides advanced workflow processing capabilities using UNICORE resources. The Workflow service provides graphs of activities, submits and manages the execution of single UNICORE jobs. The Workflow service offers a REST API for workflow submission and management and uses an easy-to-understand workflow description syntax in JSON format [20]. EBRAINS users may configure a UNICORE/X server to run High-Performance Computing (HPC) jobs, or they can run Jupyter Notebooks that use UNICORE to submit HPC jobs. With that approach, authentication is taking care off and a mapping between users' credentials and HPC accounts is taking place, so only users or service accounts with HPC quotas can run HPC jobs in an HPC system.

4.1.5.2 Virtual Machine Services

OpenStack [22], [23] is used as Virtual Machine Service in EBRAINS. OpenStack is a free, open-standard, cloud computing platform that provides the foundation for building private or public Infrastructure-as-a-Service (IaaS) clouds. The tools that comprise the OpenStack platform handle the core cloud-computing services of compute, networking, storage and image services. Developers can create their own infrastructure of Virtual Machines (VMs) by creating their own network and router, and setting up ssh key pairs and security groups for the VMs. Other than VMs, OpenStack Platform provides virtual disks, virtual networking, object storage, optional GPU-passthrough. A significant number of EBRAINS components have already deployed their services on the OpenStack Platform (for more details see **Annex IV: EBRAINS RI Components** and **Section 3.4**). There are many ways to access OpenStack either by the Graphical User Interface (GUI), or by the Command Line Interface (CLI). A very practical and important thing that developers must remember is that loss of a VM is

possible and so they should take care to replicate the application and user data in backup volumes. Finally, snapshots of VMs can be used for backup mechanisms. For more backup mechanisms, please refer to Section 4.1.5.4.1.

4.1.5.3 Container Orchestration Platform

OpenShift is used as Container Orchestration Platform at EBRAINS. OpenShift is an open-source cloud development Platform as a service (PaaS), which enables the developers to develop and deploy applications on cloud infrastructure. A great number of EBRAINS components deploy their services in OpenShift (see **Annex IV: EBRAINS RI Components** and **Section 3.4**). It leverages the Kubernetes concept of pods, which are one or more containers deployed in one host. Different OpenShift versions from 3.x to 4.x give developers notable features like an easy way to do cluster autoscaling. Developers can also use load balancing methods and configure their applications in a way that, under exceptional circumstances like high CPUor memory usage, or even many requests, the application load can be distributed to more than one pod. In January 2021, an"The Virtual Brain (TVB)" EBRAINS component successfully load-tested running a MOOC (Massive Open Online Courses) for many users on OpenShift[9].

4.1.5.4 Archival Object Storage

Swift [24],[25] is a highly fault-tolerant object storage service that stores and retrieves unstructured data objects living inside containers. Users/researchers can store data that will be used as input to simulations running at an HPC system and/or store output data for future analysis. Swift Object Storage is optimised for capacity, reliability and availability. It is a long-term storage for large amount of static data which can be retrieved and updated. Usually, data stored in long-term storages expire a few months (3) after the end of a project's contract with the Infrastructure.

4.1.5.4.1 Backup practices

Backing up data is an important task that needs to be considered by application and service developers. In both Container Orchestration Platform and Virtual Machine Services deployments, developers need to create back up strategies for their data. Virtual Machines (VMs) can be captured in a snapshot for backup mechanism. User data and databases need to be backed up in a backup volume, not in root filesystem, and then copied to an OpenStack Swift Container. For application in OpenShift, backing up the entire project is a valid technique. Since, backing up data is not a one-time procedure, automating the process with scripts and playbooks is an essential part of managing service and/ or application. [26]

4.1.5.5 Container Image Registry

Since a great number of applications and services in EBRAINS are deployed using containerisation, there should be a place where container images are stored. It is not a good technique if different development teams store images to different registries, since the ability to tightly control images gets lost in that approach. Other than this, continuous deployment benefits from having a single point of truth where the latest image tags exist for different tools. Keeping track of new images per service, application can also make bulk of releases easier to identify and versioning of the product is not so complicated.

A third-party image registry, Harbor, has been set up for EBRAINS developers to fulfil that specification. Access to that image registry is restricted only to developers with EBRAINS accounts for pushing and pulling images. Both development and production environment has been set up with

[9] https://www.brainsimulation.org/bsw/zwei/events/single/6202--personalised-multi-scale-brain-simulation-virtual-course-from-dec-3-2020-to-feb-18-2021

Keycloak Open ID Connect integration. Developers need to log in via OIDC provider, place their EBRAINS credentials first and then create a project that can be public or private. [27]

4.1.5.6 EBRAINS Gitlab

EBRAINS features a private Gitlab installation (Gitlab community edition) that is available to all accredited EBRAINS members. GitLab is a complete DevOps platform, delivered as a single application. It offers all the required functionality to automate the entire DevOps life cycle, from planning to creation, build, verify, testing, deploying and monitoring of applications and services. Additionally, it also includes wiki, issue-tracking and CI/CD pipeline features. Gitlab runners are also available and enable modern CI/CD practices to almost all EBRAINS available deployment types.

Gitlab is available to all EBRAINS development teams and provides them with several options. These include just hosting only the source code repository or only the CI/CD pipelines for the DevOps activities, or even using it as the main platform to host all their development/DevOps activities. For further information on how EBRAINS Gitlab is intended to be used in the context of EBRAINS components integration efforts, refer to Section 4.2.3.

4.1.5.7 High Level Support Team

The High-Level Support Team (HLST) provides a full range of support services for EBRAINS, covering frontline support (including request for access to EBRAINS), in-depth support for tools and services, data, model and software curation support, and operations support. Users can submit support requests via email (support@ebrains.eu, support@humanbrainproject.eu) or web (ebrains.eu/support). Support tickets are managed using Zammad [28].

The HLST also coordinates and supports provision of documentation for EBRAINS tools and services. To ensure that tools and services offered by EBRAINS can be properly supported by the HLST with the resources available, the HLST will, in collaboration with TC, review any new tools and services proposed for inclusion in EBRAINS and make corresponding recommendations to the EBRAINS leadership.

4.1.5.8 Deployment Environments

In the preliminary EBRAINS Phase 1, some first requirements were communicated. EBRAINS Phase 1 aimed to streamline the procedure for integrating f a component into the EBRAINS RI. Lessons will be learned and integration of more components will be more straightforward in the future. One of the biggest challenges to be addressed is to create different environments for the components' services, tools and applications, from staging, to testing to delivery. The idea is to test components' code first with small tests, which developers write while coding the unit tests (Section 4.2.2). Once all tests have been passed, integration tests (Section 4.2.2 need to take place. In most cases, integration tests need to be constructed by each component's development team, the EBRAINS DevOps team or, in some cases, via acollaboration. To test integration of a component with other EBRAINS components, a safe environment for testing must be set up. All interfaces need to have, at least, a testing environment with production versions in place (a "staging" environment), so new versions of EBRAINS components can have their functionality tested (more details can be found in Section 4.2.3).

4.1.6 Runtime Services

Apart from "Developer Tools and Services" that support the development and delivery of EBRAINS components, a set of "Runtime Services" are available to facilitate and fulfil more operational and dynamic/ephemeral requirements.

In this section, developers can find information on how interactive computing capabilities can be enabled with JupyterLab, what their functionality may be under the EBRAINS umbrella (Section

4.1.6.1), and how EBRAINS users can access transparently the underlying Fenix infrastructure, submit HPC jobs and access the Object storage (Sections 4.1.6.3 and 4.1.6.4). Retrieving and submitting logging, monitoring and observability information in the context of EBRAINS-wide monitoring (Section 4.1.6.6), as well as a quick presentation of provenance in EBRAINS (Section 4.1.6.5) and ways through which a containerized approach in cases where specific use-cases require libraries or services to run on HPC systems (Section 4.1.6.2) are also available.

4.1.6.1 JupyterLab

In general, JupyterLab [29] is a web-based interactive development environment for Jupyter notebooks, code and data. JupyterLab configures and arranges the user interface to support a wide range of workflows in data science, scientific computing and machine learning. Jupyter notebook is an open-source web application that allows users to create and share documents that contain live code, equations, visualisations and narrative text. In EBRAINS, JupyterLab installation on top of OpenShift makes Jupyter notebooks one of the most promising tools for interactive computing involving EBRAINS developers, neuroscientists and other researchers. EBRAINS members who access the Collaboratory can share their work with other members, groups or units in the Consortium using Jupyter notebooks. Jupyter notebooks via the EBRAINS Collaboratory make integration with HPC systems and neuromorphic computer hardware quite easy. A load-balancing functionality needs to be in place, so JupyterLab instances can be spawned at different Fenix sites transparently from the developers' viewpoint. Developers/scientists will be able to run Jupyter notebooks with pre-installed software in their images for their work. Currently, discussions are being made in order to fit that requirement in the Platform.

4.1.6.2 Containerisation in HPC systems

In general, there is a variety of containerisation tools, like Docker, Sarus, Singularity and Shifter, that a developer can use in High-Performance Computing (HPC) systems. The most frequently used tool currently is Sarus, due to security-oriented methods on HPC systems, and its compatibility with Open Container Initiative (OCI) standards and the workload manager. Docker is not a preferred tool, because of security concerns. More information about Sarus and the OCI standards can be found at [32]. The Singularity tool can also be used as a containerisation tool. It is already supported be some HPC systems at EBRAINS. This tool allows Docker containers to be run natively, while also serving as a replacement for Docker in HPC environments. Like Sarus, Singularity can pull images from DockerHub, but the latter can also pull images from Singlularity Hub[10] too. With Sarus and Singularity, developers can run images in HPC systems that include libraries or services that will run directly to HPC environments [30], [31].

4.1.6.3 HPC Proxy

The HPC (High Performance Computing) Proxy service is a gateway for submitting Unicore (Section 4.1.5.1) jobs using Fenix service accounts by ERBAINS users with no Fenix resource allocation made beforehand. Soon, EBRAINS users will not need to submit a proposal for resource quota allocation in the Fenix infrastructure. EBRAINS will supply accounts loosely coupled with Fenix resource infrastructure. In that way, with the HPC Proxy Service, an EBRAINS user will only need to have an EBRAINS account to be able to run HPC jobs on the Fenix Infrastructure [33].

4.1.6.4 Object Storage Proxy

As a result of the combination of the two services mentioned previously (Image Service and HPC Proxy) there may be EBRAINS users with no Fenix account or resource quotas, interested to process imaging data that developers made publicly available within Fenix. The goal of this service will be

[10] https://singularity-hub.org/

to allow EBRAINS users with no Fenix account to Swift containers inside Fenix infrastructure. As a solution, a central Storage Service handling a central Quota on behalf of applications is investigated [34].

4.1.6.5 Provenance

Provenance is a broad term that is defined differently, depending on the context where it is applied. At EBRAINS, data provenance is associated with automatically capturing and automatically storing information that helps users to determine the derivation history of some data assets, methods and models. It is also strictly associated with reproducibility of experiments, meaning that users can tell which data or models was generated by who, in which environment, with what hardware configuration, etc. For now, automatically capturing of provenance metadata is related to computational workflows running in EBRAINS. Later steps regarding provenance will include more approaches other than computational workflows. The goal is to create an API, where provenance metadata will be captured and stored automatically. A Task Force has already been set up,in which key people can help to identify the requirements that need to be met for automatically capturing and storing metadata, and to sketch a plan for addressing provenance in EBRAINS. As a first step, the Task Force tried to identify which EBRAINS components' use cases will benefit from provenance. Use cases provided some critical knowledge regarding what kind of information each of these would like to capture and a minimal metadata schema has been formed, based on the OpenMINDS Core. After automatically capturing the provenance metadata, those metadata are automatically stored locally. Here, "locally" is intended to stretch where the computation executed, like in a HPC system. There are a lot of cases where provenance metadata should be treated like any other output files. We can automate storage of metadata inside the Knowledge Graph for use cases that will benefit from that, without imposing this on all Use cases. Rather, we will take advantage of that automatic procedure when there is some value in having that kind of information inside the Knowledge Graph.

4.1.6.6 Monitoring

A critical need for the EBRAINS RI is to monitor the integrated software components in order to detect early any problems that may arise, assess performance by measuring the number of recourses the various applications/services utilize, identify usage patterns, pinpoint workload spikes and even detect malicious behaviour. Monitoring involves the collection of the underlying infrastructure's metrics information, consumption and analysis of application logs, as well as service availability information. Achieving a multi-level global monitoring view is almost impossible and therefore monitoring must be done at different levels. These levels are further detailed below.

- **Service level monitoring.** It provides an external view of the services and applications by checking whether the services are available to the users or what is the response time. For this purpose, Uptime Robot is used to get alerts when EBRAINS services are unavailable and to have statistics on the platform's services global availability [36].
- **Application Performance Monitoring (APM).** APM allows to have an internal view on how an application works. It helps to track and diagnose performance issues. On multi-tier web architectures, it provides tools to track user activity from a webpage or URL, from inside the application code targeting and tracing even the database queries.
- **Operating System's level monitoring.** This level of monitoring is done at the Operating System (OS) regime 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 data shippers (monitoring "agents") that are installed on each component's server or VM. These agents (such as Telegraf, syslog, beats, etc.) provide the performance metrics mentioned above and thus offer monitoring insights at the operating system level. In EBRAINS, various technologies stacks are currently used for OS level monitoring, including Prometheus with Grafana Dashboards and ELK stack.

4.2 Developers' Guidelines

In the present section, the "Technical Guidelines" for a component's integration to the EBRAINS RI are presented. These guidelines will greatly help to create and maintain a streamlined,guided procedure for the integration of components into the EBRAINS RI. The aim is to offer standard, best-practice tactics to achieve smooth integration of applications/services/software to EBRAINS, featuring a minimum set of common, but also critical requirements that are intended to raise the technical foundation bar across the SGA3 board. The guidelines may be adapted on a case-by-case basis depending on the actual needs of each component/service on the integration path. Developers are encouraged to follow these guidelines from early project phases onwards, to ease the transition to higher maturity levels, and to reduce the overall effort of developing, maintaining and delivering high-quality software.

EBRAINS components must comply with the requirements set out in the EBRAINS RI Architecture. Detailed information about EBRAINS RI Architecture can be found in Section 3. Components also need to adhere to the Software Quality (Section 4.2.2) and Delivery (Section 4.2.3) guidelines compiled respectively by the Software Quality and Software Delivery Working Groups, in close collaboration with EBRAINS Technical Coordination (TC). For the original output of the two Working Groups, one can refer to the appropriate **Annex IX: Software Quality Guidelines** and **Annex X: Software Delivery Guidelines.**

The guidelines will be updated throughoutSGA3, based on EBRAINS TC directions, feedback from EBRAINS Phase 1 (see Section 4.2.1) and subsequent Phases, and directions from the various Working Groups. The guidelines are forged through the intense activities across the SGA3 board, but they equally apply to both internal development teams of EBRAINS which actively participate in the development of the different tools and services that are currently part of the EBRAINS RI, and to potential external development teams which may wish to integrate their applications/services with the EBRAINS RI.

4.2.1 Phase 1 guidelines

EBRAINS Phase 1 is intended to successfully complete a number of Proof of Concepts (POCs) for EBRAINS Research Infrastructure. Arbor, NEST, SimVisSuite and The Virtual Brain (TVB) were selected as the first candidate components to be integrated into EBRAINS, because of their maturity and readiness levels; their early selection does not signify in any way their level of importance as EBRAINS components. This procedure will streamline the integration of components, lessons will be learned, and blocking points will be revised and even handled better in the second round of integration. Integration guidelines for EBRAINS Phase 1 have already presented to the components, further discussed and work has started from their side. Guidelines [39] were broken into three (3) levels, "Required", "Welcomed" and "Suggestions". With respect to the required guidelines, the ordering is random and does not imply that one requirement is more important than others. It is very critical for component developers to update continuously their components' dedicated wiki pages in the Technical Coordination (TC) collab, so TC can have at any time the most upto-date information. That said, the documentation section for all kinds of audiences (users, developers, admins, contributors) in the TC collab will need to be available to TC. Components must be deployed or built in the case of libraries in the main FenixFenix site (CSCS) and in JSC as a secondary option. Components must be deployed using one of the deployment types described previously: in a Container Orchestration Platform, as Virtual Machine services, in Collaboratory Notebooks, as a library running in High Performance Computing (HPC) system, or as a container running in an HPC system. In cases where container images will be produced, they must be pushed in Harbor, the EBRAINS image registry. It is important for applications to authenticate and authorise their users via the central Authentication and Authorisation Identity Access Management (IAM) that EBRAINS offers. For more, please refer to the Authentication and Authorisation section. Continuous integration procedures for testing and building will be set up and will be triggered by private repo-mirrors setup by TC on the EBRAINS Gitlab instance (or, optionally, the component's public project hosted on EBRAINS Gitlab, if they choose that; otherwise the private repo-mirror must receive code pushed from the component's own external project). For more details, see Section 4.2.3 about software

delivery. In the few edge-cases (like NMC), where code cannot be submitted to testing within the EBRAINS Gitlab CI pipeline in the usual way, external testing results must be submitted to TC in an open-standard format to assist project-status monitoring in a best-effort manner.

Different deployment environments must be in place for testing, staging and production. With respect to integration with other interfaces, testing environments will need to be set up and functionality tested there, in order to not interfere with the production one. Last, but not least, the appropriate SGA3 and EBRAINS funding acknowledgements must be in place. The conclusions and results of EBRAINS Phase 1, will have a beneficial impact on EBRAINS Phase 2 and possible architectural adaptations for EBRAINS RI.

4.2.2 Software Quality guidelines

4.2.2.1 Introduction

Software quality (SQ) across all EBRAINS software is currently monitored by the EBRAINS Software Quality Working Group (ESQ-WG) which acts as a central focus point for discussions on the guidelines for assuring software quality. The ESQ-WG operates in close collaboration with EBRAINS TC to ensure alignment with all involved stakeholders. The following sections are heavily influenced or are the direct product of the ESQ-WG document (Annex IX: Software Quality Guidelines).

Required SQ aspects and possible mechanisms are described to be implemented by tool/software developer groups. Essentially, guidance is provided on how to apply existing software quality standards and procedures in the context of EBRAINS and what EBRAINS specific conventions need to be followed.

Developers are encouraged to follow these guidelines from early project phases onwards, to ease the transition to higher maturity levels, and to reduce the overall effort of developing and maintaining high quality software. Software quality aspects fall into different categories which are defined in ISO-250101 [37] as:

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

These may be achieved through different mechanisms and conventions, some of which are described in these guidelines as examples and to shed light on the possible reasoning for or against a particular tool or mechanism.

In the following sections, we first introduce some basic development practices for EBRAINS developers as well as some requirements to follow. The requirements are separated into different categories based on their priority level (must, should or may). Then, in the next two sections, we continue with practices that developers and teams should follow in order to test their software during each and every phase of the Software Development Life Cycle (SDLC), and how to approach the development of the technical documentation. Documentation for the code, as well as system requirements and software dependencies, must accompany the software and provide all the required information for developers, users and operators.

To ensure software quality levels EBRAINS-wide and provide developers with a reference point regarding the SQ-guidelines compliance andan intuitive overview of which aspects of software quality may need consideration, a checklist of required actions/items will be formed by EBRAINS TC

in due course and provided to EBRAINS component owners. In the context of ESQ-WG, an indicative checklist was created to provide a quick and brief overview; this can be found in the Annex IX: Software Quality Guidelines. The exact requirement levels of these points are still being discussed; when finalised, these will be communicated to EBRAINS component owners.

4.2.2.2 Requirements and Best practices for Developers

After Phase 1, the requirements and guidelines are expected to be more easily categorised, along the following lines:

1) Known Requirements: Things that are clearly necessary to require of EBRAINS contributors.
2) Expected Requirements: Things most likely necessary to require of EBRAINS contributors, yet to be officially confirmed.
3) Projected Requirements: Things predicted to be necessary to require of EBRAINS contributors at a later phase.
4) 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.
5) Best Practices ("nice to haves"): Things that are very helpful but are so optional that even the term "recommendation" would be too strong.

In terms of the familiar key words as known from the IETF RFCs [38] the above points correspond like this:

- 1, 2, 3 (requirements):
  - "MUST"
- 4 (recommendations):
  - "SHOULD"
- 5 (best practices):
  - belongs to "MAY", but with a hint of "GREAT IF YOU CAN THOUGH"

For practical reasons, as well as to allow some guidelines to clarify themselves as an emergent property of Phase 1, such categorisation is not yet done rigorously. Many such things are, for now, expressed as general guidelines. One aspect that is already clear is that, although requirements and recommendations will be evolved continuously by both the working groups and the tech-coordination team, definitions and details of category 5 (best practices) will predominantly appear only in the working group documents (and will at most be referred to superficially via links from the TC Deliverables). This is in order to keep the TC Deliverables focused on “Requirements”, without getting overwhelmed with best practice advice.

4.2.2.3 Testing

EBRAINS RI features software components of different characteristics, scope, complexity, maturity and role. Different testing regimes are both understandable and efficient. However, lack of adequate testing processes or non-enforcement of testing process are not. TC explicitly considers EBRAINS components to be delivered as fully tested components. The goal is to ensure adequate testing processes are in place, enforced and monitored. Testing methodologies for all components must be documented and maintained. In addition to “in-house” testing of components, EBRAINS-wide testing will also be put in place. In cases where “in-house” testing results must be submitted centrally to TC, these will be provided in an open-standard format at a well-known endpoint to assist project-status monitoring in a best-effort manner. Where needed, specific guidelines will be provided by the TC to improve existing processes or introduce new (e.g. unit, acceptance, end-to-end(E2E) testing) on a case-by-case basis. Also, where needed, common testing environments (site-specific or multi-site) will be designated (especially EBRAINS-wide integration testing).

For any non-trivial software project, testing and similar software quality techniques such as static analysis are essential for ensuring software quality and maintainability. During Phase 1, no testing is strictly required of the "pilot component" owners other than what is already employed by those teams, motivated by their own quality-requirements. For that reason and because software testing is such a vast science, there will not be in-depth descriptions and best-practise details in this Deliverable. This is to keep the content here focused on EBRAINS-specific requirements and goals, and the aspects of testing which relate directly to those.

To give an overview of the scope and scale of potential testing strategies, a concisely worded list of testing types is provided in the following sections. Some of them are described in more detail in the ESQ-WG document (Annex IX: Software Quality Guidelines), and details and best-practice suggestions for all of them can easily be found.

The more (and the higher-quality) testing that is created and run by the component owners the better; specifically on the level of unit-tests, feature-tests, and component-level integration tests. That also applies to other QA techniques like static analysis, linting, literate-code, etc. Additionally, the more that all types of testing above those levels can be implemented and run within the central EBRAINS CI pipelines the better. As nearly all tests can ultimately be automated, this should happen wherever possible to minimise the operational overhead.

A common strategy is also to structure multiple pipelines running on different schedules, often in parallel. One example is to have the following two pipelines in parallel:

- A "continuous delivery" pipeline triggered by every commit as "pass/fail" gatekeeper before delivery/deployment, with the steps:
  - Build
  - Unit tests
  - Integration tests, E2E tests, API/interface tests, acceptance tests, etc.
  - Security tests
  - Deploy & update pipeline statuses on relevant Dashboards
- A "discovery" pipeline triggered continuously whenever there are changes, with the steps:
  - Lint
  - Code quality
  - Performance testing
  - Load testing
  - Update key performance indicators (KPIs) to dashboards

A quality product, in order to be integrated in EBRAINS RI, needs to be tested hierarchically aiming to identify any possible bugs or defects. The following diagram shows how testing can be potentially divided and applied on EBRAINS RI.

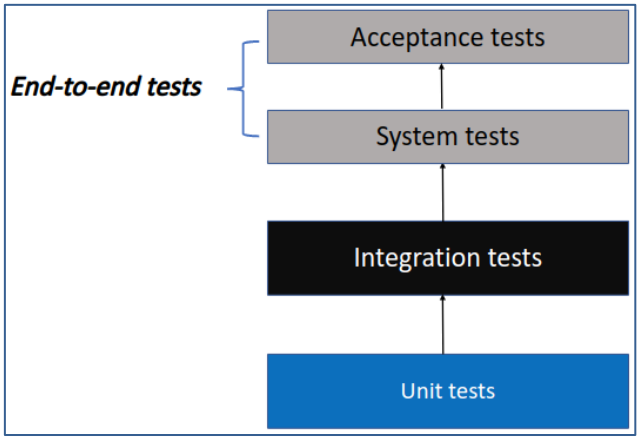

Figure 13: Different levels of testing in software tools and their hierarchy

Initially, unit-tests are written by developers in parallel with the whole development/coding process. There should be at least one test for each individual module and unit of the source code in order to determine that they perform as expected. It is necessary then to run integration tests which verify the overall functionality of the system by testing each new module which is to be integrated into the main software package. Integration tests ensure that all new components and services work together properly and do not affect the expected behaviour of previously developed code. These integration tests can be automated to run on every change made to the software (continuous integration (CI) testing) in parallel with end-to-end tests or even some component tests. Component owners should be able to set up several test suits and run them on top of EBRAINS RI, or their local machines or both.

Finally, end-to-end testing is needed to assess the fully integrated software including external peripherals. These tests can be divided into two testing subcategories:

- System tests,
- Acceptance tests.

System tests run in order to validate the complete and fully integrated software product by checking how components interact with one another (as in integration testing) and with the system as a whole. When the product is ready for production the members of user acceptance panel (see Section 2.4.4) will test it through acceptance tests.

Software that passes all testing stages and fulfils all EBRAINS "MUST" requirements (usually also enforced by automated tests) will be allowed to proceed to delivery/deployment in the CI pipeline. On a case-by-case basis (after discussion with TC), some specific components may also be required to fulfil some EBRAINS "SHOULD" recommendations. Some components may have requirements which can only be tested manually; in which case, after passing all automated tests, the pipeline will need to proceed to delivery and raise a flag that final acceptance is pending the results of those manual tests (probably run in the staging environment). Either way, these requirements and recommendations will be based on the respective ESQ-WG guidelines and collaboration with the TC team for specific decisions and general clarifications.

4.2.2.3.1 Unit Testing

Unit-tests in EBRAINS refer to testing each individual module and unit of the source code of an EBRAINS component in order to determine that they perform as expected. These tests are written by developers in parallel with the development/coding process of the software. A unit may be an individual function, method, procedure, module or object. Test scenarios should be designed to assess a specific pipeline of test cases and all cases must be as simple as possible. In the case of test cases that require data insertion, developers should make sure to provide the necessary mock datasets that are needed.

4.2.2.3.2 Integration Testing

It is then necessary to run integration tests which verify the overall functionality of the system, by testing each new module which is to be integrated into the EBRAINS RI. This step ensures that all new components and services function together properly and individual units are combined and tested as a group. Therefore, the delivery process should include running integration tests where applicable. For instance, an API intended for Collab users, should be deployed followed by an integration test consisting of running a notebook with calls against the API.

These tests should be monitored, and respective results need to be correlated with deployment in the staging environment for identifying what changes may have destabilised the environment. A dedicated monitoring system will be set up and provided to EBRAINS components development teams with dedicated endpoints for the submission of tests results and respective visualisation dashboard for taking full control over the testing process. Additionally, when an issue is found and fixed, it is important to ensure that the issue does not reappear. A regression test should be added to the test-suite for that explicitly known issue (as opposed to the speculative or predictive tests added to accompany new features). In terms of scope, these may be unit tests and/or integration tests, but

the important distinguishing factor is the "regression" part, indicating this should insure against that "regression" in the future.

4.2.2.3.3 End-to-End Testing

As mentioned above, end-to-end (e2e) testing can be divided into two categories: system tests and acceptance tests. Since all the modules has been integrated to the system and tested through integration tests, system tests must be conducted ensure all functionalities. It is very important to complete a full test cycle; system testing is the stage where this is done.

4.2.2.3.4 System testing

System testing is going to be performed to ensure every module and/or service within the EBRAINS RI is functioning properly. System performance and stability can also be tested here under different loads or test scenarios. These scenarios will be designed to simulate user behaviour to make sure the system works as expected at scale.

Security checks and performance tests are also important for the stability of the network and its “defence” against external injections. Once the security requirements are satisfied, the product should ready, after some last performance tests: as part of this, stress tests should also be applied by injecting custom faults to code paths or make the various system components operate at their limits. Crucial errors can be found, thus leading to the optimisation of the overall error-handling and scalability-assessment capabilities as well. If the system testing scenarios are successful and stress tests are passed “green”, the last step is to ensure that the application works as expected from the actual end users’ point of view. To this end, User Acceptance Testing (UATs) is performed. The end users of EBRAINS components come from a variety of different scientific fields and therefore specific user-actions scenarios should be tested to satisfy the expected scientific needs.

4.2.2.3.5 Test-driven development methodologies

Test-driven development (TDD) methodologies can be described as "code to the tests", rather than "test the code". The famous sequence is "red" (write placeholder stub-code and a tightly scoped failing test of the goal of that future code), "green" (implement the code enough to make the test pass), "refactor" (clean-up and optimise the code without breaking the passing-test). Behaviour-driven development (BDD) includes "executable specifications" so that non-developers can write the actual tests in business-terms rather than going through developer-bias and filters translating specs to test “code”. Acceptance-test-driven development (ATDD) involves a triangle including stakeholders (product manager, user advocates), testers (product owner, test engineers), and development (engineers) to reach consensus on “acceptance tests”, and to then verify the tests (usually E2E) after implementation within the context of "acceptance testing".

4.2.2.3.6 Other/Manual Testing

If the software has a (graphical) user interface (UI) or needs to be manually tested, it is important to have tests executed by a test engineer, if full automation is not possible. Often, manual tests are executed in the development environment when a new release is ready to move to the staging. During manual testing, the tester uses the application as an end-user. This also helps to write the correct test cases.

There are different types of manual tests:

- Smoke testing. Smoke testing is high-level testing used to check the primary objectives and critical defects (“First look” testing).
- Exploratory testing. Testing without test cases or structured guidelines. During exploratory testing, testers are free to take initiative.

- Cross-browser testing. Test applications in different devices and browsers or OS's. According to the software quality guidelines it is important to support different devices, browsers or operating systems in case it is applicable.
- Acceptance testing. Formal testing with respect to user needs, requirements, and business processes conducted to determine, whether a system satisfies the acceptance criteria and to enable users, customers or another authorized entity to determine whether to accept the system.
- Beta testing. This kind of testing is performed by the end-users (external user panel) in a "real" environment and it can be considered as a form of external User Acceptance Testing. Direct feedback from external end-users is a major advantage of Beta Testing.
- Localization Testing: If the software targets a global user community, it is important to include also localisation testing. The main idea is to check appropriate linguistic and cultural aspects for a particular locale.

4.2.2.4 Documentation

Documentation is an essential aspect of high-quality software. It allows users, developers, administrators and testers to get the required information to successfully use, maintain and extend the software tools.

In general, many aspects of writing documentation are very similar to writing software itself. Usually, there is a process of generating the output documents, dependencies between descriptions in different parts, and there may be tests and checks, e.g. for referential integrity, language linting, documentation style checks, etc.; all aspects pertinent for the actual code. To collect these best practices for documentation and to facilitate sharing of this knowledge between the EBRAINS software components, Technical Coordination announced the "Winter of Documentation" (one of the EBRAINS RI seasons, see Annex II: TC planning for more information). This particular season, dedicated to the maintainability and completeness of software documentation, was started with a kick-off at the CodeJam#11 and the results will form a separate EBRAINS Documentation Guidelines document.

Most of the time, documentation is the first point of contact for new users and basis of this primary decision to use a tool or search for another one. During the actual usage of the tools the documentation then requires to address the needs of various target groups to supply required information for the tasks at hand. The different groups of users, developers and operators each show a difference in experience, required depth of descriptions and language. Even within these groups there may be a huge variety that justifies further specialization of the documentation.

EBRAINS components must provide user-level documentation, developer-level documentation and maintenance/operations documentation. Fundamental requirements for the aforementioned documentation levels are detailed in the following paragraphs.

4.2.2.4.1 User-level Documentation

The requirements for user-level documentation, the interrelation between documentation of different tools and services, and interactions with support channels, workshops and classroom education are a very broad topic that is to be detailed in the separate EBRAINS Documentation Guidelines document. In general:

- User-level documentation must enable end-users to find, install and use the software/tool and understand its operations and fundamental assumptions.

4.2.2.4.2 Developer Documentation

Documentation primarily directed towards developers aims to maintain good structural overview over the code base, preserve expert knowledge, lower the threshold for new developers and define project management structures and conventions.

As the goal of this is to ensure long-term sustainability of the software, it draws from all other sections of these SQ guidelines. The following points relate to corresponding checklist items that should be met.

- Software should have an internal architecture view of the software.
- Documented coding guidelines and stick to conventions.
- Documented integration process (e.g. automated tests, Code Review (CR), approval processes).
- Projects should define a code-of-conduct and requirements for contributing (e.g. Contributor License Agreement (CLA)).
- Project management conventions should be documented (development practices & processes, code review guidelines).

4.2.2.4.3 Maintenance/Operations Documentation

Exact requirements for operations documentation are to be discussed between the relevant EBRAINS groups and Technical Coordination. They are expected to be continuously refined with feedback from administrator teams at the different deployment sites and for the different deployment types (VMs, Container Orchestration, HPC centres, Collaboratory, etc.). For additional requirements refer to the Section 4.2.3. In general:

- Maintenance/Operations Documentation must enable users without internal knowledge of the tool to deploy and provide the tool and to maintain basic development structures (building documentation, updating packages/dependencies, install and deploy containers, etc.)

4.2.3 Software Delivery guidelines

Software delivery (SD) across EBRAINS is currently monitored by the EBRAINS Software Delivery Working Group (ESD-WG), which acts as a central point of focus for discussions on the guidelines for a common set of software delivery practices. The ESD-WG operates in close collaboration with EBRAINS TC to ensure alignment with all involved stakeholders and communicate an EBRAINS-wide delivery process, along with a minimum set of required actions that need to be accomplished by EBRAINS RI integrated components. The following sections are influenced by the ESD-WG activities (see Annex X: Software Delivery Guidelines).

EBRAINS RI is a complex e-infrastructure and relies upon federated, multi-site, underlying 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.

In the software delivery pilots (Section 4.2.3.1) that were conducted during EBRAINS Phase 1, a minimum subset of the SD guidelines were put to the test. Current individual (component specific) deployments are efficient (fast, bottom-up) but difficult to manage at scale.

To sustainably operate EBRAINS-wide, we need to gradually move to a centrally coordinated and horizontal deployment process with cross-site tracking of deployed versions (production & experimental, site-specific, multi-site and deployment type). We need to move towards a shared delivery schedule for new releases with common requirements regarding planning, dependency management, downtime, security assessment, versioning and testing.

In the following sections, the conception of the Product Delivery flow across EBRAINS will be presented. It was formed as a product of the discussions in the context of ESD-WG as well as the general overview EBRAINS TC gained through the Phase 1 Software Delivery pilots. The guidelines are to be applied rigorously during Phase 2 and subsequent phases and are expected to be further honed based on the interaction with a broader user base.

4.2.3.1 Software delivery pilots

During EBRAINS Phase 1, certain software delivery pilots were conducted to collectively gain a better understanding of the current component-specific individually managed deployments, identify requirements, test procedures and lay the foundation for an EBRAINS-wide product delivery scheme.

Current individual (component specific) deployments are efficient (fast and bottom-up) but difficult to manage at scale. The Arbor, NEST, SimVisSuite and The Virtual Brain (TVB) components were selected as the first group of EBRAINS integration activities due to to their level of maturity and technical readiness.

These components span various application categories and more specifically simulation engines with various deployment types (VMs, containers, Jupyter notebooks), desktop tools providing visualisation services through containerised HPC deployments, as well as web applications featuring scalable deployments over OpenShift. Different set of requirements and issues were identified by the participating development teams during the application of a minimal subset of SD guidelines that were provided by EBRAINS TC during Phase1.

Although different types of components tended to have different requirements and different development teams face different delivery issues, a common set of requirements was identified regarding deployment types, planning, dependency management, downtime, security assessment of new releases, versioning, operation in different service environments and testing during the product delivery flow.

By considering this common set of requirements, widely accepted best practices in operating in such a large-scale environment as EBRAINS, case-dependent features, and technical debt, the conception of SD guidelines for EBRAINS was achieved and are outlined in the following sections.

4.2.3.2 Deployment types

Contributions can be deployed for running as services or delivered to be sourced for use within services (either long-running or ephemeral in both cases) in the following forms:

- VMs (OpenStack)
  - Orchestration recipes/playbooks & VM metadata (Ansible recommended but not mandated, pre-built VM images possibly acceptable on a case-by-case basis).
- Containers (OpenShift, Sarus/Singularity/Shifter for HPC)
  - Helm charts (source, built).
  - OpenShift templates are also acceptable although Helm charts are preferred (due to release-versioning, more sophisticated capabilities, more expressive syntax, and no intrinsic vendor-locking).
  - LXC-style “system containers” may also be accepted as more efficient and safer equivalents to VMs, on a case-by-case basis.
- Software bundles
  - HPC/NMC modules.
  - OS packages (possibly accepted for cloud-deployment on a case-by-case basis).
- Scripts
  - Notebooks (JupyterLab for running in JupyterHub).

Contributions can be delivered as downloadable products in the following forms:

- Executables (CLI, graphical [desktop/mobile], IoT).
- Libs for facilitating programmatic access by external developers to public APIs.

4.2.3.3 Running Service Environments for delivery

There will be three environments used for progressing the infrastructure and its components from development to production: “Testing”, “Staging”, and “Production”.

4.2.3.3.1 Testing

This is an internal “continuous deployment” destination for anything that passes through static analysis checks and then unit-testing in continuous integration pipelines.

There is no need for sophisticated “continuous deployment strategies” (like canaries, blue/green, etc) here because it is a private testing environment.

Here integration testing happens (p2p and horizontal), then e2e testing, then system/solution testing (including eventually performance, stability under load, etc), then any automated acceptance testing.

System testing will include testing for components’ use of (or integration with EBRAINS-provided middleware for) standard protocols, data-formats, channels, namespaces and resources, and APIs provided for control by supervisor-processes. Although this will be permissive, especially in the early phases, it will be strict enough to ensure efficiency and ease-of-maintenance at the infrastructure-wide scale. A particular goal with respect to interactions between components is to avoid a combinatorial explosion of *ad-hoc* interactions, potential incompatibilities, assumptions leading to race-conditions, tight coupling leading to action-at-a-distance, and so on. Here there will be a clear correlation of recommendations with “warnings” and requirements with “failures” in tests.

4.2.3.3.2 Staging

This is a private continuous deployment destination for anything that passes through the Testing environment, a private incubating buffer zone before final release of components.

Some “continuous deployment strategies” could be employed here in order to keep the environment as production-like as possible, but, in most cases, it would still be overkill here too, as it is not a public-facing environment, and the overhead of using such “strategies” would therefore be a lot less justified than if we were doing full “CD-to-production”.

Here manual user-experimentation and any manual acceptance testing is done.

4.2.3.3.3 Production

After passing manual acceptance testing and a period of stability during real-world-like usage in Staging, code is continuously delivered (not continuously deployed) and packaged in versioned form with dependencies for manual staged deployments to Production. Manual production deployments ensure that all packages/services/components are well tested against each other (thus dependencies are managed) and deployed in sync in pre-defined release cycles.

Note, that dependency management will be performed on a case-by-case basis, depending on each deployment type and platform. Work is already underway to mitigate tight dependency risks between components. PoCs for headless testing of Collaboratory notebooks, formalising official Collaboratory images creation process and managing dependencies for HPC-deployed services are some of the initial actions taken to manage the dependency issues.

Even though it will happen manually, where possible, things will be deployed in rolling form (similar to blue-green, with an elegant tailing-off of old-versioned sessions, possibility of rolling back during windows) to aim for zero-downtime seamless upgrades. When deeply-invasive changes are required, maintenance windows will be scheduled, even as a speculative measure, in case there is an unexpected outage.

Using common terminology among distribution-vendors, EBRAINS RI will be a “rolling release” of packages (and canonical running instances thereof), not a “stable release”. Therefore, although all

components will be strictly versioned and manually deployed into production, the EBRAINS infrastructure as a whole will not have “feature-freeze” periods in order to stabilise on set “EBRAINS Versions”. This difference is analogous to the difference between the “Debian GNU/Linux” distribution’s “Testing” release (a rolling release) versus their “Stable” release (versioned, only bug-fixes accepted to each still-supported release). In the future, if requirements change enough, a “stable” release may become appropriate in addition to a rolling release, in which case, it will be considered and decided then. Note that the aforementioned “testing versus stable” comparison is commonly considered to be more of a “stable versus extremely stable” comparison in reality. The main factor, that might influence decisions about this, would most likely be the overhead caused by “dependency sliding-window" maintenance and compatibility-testing churn in a rolling release (not a problem in a stable release).

4.2.3.4 Product delivery

There will be a four-phase release flow for downloadable products contributed by component-owners: “Static analysis and unit-testing", “Bundling/packaging”, “e2e/solution testing” (where appropriate), “Upload to registry”.

- Static analysis checks and unit-testing.
- Bundling for appropriate platform (use universal portable-build frameworks like Packer to avoid duplication of effort and build-logic wherever possible).
- e2e/solution testing (automated & manual where appropriate) against related services if applicable.
- Upload to public download-registry.

4.2.3.5 Development environment

For developers who wish to create their project in EBRAINS Gitlab (and those who wish to migrate to it), we provide the full EBRAINS Gitlab SCM platform, including the Gitlab Continuous Integration service-chain.

For those who wish to setup or retain their project independently elsewhere, as long as some standard quality requirements are met (including having good unit-test coverage, either manually or as part of their own CI setup for testing), this is OK. It does mean that an extra step is needed to connect the result of their development process to the EBRAINS CI/CD system for (authoritative) unit-testing, integration-testing and delivery/deployment. They must provide a git repository which pushes code to a private mirrored bare git-repo (not a full mirrored-project) on the EBRAINS Gitlab. Whether the repo used for that is their main repo or just a conduit for translation from their source repo (perhaps a non-git repo like Subversion, etc) does not matter as long as it is used to push “release-ready” (and version-tagged) code to our mirror-repo to trigger our CI/CD runners. That push could happen either manually or automatically from a specific branch via a post-receive hook.

For projects hosted at EBRAINS Gitlab and for independently hosted projects pushing release-ready code to a private mirror-repo at EBRAINS Gitlab), each project will need to have a dedicated separate git-repo at EBRAINS Gitlab which hosts only the Gitlab CI configuration file(s) and the original git-repo in a subdirectory as a “git submodule”. This “wrapper repo” should be editable by EBRAINS TC and will be where the integration CI/CD flow is triggered from. This will mean that the CI configuration can be centrally or collaboratively maintained, while keeping the main repo editable only by the component owners (whether locally or elsewhere).

This means that the variations of development workflow “integration vs. Autonomy" can be:

1) Wrapper CI-repo and source repo entirely hosted at EBRAINS Gitlab.
2) Wrapper CI-repo and private git repo mirror at EBRAINS Gitlab, source git repo hosted elsewhere (e.g. GitHub using git-flow), pushing to the private mirror-repo.

3) Wrapper CI-repo and private git repo mirror at EBRAINS Gitlab, source git repo hosted elsewhere (e.g. self-hosted using another workflow like changesets with Gerrit), pushing to the private mirror-repo.
4) Wrapper CI-repo and private git repo mirror at EBRAINS Gitlab, source non-git repo and other workflow hosted elsewhere, translating through a conduit git repo to push to the private mirror-repo.
5) Some rare special cases will unavoidably have their own CI/CD pipeline entirely for private deployments (e.g. Neuromorphic systems which are "under the EBRAINS umbrella" but are effectively autonomous hardware and software silos). In order to avoid the unnecessary burden of new workflows there will only be a requirement for access to those separate CI systems, enough to be able to pull (or receive pushed) pipeline results into central EBRAINS development status dashboards. This will also be used to cover high-level "dependency" issues like certain engines requiring certain releases of NMC platform code to fully function.

Components which develop multi-part vertical stacks (e.g. a web-interface, a REST API, a back end dynamically-linked executable engine, and the software-library providing the core logic for the engine) will be encouraged (but not forced) to provide separate repos for each such horizontal domain within their project. Where this can happen, it will help with granular testing, debugging, packaging and dependency-tracking for EBRAINS maintainers, and will also usually help component-owners approach their solution with horizontal separation-of-concerns as a goal, rather than monolithic siloisation. Where this is possible and feasible, in combination with open licensing, it will in-turn open the components to more opportunities for cross-pollination of ideas, easier interoperation, and synergistic evolution.

All components must be coherently versioned (and releases tagged accordingly) and will be encouraged - but not forced - to use "semantic versioning" for consistency with the rest of the infrastructure.

With this "decoupled by default" EBRAINS integration/delivery workflow, maximum developer-flexibility is facilitated. This means:

1) Any requirements of "developer best practices" can be incremented over time almost purely in accordance with increasing infrastructure-wide quality thresholds, rather than in order to accommodate inflexibility/monoculturalism in the infrastructure.
2) It provides the most gradual learning-curve and least overhead for onboarding developers during initial phases.
3) Even though there are volumes and volumes of "best practices" which can be suggested to developers (and which they can already find easily when they want to search for them) the decoupling means that much of it is not a "hard blocker" for integration and/or onboarding, so all that information can be provided through supporting documentation and links, without overwhelming required-reading for component owners (and Deliverables).

4.2.3.6 Delivery-First Principles

Because most of the components contributing to EBRAINS are large and developed by teams with strong vertical foci of their own, EBRAINS as an infrastructure needs to encourage and focus on some key best-practice concepts to horizontally counter this momentum, and to guarantee the ongoing adaptability and resilience of the overall infrastructure. This is above and beyond the huge number of typical best-practices usually recommended for general code-quality reasons.

1) Automation
2) Integration
3) Deduplication
4) Decoupling
5) Composition

6) Maintainability

4.3 Quality assurance

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. b) EBRAINS platform services are continuously monitored in terms of their availability and reliability, to mitigate potential risks and ensure that the respective service level agreements (SLAs) are met.

Quality assurance procedures have been established by EBRAINS TC in order to coordinate the integration process and effectively tackle the emerging issues. These procedures are expressed through dedicated series of meetings and methods which are the formally scheduled "TC Weeklies" and "TC Task Force" meetings described in Annex I: Technical Coordination, and regular *ad-hoc* "debrief" meetings for synchronising about KPI outputs (see Section 4.3.1).

Quality mechanisms are also in place.These are described In Section 4.1.2, in particular the Kanban boards provided by Gitlab and dedicated wiki pages provided by the Collaboratory.

Metrics/KPIs for monitoring and assuring the quality of the EBRAINS RI platform specifically for the integration process (that is monitored by EBRAINS TC) are detailed in the KPIs subsection (Section 4.3.1). The aforementioned KPIs are the aggregation result of a number of component specific indicators that are averaged and correlated.

These indicators will be the product of the Technical Coordination and the established Architecture, Software Quality, Software Delivery, and Security Working Groups. Their refinement is ongoing and they will be updated in subsequent versions of this Deliverable.

4.3.1 Software Quality Guidelines and Indicators

These comprise of basic criteria for all components including, but not limited to:

- Documentation (levels, standards)
- Testing (acceptance, system, integration, end-to-end, security)
- Accessibility (open-source versioned repository)
- Licensing (defined license)

For more details, see Section 4.2.2 and Annex IX: Software Quality Guidelines.

4.3.2 Software Delivery Guidelines and Indicators

Integration to the EBRAINS RI will be evaluated according to specified criteria, such as maturity and a set of pre-defined deployment rules (see Section 4.2.3).

- Testing coverage in accordance with the above software quality guidelines, using the appropriate EBRAINS provided (testing, staging and production) environments.
- Adoption of continuous integration (CI) and continuous delivery (CD) pipelines. (see also 4.2.3.5 Development Environment)
- Security Indicators (to be discussed and defined in the Security Working Group-the scope of this working group is described in Annex I: Technical Coordination)

4.3.3 Component/service level monitoring

The indicators below assume that Service Level Agreements for each component/service will be in place and evaluated at regular intervals. Some of these indicators will depend on base infrastructure (FenixFenix) monitoring services (see Section 4.1.6.6):

- Availability (uptime, downtime)
- Usage of resources (per FenixFenix site (CSCS, Juelich, etc.) and type of resources (VMs, HPC)
- Application performance monitoring (performance issues)
- Notifications (security, maintenance)

4.3.4 KPIs

In this subsection, the **metrics/KPIs** for monitoring and assuring the quality of the EBRAINS RI, specifically for the integration process, are detailed. The aim is to coordinate all components to adhere to EBRAINS architecture and integration guidelines and to report on the overall performance/operational status of the EBRAINS RI in its entirety. Moreover, the RI's overall engagement, collaboration and customer satisfaction will be measured and relevant metrics are intended be made available.

The proposed KPIs list that will be measured by the EBRAINS TC is presented extensively in Table 6 and Table 7. For easy communication, the KPIs list was decomposed to 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
  - Impact Indicator (Customer Satisfaction & Fidelity)
  - Collaboration indicators
- Development and Operations
  - Documentation
  - Service Operations
  - Component maturity
  - Responsiveness/Readiness

Note here that the proposed KPIs are intended for goal-setting and intention signaling, not as commitments in the sense of SLAs. For **KPIs** that reference TC instruments and TC quality assurance procedures and mechanisms please refer to **Annex I: Technical Coordination** for more details.

Moreover, note that the below KPIs are aggregated from metrics from individual components. These are used to show trends for the entire RI and not intended for performance/progress analysis of a particular component.

Documentation is a foundational aspect of the delivery of the RI, so dedicated KPIs keep track of the components' efforts towards documenting their offerings.

KPIs regarding Service Operations and specifically "Deployments to prescribed service environments" exist to track conformance to the integration/delivery guidelines and to monitor "EBRAINS APIs adoption" and timely implementation of EBRAINS physical deployment plans.

The KPIs based on metrics from TC Gitlab issues (see Section 4.1.2.2) 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.

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". Target values for important project milestones are included (namely M18, M24, and M36). Descriptions also indicate which are extracted manually, semi-automatically, and automatically and the calculation procedures where appropriate.

Table 6: Administration and Management KPIs

Administration and Management	KPI name	KPI description	Target value M18	Target value M24	Target value M36
Impact Indicator (Customer Satisfaction & Fidelity)	UATs satisfaction	Point in time KPI. User Acceptance Panels fill relevant Questionnaires about satisfaction on EBRAINS features/functionality with scores from 1-10. Calculated as the average score of all submissions for a given Questionnaire. The Questionnaires will be handed out to the User Acceptance Panes approximately every 6 months.	-	>7	>8.5
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%	>85%	>85%
	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 over the preceding 3 months.	<2 weeks	<2 weeks	<1 week

Table 7: Development and Operations KPIs

Development and Operations	KPI name	KPI description	Target value M18	Target value M24	Target value M36
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%	>90%	100%
	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%	>80%	100%
	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%	>60%	100%
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.	-	>85%	100%
	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.	-	>85%	100%

Development and Operations	KPI name	KPI description	Target value M18	Target value M24	Target value M36
	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.	-	>85%	100%
	Uptime	6-month average EBRAINS RI uptime percentage. Calculated automatically from the monitoring system.	>98%	>98.5%	>99%
Component maturity	EBRAINS APIS adoption	Point in time KPI. Percentage of components that are using the EBRAINS APIs. Calculated semi-automatically from the EBRAINS API gateway every 3 months. (Note: target values may appear low because APIs adoption does not apply to all components)	>20%	>40%	>50%
	Use cases coverage	Point in time KPI. Percentage of use cases defined in Annex VI that are fulfilled. Calculated manually approximately every 6 months.	>60%	>85%	100%
	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%	>70%	100%
	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%	>70%	100%
Responsiveness/Readiness	Turnaround of TC Gitlab active issues.	3-month average percentage of active issues which have been active more than 3 weeks. Calculated semi-automatically from Gitlab.	<70%	<50%	<20%
	TC Gitlab active issues	3-month average number of active issues. Calculated semi-automatically from Gitlab.	<75	<115	<35
	Turnaround of TC Gitlab active SC prioritization issues.	3-month average percentage of active SC prioritization issues which have been active more than 3 weeks. Calculated semi-automatically from Gitlab.	<70%	<50%	<10%
	TC Gitlab active SC prioritization issues.	3-month average number of active SC prioritization. Calculated semi-automatically from Gitlab.	<35	<53	<13
	TC Gitlab open issues	3-month average number of open issues. Calculated semi-automatically from Gitlab.	<20	<25	<10
	TC Gitlab backlog issues	3-month average number of backlog issues. Calculated semi-automatically from Gitlab.	<25	<45	<10

Development and Operations	KPI name	KPI description	Target value M18	Target value M24	Target value M36
	TC Gitlab in-progress issues	3-month average number of in-progress issues. Calculated semi-automatically from Gitlab.	<30	<45	<15
	TC Gitlab open SC prioritization issues	3-month average number of open SC prioritisation issues. Calculated semi-automatically from Gitlab.	<10	<8	<5
	TC Gitlab backlog SC prioritisation issues	3-month average number of backlog SC prioritisation issues. Calculated semi-automatically from Gitlab.	<10	<20	<5
	TC Gitlab in-progress SC prioritisation issues	3-month average number of in-progress SC prioritisation issues. Calculated semi-automatically from Gitlab.	<15	<25	<3
	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%	>80%	100%

5. Conclusions

The current document is a *"Report presenting (a) the EBRAINS Infrastructure user requirements, its detailed logical architecture, deployment planning, integration approach and guidelines, (b) the delivery processes, quality assurance procedures, mechanisms and metrics/KPIs, software and software production management tools and dependency/licensing policies. The report will be continuously updated throughout the duration of the project to reflect the evolution and alignment of the EBRAINS infrastructure."*[11]

The work presented in this document can be summarised as follows:

- The methods for the elicitation of the **requirements** in accordance with ISO/IEC/IEEE 29148:2018 [1], the **use cases** used during this process and a **first set of requirements** for the technical part of EBRAINS RI in Section 2, Annex VI and Annex VIII respectively.
- The **Logical Architecture** (Section 3.3) with diagrams, the **Physical Deployment** with diagrams of present and projected states (Section 3.4) and the diagrams for the different deployment types (Section 3.8.2).
- the **Developer Services and Tools** (Section 4.1.5) including project **management services**, how they connect with **continuous integration** and how it provides **delivery pipelines**, and **deployment environments**.
- **Quality assurance procedures** and **KPIs table** (Section 4.3).

We should also note that the writing of this deliverable highlighted the importance of the collaboration with the different **Service Categories** (SCs), the **ICEI**, the **SLU,** and the **HLST**. All these collaborations have been already established and are expected to be more active in the next period.

In order to be **proactive** and **assess what is not working and how it can be fixed**, we are constantly collaborating with the component owners, developers, members of the HBP, helping them, explaining to them our approach; we are trying to be in contact with all the HBP Partner who need us.

We are, of course, aware of the **risk** that people get lost in all these "rules" described in such a long document, but the guidelines presented above have already been organically discussed, agreed, and enacted with SGA3 Partners. We do not assume that the Partners will read this document and follow it; D5.3 only serves reporting needs and as a point of reference. Most of the "rules" described in the Deliverable are **living knowledge** and already applied. They are collected and presented in the Deliverable in order to have well-structured information in one place. Moreover, a discussion with every component owner is taking place during the component's integration process, so all these guidelines are explained and discussed.

Closing this document, we should again stress that this is a **live document**. It will be continuously updated throughout the duration of the Project to reflect the evolution and alignment of the EBRAINS infrastructure with the scientific, technical, and sustainability requirements of SGA3 and the EBRAINS AISBL. Once officially delivered, the report will be available in the **Technical Collaboration collab** (Annex I: Technical Coordination). The information included in this document and especially the sections targeting developers (e.g., technical guidelines) will be maintained and updated outside this report and in the **TC Collab**, as separate documents, shared with **all SGA3 Partners**.

[11] D5.3 description in Specific Agreement number: 945539 — Human Brain Project Specific Grant Agreement 3 (HBP SGA3)

6. References

[1] ISO/IEC/IEEE 29148:2018: Systems and software engineering - Life cycle processes - Requirements engineering
[2] https://www.humanbrainproject.eu/en/education/codejam11/
[3] https://wiki.ebrains.eu/bin/view/Collabs/tech-coordination-weeklies/Timeline%20for%20ICEI%20services/
[4] https://fenix-ri.eu/infrastructure/resources/available-resources
[5] https://drive.ebrains.eu/smart-link/c85e59e6-cdc0-4b08-9e20-6f16e2d81e3a/
[6] https://wiki.ebrains.eu/bin/view/Collabs/the-collaboratory/
[7] https://wiki.ebrains.eu/bin/view/Collabs/the-collaboratory/Getting%20Started/#HWikipages
[8] https://wiki.ebrains.eu/bin/view/Collabs/collaboratory-community-apps/Community%20App%20Developer%20Guide/
[9] https://element.io/blog/welcome-to-element/]
[10] https://wiki.ebrains.eu/bin/view/Collabs/matrix-communication/
[11] https://wiki.ebrains.eu/bin/view/Collabs/collaboratory-community-apps/Community%20App%20Developer%20Guide/1.%20Registering%20an%20OIDC%20client/
[12] https://wiki.ebrains.eu/bin/view/Collabs/collaboratory-migration/Tutorial/Migrate%20OIDC%20Client/
[13] https://kg.ebrains.eu/develop.html
[14] https://github.com/HumanBrainProject/openMINDS
[15] https://ebrains.eu/news/new-openminds-metadata-models/
[16] https://fenix-ri.eu/sites/default/files/public/file-uploads/ICEI-D3.6-v3.1_clean.pdf
[17] https://wiki.ebrains.eu/bin/view/Collabs/hbp-ebrains-use-cases/
[18] https://www.unicore.eu/docstore/unicorex-8.0.4/unicorex-manual.pdf
[19] https://www.unicore.eu/docstore/tsi-8.0.0/tsi-manual.pdf
[20] https://www.unicore.eu/docstore/workflow-8.0.0/workflow-manual.pdf
[21] https://www.unicore.eu/about-unicore/case-studies/jupyter-at-jsc/
[22] https://www.redhat.com/en/topics/openstack
[23] https://user.cscs.ch/tools/openstack/
[24] https://docs.openstack.org/horizon/latest/user/manage-containers.html
[25] https://fenix-ri.eu/infrastructure/services/archival-data-repositories
[26] https://wiki.ebrains.eu/bin/view/Collabs/infrastructure/backups/
[27] https://wiki.ebrains.eu/bin/view/Collabs/kubernetes/docker-registry/
[28] https://zammad.org/
[29] https://jupyter.org/
[30] https://fenix-ri.eu/sites/default/files/public/file-uploads/Dec10-2019-Fenix-Webinar_Sadaf.pdf,
[31] https://singularity-hub.org/
[32] https://sarus.readthedocs.io/en/stable/
[33] https://wiki.ebrains.eu/bin/view/Collabs/technical-coordination/EBRAINS%20components/Supercomputing%20Proxy
[34] https://wiki.ebrains.eu/bin/view/Collabs/technical-coordination/EBRAINS%20components/Data%20Parallel%20Proxy/
[35] https://www.zabbix.com/whats_new_5_2
[36] https://status.humanbrainproject.eu
[37] https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?limit=3&start=6
[38] https://tools.ietf.org/html/rfc2119
[39] https://drive.ebrains.eu/smart-link/6172acb5-89ba-4819-9fe7-cfd41eb9d5fb/
[40] https://wiki.ebrains.eu/bin/view/Collabs/matrix-communication
[41] https://www.elastic.co/what-is/elk-stack

7. Annexes

The Annexes provided in D5.3 offer additional insights and details to the reader.

- **Annexes I** and **II** detail the approach of the SGA3 Technical Coordination team and its current progress.
- **Annexes V, IX** and **X**, present the outputs of the different Working Groups contributing to this report. In this version of D5.3, only the URLs to these documents and a short description of each one of them are included, in order to reduce the length of the current document.
- **Annexes III** and **IV** present the current list of EBRAINS components and Web-APIs; **Annex IV** contains a URL to the respective list, since its detailed content increased the volume of the current document.
- **Annexes VI**, **VII** and **VIII** present the EBRAINS Use Cases, (the URL to) the questionnaire used in requirements elicitation process, and the established set of requirements respectively.

Annex I: Technical Coordination

How Technical Coordination works, which are the TC Instruments.

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. It also lays out the rules for software quality in the project. EBRAINS TC is headed by the WP5 Lead (T5.11 is the main task of EBRAINS TC): Prof. Yannis Ioannidis is the EBRAINS Technical Coordinator. ATHENA RC, the lead partner of TC team, and EPFL (the members of EPFL now belong to AISBL) are the two partners that form EBRAINS Technical Coordination.

At the beginning of SGA 3, TC had to tackle different findings, already discovered before the start of SGA3; so their disciplined and proactive management was a core activity of SGA3 since its start, and embedded within the TC and EBRAINS coordination actions.

While the second month of the project was running, TC organized a 3-day virtual TC kick-off event, attended by more than 200 SGA3 members. During this event, we presented and discussed the **RI vision, TC instruments/processes** and **interventions,** while all components prepared and delivered in advance a **2-page summary** of their component **status** (e.g. roadmap, testing, deployment, documentation) and core SGA3 components delivered presentations presenting their current state of work and challenges. During a **Plenary consensus-building** session at the end of the event, the vision, directions, and interventions were discussed in order to create the basis for the next steps and the planning until the end of SGA3.

Our work begun with the creation of a comprehensive, **EBRAINS RI components catalogue**, with the initial input of all SGA3 partners delivered during our 3-day TC Kick-off meeting (M2). The catalogue has been created as a *stable and constantly updated* knowledge base for the collection, disambiguation, and dissemination of technical knowledge across the Consortium. The original input from partners was assessed and cross-referenced across all available distributed sources of current and historical information (i.e. SGA3, Collab 1/2, HLST, PLUS) under a **critical perspective.** A comprehensive reporting template was prepared, and information was transferred into the joint **TC Collab,** while partners were instructed to update/complete their respective contributions and ensure the provided information remains updated during the **entire duration** of the project. This catalogue is the foundation for **sharing** and **monitoring** up to date information per EBRAINS component, promoting **synergies** and reuse with other activities (e.g. front-facing public roadmap, HLST), while also guiding the planning of the different phases of development, integration and delivery activities. A concise version of this catalogue is available in **Annex IV** of the current deliverable.

Further, three working groups (WGs), i.e. thematically driven teams comprising members of the SGA3 Consortium 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' and EBRAINS **Software Delivery** WG – 'ESD-WG'), providing valuable feedback, that informed the guidelines of the respective sections of this deliverable. The output of these working groups is available in **Annex V**, **IX,** and **X** respectively. The WGs will remain active throughout SGA3, offering their continuous feedback and support, complemented by two additional WG on security (EBRAINS Security WG – 'EIS-WG') and EBRAINS APIs (EBRAINS-APIs WG – 'EA-WG'). Their role is described below:

- **EBRAINS Architecture WG (EAI-WG):** monitors and validates the architecture and linking of components.
- **EBRAINS Infrastructure Software Quality WG (ESQ-WG):** supervises software production methodology and software quality itself.
- **EBRAINS Software Delivery WG (ESD-WG):** monitors activities regarding the delivery and support of software
- **EBRAINS Security WG (EIS-WG):** establishes a common understanding of the intertwined technical and organizational matters relating to the EBRAINS Security.

- **EBRAINS APIs WG (EA-WG):** define the EBRAINS-wide use of select APIs.

The EBRAINS TC operates through two major meetings it has established: **TC weeklies** and **TC-Task Force (TC-TF)**.

TC weeklies are open meetings being held once a week, monitoring technical progress on the RI level; they have already become a meeting point of all the technical people of the project, critical not only for the technical progress required but also for enhancing the spirit of integration and contribution in achieving a common goal. TC weeklies are **open** to all members of SGA3 with **standing** members representing WPs, ICEI/FenixFenix sites. They are weekly 'agile' meetings for monitoring, reporting and follow-up of formal technical directions, while, when necessary, there is a **follow up** with ad hoc meetings for specific issues/areas of interest. In TC weeklies partners can/should request technical directions and TC Team raise technical issues. The critical issues that require decision-making and broader coordination are forwarded to the **TC-TF. TC weeklies** bring together more than **60 participants**, a number that proves the level of **engagement** of SGA3 members. In TC weeklies, a **GitLab Kanban** (Section 4.1.2.2) is used for creating issues, keeping track of progress and resolving any blocking points. It was available also during SGA2 and extended in SGA3. Developers, but also neuroscientists, researchers, component owners, who are members of the SGA3, are welcomed to create issues - also known as cards- in the GitLab Kanban that EBRAINS carries. Whoever wants to open issues, must have an EBRAINS account and must contact the support team to include them in the GitLab Project. Issues may come with tags and priorities to the frontline so a prioritisation could possibly be made, if applicable. Until now (January 2021), more than **150** issues have been closed, more than **40** issues are open or even in progress, while most issues close successfully in less than 3 weeks.

The **TC Task Force** is a monthly panel, where TC issues are discussed and implementation directions are provided by the EBRAINS Technical Coordinator; it provides technical directions to all EBRAINS component teams. There are **scheduled** and ad hoc sessions during which TC issues are **raised, discussed, and explored.** TC-TF is the place where dispute escalated by the EBRAINS TC to the EBRAINS Coordination is resolved.

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**[12]), while the information regarding TC weeklies is available in the respective collab (**Tech Coordination weeklies**[13]). **TC Collab** is the central collaboration point on all TC aspects and is expected to be used by all **TC instruments** in order to facilitate coordination and planning activities, flow of information and knowledge transfer.

Individual sections will be used as the main synchronization and reference points to establish a solid and effective workflow.

In full alignment with the SGA3 provisions and following a series of discussions and consultations in the context of the TC-TF (Technical Coordination Task Force), Working Groups, and higher-level decision-making bodies of SGA3, a *pragmatic high-level roadmap* towards the delivery of EBRAINS has been established, agreed up, and enacted. This roadmap will be applied to deliver, monitor, and convey the shared responsibility for the delivery of EBRAINS, while specializing and integrating all technical guidelines identified within this report.

The roadmap comprises **four phases**, spanning the course of SGA3, presented in Annex II: TC planning:

- Phase 1 (Duration: M4-M11; Output: MS5.1 Proof of Concept EBRAINS RI).
- Phase 2 (Duration: M12-M18; Output: MS5.2 Beta EBRAINS RI).
- Phase 3 (Duration: M19-M30; Output: RC EBRAINS RI).
- Phase 4 (Duration: M31-M36; Output: EBRAINS RI).

[12] https://wiki.ebrains.eu/bin/view/Collabs/technical-coordination/

[13] https://wiki.ebrains.eu/bin/view/Collabs/tech-coordination-weeklies/

Currently, **Phase 1** is ongoing, with its early output already integrated in the current guidelines. Four components (**Arbor**, **NEST**, **SimVisSuite**, **The Virtual Brain (TVB)**) have been selected in order to provide a manageable, yet representative, assembly of technical maturity, established development and testing processes, and ongoing deployment practices. This phase will allow us to streamline the integration practices for all EBRAINS components, identify potential issues and blocking points, test the guidelines presented in this report, and ultimately inform and improve the EBRAINS technical guidelines to be applied and enforced across the board in Phase 2.

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. 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.

The next diagram (Figure 14) depicts the approach followed by the Technical Coordination:

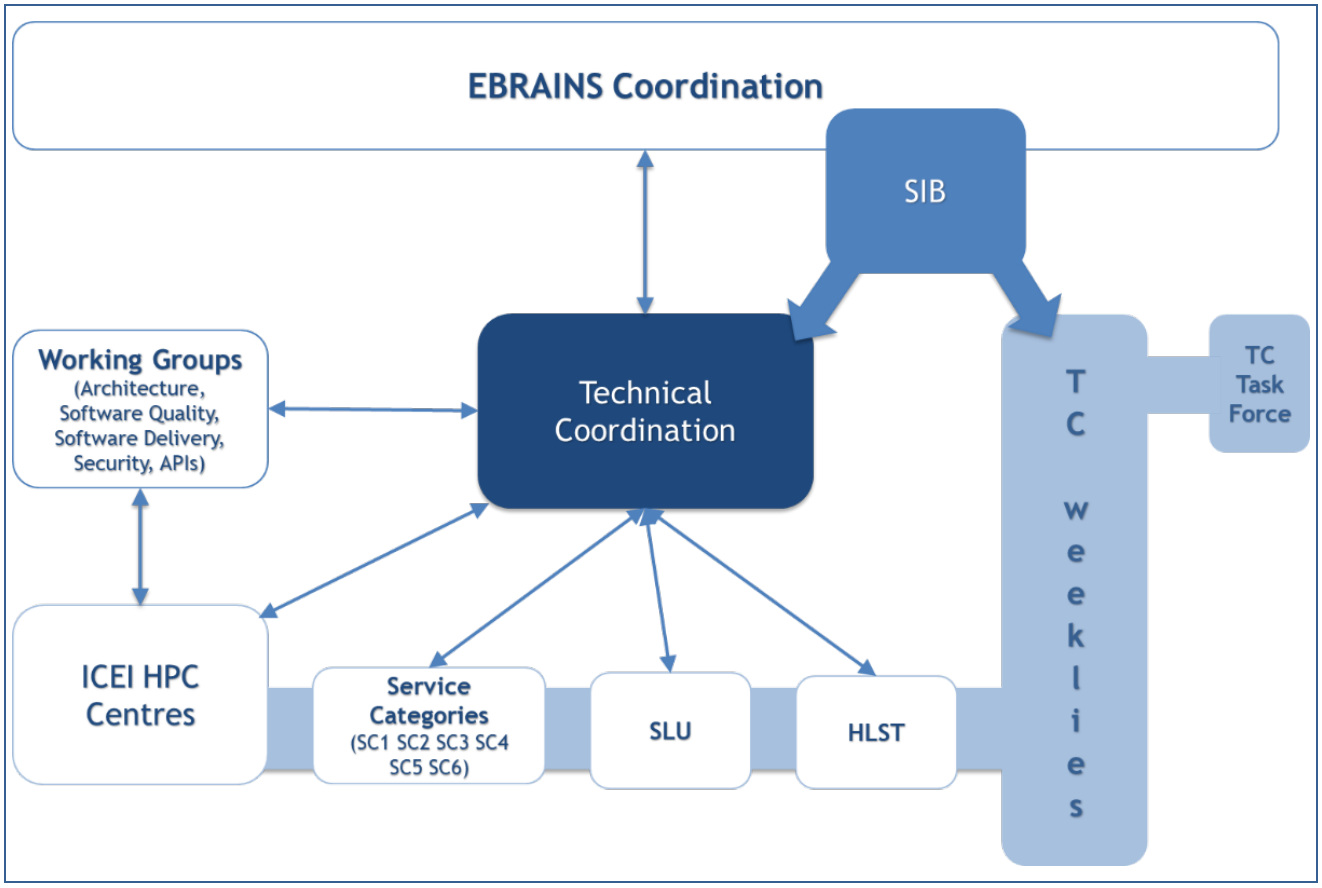

Figure 14: Technical Coordination approach

Annex II: TC planning

This document serves to present a high-level overview of the roadmap towards delivering the EBRAINS RI at the end of SGA3. Its purpose is not to replace the contractual milestones, deliverables, and deadlines of SGA3, but rather to (a) establish a pragmatic roadmap for the delivery of the EBRAINS RI, (b) provide technical and infrastructural stability, (c) inform the development and delivery planning of the various EBRAINS RI components.

Contents

Planning

The following is a high-level overview of the roadmap towards delivering the EBRAINS RI at the end of SGA3. This proposal is only a *starting point*, which will be updated to reflect the discussion, insights, and decisions of the TC-TF panel. Its purpose is not to replace the contractual milestones, deliverables, and deadlines of SGA3, but rather to (a) establish a pragmatic roadmap for the delivery of the EBRAINS RI, (b) provide technical and infrastructural stability, (c) inform the development and delivery planning of the various EBRAINS RI components.

Terminology

EBRAINS RI

Please note that in this text, and to avoid confusion, the term 'EBRAINS RI' refers to the final output of SGA3 (i.e. the EBRAINS Research Infrastructure), produced during SGA3 and delivered at the end of the project. All services currently offered are assumed as currently not integrated in the EBRAINS RI. This terminology has no effect to 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 graduation of current services as integrated EBRAINS RI components.

Integrated Component

An EBRAINS RI component is considered to be integrated in the EBRAINS RI when all of the following conditions are met:

- The component's **documentation** in the **TC Collab** is approved by the TC team (*aka: 2-pagers*).

- The component adheres to the requirements set forth in the EBRAINS RI **Architecture** prescribed in D5.3 (due M10) as independently evaluated and approved by the TC team.
- The component adheres to the **Software Quality** and **Software Delivery** guidelines prescribed in D5.3 (due M10) as independently evaluated and approved by the TC team.

Upon acceptance of a component as integrated in the EBRAINS RI:

- The component may continue to be developed (*e.g. new features, patches*) from the respective SGA3 partners following its established development processes, ensuring however that at all times: (a) the documentation in the TC Collab is up to date, (b) the provisions for the component in the EBRAINS RI Architecture (and its iterations), are satisfied.
  - *Note: CI/CD is encouraged for the internal development process*
- A new version of the component may be released and tested in the EBRAINS RI Testing environment at any point in time following the Software Quality and Software Delivery guidelines.
  - *Note: CI/CD is encouraged for the Testing environment*
- A new version of a component may be released as ready for deployment in the production environment at any point in time following the acceptance processes established in the Software Quality and Software Delivery guidelines.
- The accepted new version of the components will be deployed in the production environment of the EBRAINS RI at **predefined release cycles** (e.g. once every 2-4 months).
  - *Note: a select number of components may be accepted for continuous delivery in the production environment. Further, a process for hotfixes outside the release cycle will be established and applied as required.*

The above processes will be developed and tested during Phase 1 (see EBRAINS RI Phases).

EBRAINS RI Phases

Phase 1 (Duration: M4-M11; Output: MS5.1 Proof of Concept EBRAINS infrastructure)

EBRAINS RI Deployment.

EBRAINS RI will comprise **1 Testing, 1 Staging, and 1 Production** environment at **CSCS** (main site), and **FZJ** (secondary site)

- This infrastructure will be considered as stable until the end of Phase 1; Until then, services from other sites will not be used.
- The development of EBRAINS RI components from SGA3 partners, as well as the allocation of FenixFenix resources to projects will not change; the only exception is a small number of EBRAINS components which will participate in this testing-period (TBD)
- This infrastructure will be used to:
  - assess and finalize ICEI/Fenix-powered services (e.g. containerization) to be 3deployed and become available across all sites during Phase 2.
  - setup all TC-related processes and instruments for monitoring software quality and delivery
    - release management (features, dependencies)
    - individual component testing
    - integration testing
    - end-to-end (e2e) testing (EBRAINS)

- graduation to release (availability, announcement, deployment)

- migrate a select number of components from a project-centred deployment and management mode to an EBRAINS-centred one.
- evaluate and propose the required middleware components.
- deliver a series of Proofs of Concept (PoCs) (see next)
- inform the elaboration of the EBRAINS RI Architecture (conceptual, physical, logical)
- prepare the on-boarding of the 3 additional Fenix sites.

EBRAINS RI Resource Allocation.

The aforementioned Fenix resources will be reserved and allocated from the TC team. In subsequent phases, this process will be expanded and enforced for all components comprising the EBRAINS RI, aiming to deliver this responsibility to the EBRAINS AISBL post-SGA3.

Facilities.

The following services will be applied to support the implementation of this phase.

- Matrix (CSCS). EBRAINS-wide secure communication service facilitating timely communication on a technical level. Further, it will be used as a broadcast service for EBRAINS-wide announcements of urgent technical issues (e.g. unscheduled downtime).
- OpenShift (CSCS + JSC). The containerization service of choice for EBRAINS RI services, currently in small-scale production deployment at CSCS. Its use will be expanded (including the allocated resources).
- Docker registry (CSCS). Will be used to manage the life-cycle of all images used and produced (e.g. components, applications, notebooks).
- Jenkins (CSCS + JSC). CI/CD framework for orchestrating and monitoring integration and delivery/deployment of EBRAINS RI components.

Proof of Concepts (PoCs).

A series of PoCs will be designed, developed, iterated, and delivered in this phase. They represent the most important architectural and operational decisions established for the EBRAINS RI thus far. The PoCs serve to **test** these concepts, **improve** them, and deliver **blueprints** for large-scale adoption in the subsequent phase. If something will not work as envisaged or if it ultimately does not serve the needs of the EBRAINS RI, this is the time to discover it.

- CI/CD framework & pipelines. Setup/extend facilities, apply for select Phase 1 EBRAINS components, notebooks, images, HPC environment (work has started, all related TC Weeklies cards will be consolidated)
- OpenShift. Apply for select Phase 1 EBRAINS components, prioritize containerized deployment vs. VMs (where applicable). Apply across all other PoCs (e.g. observability, notebooks).
- Multi-scale, multi-site workflows. Small set (e.g. 2-5) of workflows implemented and working across CSCS and JSC. Any technical interventions required to deliver them will be addressed formally in the next phase.
- Monitoring & Observability. Establish available component/platform/infrastructure- specific sources of information, assess completeness of collected information, prototype implementations for automated sharing (push/pull) of information to a centrally managed monitoring service.
- Collaboratory/Notebooks. Provide multiple options for curated images and instances for end-users (back-end & prototype front-end), spawning within the same or remote ICEI/Fenix sites (no

longer a single monolithic image for pseudo-integration). Users must be able to select (a) from a selection of curated images and (b) processing environments of different size (tiered, from free to commercial) CPU/memory and/or type (e.g. HPC via service accounts). Further, introduce simple e2e testing for production notebooks (pass/fail).

- Provenance. Identify all sources for information that needs to be captured (e.g. library notebook, workflow, processing environment, data/model) per use case and the currently available metadata for each source (emphasis on automatically generated metadata/logs). Prototype assembling the available metadata under one record (with PID (persistent identifier)) and providing a simple API. Limited interventions in EBRAINS components where needed; establish roadmap for formal PROV-O metadata/service in Phase 2.
- High Availability. Prototype multi-site HA deployment for Collaboratory: fail-over (with automated sync) for Wiki/Drive at JSC and experimental multi-site for notebooks (link with previous action point).

Action items

ICEI/Fenix

- Information per site of available VM resources and HPC resources (i.e. maximums) in TC Collab
- Expose EBRAINS resource utilization per site for VM resources and HPC resources.
- Stop-gap measure till EBRAINS-wide monitoring is in place (see observability): monthly per site update of utilization for VM resources and HPC resources (i.e. % of maximum) in TC Collab.
- Monthly per site progress updates on the delivery of ICEI/Fenix services in TC Collab
- Provide access to FURMS specifications & design documents as they currently stand; establish pathway for influencing them to accommodate EBRAINS RI requirements

Component owners (SGA3 partners)

- Update initial information for ALL components in TC Collab and continuous updates from that point on. This will be a KPI monitored and reported by the TC team.
- Candidates to participate in Phase 1 exploratory and testing activities for setting up integration, testing, and delivery guidelines.

@ALL

Common licensing guidelines and attribution.

- All EBRAINS components (software, hardware) must contain appropriate acknowledgments to SGA3 (also SGA2-1 as relevant) and EBRAINS; link with PCO guidelines & WP8.
- All software components should be licensed with a permissive14 license (preferably the Apache License15), which allows proprietarisation, except for cases where:
  - The software is already proprietary;
  - The software is an extension of existing software already licensed; all SGA3 contributions must conform to the software's original license;
  - The software applies non-permissive software; these cases need to be discussed with the TC to avoid potential future problems related to commercialisation

*The exact scope of the subsequent Phases will be established in M9 (December 2020) following the progress and insights extracted from Phase 1. The following sub-sections are only placeholders and include a limited selection of high-level actions, so **please ignore for the time being**.*

[14] https://en.wikipedia.org/wiki/Permissive_software_license

[15] https://en.wikipedia.org/wiki/Apache_License

Phase 2 (Duration: M12-M18; Output: MS5.2 Beta EBRAINS Infrastructure)

- Integrate 80% of current components to the EBRAINS RI (full conformance to D5.3 EBRAINS Technical Coordination Guidelines)
- Expand the stack of services provided by CSCS and JSC to the remaining 3 ICEI/Fenix sites (multi-site deployment and site-specific considerations according to the EBRAINS Architecture, see D5.3; acceptance test for all site services and corresponding SLA)
- Formalization of EBRAINS APIs (de facto, documentation, new)
- Monitoring, Accounting, and Observability: EBRAINS RI-wide services in operation
- Middleware components gradually integrated across the EBRAINS RI
- Period 1 EC Review preparation

Phase 3 (Duration: M19-M30; Output: Alpha EBRAINS Infrastructure)

- Define, develop, and test business/pricing models
- All processes for handover to AISBL elaborated, bootstrapped and tested

Phase 4 (Duration: M31-M36; Output: EBRAINS Infrastructure)

- Handover to AISBL
- Period 1 EC Review preparation

EBRAINS RI Seasons

All SGA3 partners participate as relevant in the following EBRAINS RI Seasons. The term is inspired by Google's Summer of Code and aims to establish a series of *thematic-specific* periods of focus for cross-EBRAINS technical advances. The current planning is for:

Autumn of Preparation (2020).

This is an introspective period, building upon the outputs of the Internal Review focused on designing, developing, and testing: (a) artifacts enacting the core concepts of the EBRAINS RI Architecture (conceptual, logical, physical), (b) technical interventions to support the SGA3 partners in the subsequent phases of the project, and (c) the process and instruments to be applied for EBRAINS RI component integration, testing, and delivery. In a nutshell, we will be setting up and testing the concepts and process to be applied for the development of the EBRAINS RI during SGA3.

Winter of Documentation (2020-2021).

In this period, the emphasis of all partners developing EBRAINS RI components will be placed on improving the quality and completeness of the documentation for their respective components. This includes (a) improvement of all types of relevant documentation (e.g. technical, developer, end-user), (b) setup and enforcement of stable and well-known processes to constantly update and improve documentation, (c) SGA3-wide linked actions for intercepting and resolving documentation issues.

Spring of Integration (2021).

In this period, our focus is bootstrapping the majority of EBRAINS RI components to the integration and testing processes of D5.3 (Technical Coordination Guidelines). We will be aiming for at least 80% of EBRAINS RI components fully adhering to these guidelines by the end of this period, which is timed with **MS5.1 PoC EBRAINS infrastructure (M12)**. Please note that up to, and till the end of this period, EBRAINS RI PoC has not been made public yet; integration and testing is performed on non-production setting.

Summer of Deployment (2021).

In this period, our focus is on externalizing the EBRAINS RI produced thus far, by deploying all its integrated components in a **production setting**. In parallel, the remaining 20% of components are similarly integrated, tested, and deployed. In nutshell, the SGA3 produced EBRAINS RI materializes and becomes publicly available in this period, producing substantial feedback to be collected, assessed, and acted upon via short-/medium-term corrective and realignment actions.

Autumn of Ambition (2021).

This period has a singular focus on achieving **MS5.2 Beta EBRAINS infrastructure (M18)** and preparing the review of the EBRAINS RI from the EC.

Annex III: EBRAINS components Web-APIs list

The list of currently known and tracked component web APIs.

Component Name	API endpoint	Documentation	Type
NMPI (NMC either BrainScaleS or SpiNNaker)	https://nmpi.hbpneuromorphic.eu/api/v2	https://electronicvisions.github.io/hbp-sp9-guidebook/reference/python_client.html	REST
Wiki	https://wiki.ebrains.eu/rest/	https://wiki.ebrains.eu/bin/view/Collabs/collaboratory-community-apps/Community App Developer Guide/Interacting with the wiki/	REST
IAM	https://iam.ebrains.eu/auth/	https://www.keycloak.org/documentation.html	REST
Drive	https://drive.ebrains.eu/api2/	https://wiki.ebrains.eu/bin/view/Collabs/collaboratory-community-apps/Community App Developer Guide/Interacting with the drive/	REST
Model Validation service	https://validation-v2.brainsimulation.eu/	https://validation-v2.brainsimulation.eu/docs	REST
Ilastik	To be confirmed	To be confirmed	tbc
Insite	To be confirmed	https://devhub.vr.rwth-aachen.de/VR-Group/in-situ-pipeline/insite/-/wikis/api/api_reference, https://devhub.vr.rwth-aachen.de/VR-Group/in-situ-pipeline/access-node/-/blob/develop/access_node/swagger/swagger.yaml	REST
KnowledgeGraph	https://core.kg.ebrains.eu/v3-beta/	https://core.kg.ebrains.eu/swagger-ui/index.html?configUrl=/v3/api-docs/swagger-config#/	REST
Neo-Viewer	https://neo-viewer.brainsimulation.eu/blockdata/	https://neo-viewer.brainsimulation.eu/	REST
UNICORE	API endpoint example: https://gateway_url/SITENAME/rest/core Deployments exist in all 5 sites: 1. https://zam2125.zam.kfa-juelich.de:9112/JUSUF/rest/core 2. https://brissago.cscs.ch:8080/DAINT-CSCS/rest/core 3. https://grid.hpc.cineca.it:9111/CINECA-MARCONI/rest/core 4. https://hbp-unicore.ccc.cea.fr/irene/rest/core 5. https://unicore-hbp.bsc.es:8080/BSC-MareNostrum/rest/core	https://sourceforge.net/p/unicore/wiki/REST_API/	REST
Image service	https://img-svc.apps.hbp.eu/api/	https://img-svc.apps.hbp.eu/api-docs	REST
HPC Proxy	https://unicore-job-proxy.apps.hbp.eu/api/	https://unicore-job-proxy.apps.hbp.eu/	REST
Object Storage Proxy	https://data-proxy.ebrains.eu/api/	https://data-proxy.ebrains.eu/api/docs	REST

Annex IV: EBRAINS RI Components

A list of all EBRAINS components that have contributed to the TC Collab has been produced and can be found at:

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

The list contains information about the Architecture, the Documentation, the Source code repository and the Deployment status of every component. The maintenance of this list (in close cooperation with HLST) is of critical importance in order to be able to identify at any given time all the development effort that is conducted under the umbrella of EBRAINS RI.

Annex V: High-level EBRAINS architecture

The document was the output of EBRAINS Architecture Infrastructure Working Group (EAI-WG) detailing the high-level architecture of EBRAINS. The text from the document has significantly influenced the Architecture section and in particular the "EBRAINS conceptual architecture" sub-section in which significant pieces of the text can be found. It has not been included in the current version of D5.3 in order to reduce the length of the document, and can be found at:

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

Annex VI: Use Cases

A set of use-cases (UC) for the HBP and the EBRAINS platform. A use-case can be a specific scientific or technical challenge as encountered in the HBP or EBRAINS and describes the current or expected ICT needs. The use-cases are created in collaboration between domain scientists or technical coordinators and the use-case management team.

In this annex we included the description of the defined use-cases. The use case ID is the one used since its definition. More information is available to HBP members in the respective wiki page in EBRAINS Collaboratory.

Table 8: UC01_Olfaction

Use Case ID:	UC01_Olfaction
Title:	Data-driven cellular models of brain regions, Olfactory Bulb (#1)
Scientific Owner:	Michele Migliore
Technical Contact:	Wouter Klijn
Description:	
The scientific aim of this use case is the development of a brain prosthesis, using a morphologically and physiologically realistic computational model of a brain region involved with sensorial inputs, in order to activate the cortical neurons of a live mammal bypassing the real system. The model, implemented in its natural 3D layout and directly driven by experimental data, will be interfaced with a living animal in an almost natural setting, to guide behavioural experiments.	

Table 9: UC02_HumanBrainAtlas1 - Multiresolution

Use Case ID:	UC02_HumanBrainAtlas1 - Multiresolution
Title:	Online visualization of multi-resolution reference atlases
Scientific Owner:	Katrin Amunts
Technical Contact:	Pavel Chervakov
Description:	
Interactive access to multilevel human atlas data through the HBP atlas. HBP is aggregating a unique portfolio of high-quality human brain templates, maps and multilevel data. To generate an impact for the wider community, these data have to be provided in a simple, user-friendly fashion to users of the web interfaces and APIs of the Neuroinformatics Platform. At the macroscale, we will provide neuroimaging researchers with the data and functionality to interactively overlay HBP's whole-brain maps and parcellations with their own data in the template space of their choice, and subsequently download the transformed data for their own purposes. Here, we address the fact that today, the existence of several incompatible reference spaces used to aggregate human neuroimaging data is an impediment for meta-analysis. By providing a set of spatial transformations between the main reference spaces in the Neuroinformatics Platform, we provide a simple solution to the outside community. As of today, no other repository provides a comparable set of cross-aligned labelled human brains, and the possibility to adapt them to most of the standard spaces. At the mesoscale, we will target modellers and theoreticians, and provide them interactive access to realistic numbers of cell counts for different brain areas. HBP will provide realistic experimental 3D measures of neuronal cell numbers and densities that go significantly beyond the tables provided by von Economo. Such quantitative numbers are still missing today, in spite of being a critical requirement for setting up simulations of cortical network models. We will co-design the functionality to comfortably retrieve such data from the HBP atlas by visual exploration, or by API access in Python. Key problem here: We have very large, high-resolution 2D and 3D images that cannot be downloaded en bloc by the user for visualization. For visualization, such data needs to be streamed dynamically as multi-resolution tiles via http, so that only the data that the user actually looks at is transmitted: Either large parts are downloaded at much lower resolution, or small parts at the full resolution. This process is realized by streaming through https, as known from Google maps for the 2D case.	

Table 10: UC03_HumanBrainAtlas2 - DataManagementMicroscopy

Use Case ID:	UC03_HumanBrainAtlas2 - DataManagementMicroscopy

Title:	Data management and big data analytics for high throughput microscopy
Scientific Owner:	Dickscheid
Technical Contact:	Marcel Huysegoms
Description:	
Enabling data management and analysis for the Human Brain Atlas. The HBP Human Brain Atlas is a multi-scale, multi-model atlas with highly diverse qualitative and quantitative datasets that need to be spatially and semantically registered. These datasets have different file formats, come from different partners and need different kinds of user interface functionality. To make this data discoverable and accessible, the Neuroinformatics Platform (NIP) supports users in curating and sharing the data with other researchers in the HBP. It builds upon the infrastructure and services provided by Fenix and the HPAC Platform. In particular, the NIP requires “central” HBP data repositories (which may be federated as long as this is transparent to the user), access control and long-term data storage. The different types of user interfaces need to be supported and provenance tracking should be enabled. An efficient, sophisticated data management is of particular importance since some of the datasets are quite large. To give an example, a single complete human brain at 1-micron resolution requires, depending on the method, 2-6 Petabytes of storage space just for the original data. The HBP Rodent Brain Atlases have very similar requirements with respect to the infrastructure, data management and tools. This use case focuses on the Human Brain Atlas, but synergies will be used to enable also the data management and analysis for the Rodent Brain Atlases.	

Table 11: UC04_HumanBrainAtlas3 - LargeCohort

Use Case ID:	UC04_HumanBrainAtlas3 – LargeCohort
Title:	Data management and big data analytics for large cohort neuroimaging
Scientific Owner:	Caspers, Eickhoff
Technical Contact:	Jan Schreiber, Felix Hoffstaedter
Description:	
Neuroimaging analytics: important aspects • Large amounts of data to be processed. • To be processed data potentially includes personal data. • Data not coming in continuously - coupled to releases of repositories. • Huge amounts of single files (e.g. NIfTI) cause problems with inode quotas • Inherently parallel preprocessing pipelines (applied to each image) • Frequent sampling of different subsets of derived data for analysis • Planned to become an NIP service. Software and workflows well established and standardized, but image formats not designed for HPC	

Table 12: UC05_Learning2Learn1 - OpenLoopNetwork

Use Case ID:	UC05_Learning2Learn1 – OpenLoopNetwork
Title:	Open loop run of a complex spiking network with input data, output data and network reconfiguration by learning
Scientific Owner:	Sebastian Schmitt (BrainScaleS)
Technical Contact:	Sebastian Schmitt (BrainScaleS)
Description:	
This is the prototype Use Case for a feed forward data analysis. It corresponds to the basic way of using classical artificial neural networks (ANN), but transfers the concept to the HBP brain-inspired systems SpiNNaker and BrainScaleS, which are spiking networks. Use Cases of this type are directed towards an understanding of learning in spiking networks to exploit the potential benefits of this technology over traditional deep networks. Such benefits are energy efficiency, resilience and the ability for real-time or accelerated learning. For this Use Case, a user has developed a spiking neural network architecture, which may also apply supervised, unsupervised or reinforcement learning mechanisms. The network receives spike-coded input data, either from recorded biological sensors, or from generic databases with abstract data. Abstract data can originate from various sources in science, business or government. On a traditional computer, network simulation, and in particular learning, takes a long time. This may be prohibitive if the research includes	

extensive parameter scans for optimization purposes. The user can take advantage of the real-time capability of SpiNNaker or the acceleration factor of BrainScaleS to reduce run time, so that the effects of learning mechanisms and parameter variations can be studied. Since a user expects to submit many jobs with different parameters, the job submission process should be scripted. Output data consist of spike recordings, selected membrane traces and the learned network configurations for further interpretation.

Table 13: UC06_Learning2Learn2 - ClosedLoopNetwork

Use Case ID:	UC06_Learning2Learn2 – ClosedLoopNetwork
Title:	Closed loop run of a complex spiking network with input data, output data and network reconfiguration by learning
Scientific Owner:	Eric Müller (BrainScaleS)
Technical Contact:	Eric Müller (BrainScaleS)
Description:	
This is a Use Case which exploits learning in timing critical closed perception-action loops. Input and output happen via a simulated environment. It makes use of the specific hardware infrastructure of the neuromorphic computing platform, which allows for timing critical real-time or accelerated emulation of a network and the corresponding timing constrained simulation of an environment, sensors and actuators on a conventional computer are linked via a very low latency (µs) network. A user plans to implement and evaluate large-scale networks of neural sensorimotor cortical areas in a closed-loop configuration, linking sensory processing to a behavioural output (active perception) and a reward. The goal is to study how the functional and encoding roles of diverse neuronal populations across areas vary in time and how they are connected to the intra- and inter-cortical dynamics. This time-varying encoding across cortical areas should be considered as the key underlying mechanism for both stimulus-encoding and perceptual behaviour, which have not been studied before. On a traditional computer, network simulation, in particular learning, takes a long time. This may be prohibitive if the research includes extensive parameter scans for optimization purposes. The user can take advantage of the real-time capability of SpiNNaker or the acceleration factor of BrainScaleS to reduce run time, so that the effects of learning mechanisms and parameter variations can be studied. Since a user expects to submit many jobs with different parameters, the job submission process should be scripted. Output data consist of spike recordings, selected membrane traces, the learned network configurations and the changes to the simulated environment during the closed-loop operation.	

Table 14: UC07_Learning2Learn3 - LTLNeuromorphicNRP

Use Case ID:	UC07_Learning2Learn3 – LTLNeuromorphicNRP
Title:	Learning-to-learn (LTL) in a complex spiking network on HPC and Neuromorphic hardware interacting with NRP
Scientific Owner:	Prof. W. Maass, Prof. K. Meier
Technical Contact:	Sandra Diaz
Description:	
This is an ambitious Use Case with a strong research component and the potential for a very high impact in basic research and applications of biologically-inspired machine learning. It runs through a second loop for the optimization of the neuromorphic system parameters to achieve optimal learning capabilities. Traditional learning approaches start from a predetermined network architecture and adjust the synaptic connection strengths by established learning algorithms, mostly based on a gradient-descent method. Neural networks in biological brains are the result of a very long evolutionary process, which provided them with the ability to learn. This ability is based on a multitude of parameters, including the network architecture and size, the parameters of neurons and synapses, and, of course, the synaptic connection strength. The result of evolution is a high degree of variability in those parameters which cannot be tuned by traditional learning approaches. The learning-to-learn approach follows a double-loop strategy. An inner loop made of a neuromorphic circuit and a simulated environment, observed by sensors and modified by actuators running in a timing constraint fashion. An outer loop runs an optimization algorithm to tune the network parameters to achieve optimal learning in the inner loop. The outer loop has no latency constraints, but requires fast execution of the optimization algorithm. The outer loop is ideally suited to run on an external computer. On a traditional computer, inner loop simulation, and in particular the learning process, takes a long time. This may be prohibitive if the outer loop includes extensive parameter scans for optimization purposes. The user can take advantage of the real-time capability of SpiNNaker or the acceleration factor of BrainScaleS	

to reduce run time, so that the effects of learning mechanisms and parameter variations can be studied. Since a user expects to submit many jobs with different parameters, the job submission process should be scripted. Output data consist of spike recordings, selected membrane traces and the learned network configurations.

Table 15: UC09_Cerebellum

Use Case ID:	UC09_Cerebellum
Title:	Large scale simulations of models: Cerebellum
Scientific Owner:	Egidio D' Angelo
Technical Contact:	Claudia Casellato
Description:	
This use case aims to refine and apply molecular/cellular level models of cerebellum to (1) simulate dynamic control of plasticity in trial-and-error learning, (2) integrate the cerebellum with extra-cerebellar circuits for large-scale network simulations, (3) simplify such models and integrate them into whole-brain robotic simulators, and (4) extend cerebellar modelling through the Collaboratory. (5) All this activity will be coordinated with the development and refinement of neuroinformatics tools. The cerebellum models will also be exploited for simulations of pathological alterations of plasticity and circuit dynamics. Therefore, there will be a multiple fallout at the level of brain modelling, theoretical understanding of brain function and disease and infrastructure implementation.	

Table 16: UC10_Hippocampus

Use Case ID:	UC10_Hippocampus
Title:	Large scale simulations of models: Hippocampus
Scientific Owner:	Michele Migliore
Technical Contact:	Courcol Jean-Denis
Description:	
This Key Result will integrate several tasks and components, with the main aim to reach the main goals of HBP. The focus here will be on models of synaptic plasticity of hippocampal synapses, and how they can be integrated into cellular level microcircuit models using data-driven subcellular pathways and/or rule-based effective implementation. The effect at the microcircuit level will be investigated in terms of network self-organisation during synaptic inputs activated under different conditions of timing and spatial activation. The emphasis will be on the mechanisms underlying associative memory processes and spatial navigation, integrated into a user-friendly user interface allowing an easy community engagement to the Brain Simulation Platform and its functionalities.	

Table 17: UC11_ElephantBigData

Use Case ID:	UC11_ElephantBigData
Title:	Elephant: interactive supercomputing for the analysis of neuronal activity
Scientific Owner:	Michael Denker
Technical Contact:	Kim Sontheimer, Alper Yegenoglu
Description:	
The Electrophysiology Analysis Toolkit (Elephant) is a toolbox for the analysis of electrophysiological data, i.e. activity data recorded either in experiment or neural network simulations. Elephant provides fundamental methods that are in use by the community to analyse both spike time data as well as time-series data of neuronal population signals, such as local field potentials (LFPs). Besides methods to characterise the dynamics of single neurons or population signal recordings, its focus is on methods that analyse the ensemble activity in massively parallel data, as well as methods that bridge scales of observation (e.g. spike-LFP relationships). The library follows several design principles. All analysis functions are based on the Neo data object model. This common data representation allows methods to be easily applied to neuronal data coming from different sources, including experimental file formats or neuronal network simulations. Furthermore, the library follows a modular design; such that complex analysis methods can be built from simpler analysis steps where appropriate. This approach guarantees results of complementary methods can be meaningfully related to one another. In order to follow a principle of co-design, methods are typically provided by experts in utilizing a particular analysis function, or by authors of the original method. The library is structured by the types of analysis methods it provides.	

A full documentation is provided with the methods. Elephant is a toolbox for the analysis of electrophysiological data based on the Neo framework.

Table 18: UC13_NovelDecoderCytoarchitecture

Use Case ID:	UC13_NovelDecoderCytoarchitecture
Title:	Towards a novel decoder of brain cytoarchitecture using large scale simulations
Scientific Owner:	Cyril Poupon
Technical Contact:	Jacques-Charles Lafoucriere
Description:	

Mapping and understanding the cytoarchitectonics of the human brain is a challenge that started back at the beginning of the 20th century with famous neuroanatomists who segregated the cortex of a post-mortem human brain sample into a few dozens of areas from the observation of the laminar structure of the cortical ribbon and of its cellular organization using optical microscopy. The most famous atlas was developed in 1905 by Korbinian Brodmann and remains today widely used by neuroscientists even if it suffers from several biases: first, because it was developed from a single sample, it cannot capture the inter-subject variability of the cytoarchitecture maps; second, boundaries of the areas have been drawn from visual observations, and may not reflect the real boundaries of functional areas. The community is fighting to go beyond Brodmann areas and during the last decade, several teams attempted new strategies to map the brain cytoarchitectonics. On the one hand, the Institute of Neuroscience for Medicine (INM1, Julich Forschungszentrum, headed by Prof. K. Amunts) has launched a decade ago a large project aiming at developing a new approach to establish a novel cytoarchitectural atlas, called the Big Brain, based on the mapping of the receptor neurotransmitters. This project has become a core development in the Human Brain Project and will provide a unique mapping of functional areas of a few post-mortem samples. On the other hand, several teams have tried to establish maps from the acquisition of large cohorts of in vivo human healthy volunteers. For instance, the team of Mangin et al has investigated the structural connectivity of the cortex inferred from diffusion MRI as a potential information to further segregate Brodmann areas and propose a new parcellation of the cortical surface. More recently, the team of Glasser & al published a novel atlas including more than two hundred of cortical areas established from the individual relaxometric and functional MRI scans acquired on the Human Connectome Project cohort.

The next challenge is now to develop methods to segregate the human brain cytoarchitecture in vivo and at the individual scale. The success of such a challenge relies on the capability of modern neuroimaging methods to probe the variations of the cellular organization of brain tissues in vivo. Quantitative and diffusion MRI are known to be sensitive to the myelo- and cyto-architecture of tissues and might be good candidates to perform virtual biopsy in vivo. Quantitative MRI has been successfully used during the last decade to map the myelin water fraction using T1-weighted and T2 weighted MRI scans. Similarly, diffusion MRI has proven its potential to probe not only the structural connectivity of the human brain through the observation of the anisotropy of the random displacement of water molecules in brain tissues, but also some quantitative microstructural features characterizing their cellular organization such as the axon density or the axon diameter. Unfortunately, the cellular organization of brain tissues (gray and white matter) can be extremely complex, and today, few is known about the diffusion MRI signature of the plethora of possible cellular environments met in the brain. Diffusion MRI scans require the tuning of several sequence parameters that obviously impact the nature of the diffusion contrasts obtained at the end and few is known about the parsimony of the resulting parameter space with respect to this contrast. Investigating this question is essential to establish the reduced set of parameters to be used in vivo to preserve a reasonable scan time and still be able to collect enough diffusion MRI data to segregate the cytoarchitectural areas both in gray and white matter. Obviously, one cannot achieve an exhaustive scanning of the sequence parameter space in vivo.

This Use Case project aims at replacing in vivo diffusion MRI scans by in silico diffusion MRI scans enabling to reach a much higher level of completeness of the parameter space sampling. To do so, the Use Case will first focus on white matter (WM) cytoarchitecture being simpler than gray matter (from a cytoarchitectural point of view) and will require to:

- Task #1 - create an exhaustive bunch of in silico realistic white matter virtual tissue samples by numerical simulations of cellular membrane geometries,
- Task #2 - simulate the diffusion process of water molecules in every realistic in silico WM tissue sample using a Monte-Carlo approach,
- Task #3 - simulate the diffusion MRI signature of every WM tissue sample for an exhaustive set of diffusion MRI sequence parameters achievable on actual preclinical and clinical MRI systems,
- Task #4 - learn a deep neural network to build a decoder/regressor of the WM microstructure.
- Task #5 - use the decoder to establish an atlas of the WM microstructure

Table 19: UC14_MultiscaleCortex

Use Case ID:	UC14_MultiscaleCortex
Title:	Multi-scale co-simulation: Connecting Arbor, NEST and TVB to simulate the brain
Scientific Owner:	Jirsa, Morrison,
Destexhe, Diesmann	
Technical Contact:	
What are useful abstractions to understand the brain? What level of detail is needed to simulate cognition? This use case provides a testbed for answering these questions by integrating HBP supported and developed simulators in a single neuroscientific system: Morphologically detailed neurons simulated in Arbor or Neuron, spiking neurons networks in NEST, and large-scale whole brain models in TVB. Each platform has produced insight at specific spatial scales, but their incorporation has been limited due to different resource requirements of each simulator. The aim is to incorporate existing infrastructure to assess the contribution of neural mechanisms across scales to the generation of neural signals and ultimately cognition. This aim will be pursued with a multi-scale model: Populations of morphologically detailed neurons are simulated, embedded in a brain area of spiking neurons, communicating with connected brain regions through TVB, and producing brain imaging signals commonly used in clinic and research (iEEG, EEG, MEG, fMRI), see 1. In this way, neural signals from spikes to local field potentials and macroscopic brain signals can be examined across scales and empirical modalities. Two sets of models can be distinguished when focusing on resource requirements, differing in the manner the morphologically detailed and spiking neuron models are related to each other. The TVB model details are typically not changed between these two model sets. Although TVB modelling is essential, from a resource usage perspective it requires trivial amounts compared to NEST and Arbor/Neuron. In the first set of models, the areas simulated by spiking and morphologically detailed neurons are spatially discrete. The different simulators are modelling spatially separate parts of the brain. The result of this is that the majority of the spikes generated in each simulator will only need to be transported within the same simulator. Only the subset of neurons connected between the scales will have spikes transported to the different simulator. This relatively small interface allows the three simulators to be located on differently optimized physical systems (in the same MPI network). E.g. a large memory system with high bandwidth interconnect for NEST and a many-core booster system for Arbor/Neuron. The second set of models, spatially mixed, would see morphologically detailed neurons embedded into the spiking neuron network. The neurons at the different scales and simulated in the different simulators, are potentially fully connected. This results in a high bandwidth demand between the two simulators. This type of system can run ideally on a fat GPU nodes. Nodes with both a high-performance multi-core CPU, large memory and a many-core accelerator. Co-location of these two simulators on one node will make optimal use of all resources. For both model sets, the resource requirements for TVB are modest.	

Table 20: UC15_NRP

Use Case ID:	UC15_NRP
Title:	Neurorobotics platform, large-scale brain simulations
Scientific Owner:	Villarroel Felipe Cruz, Axel von Arnim
Technical Contact:	Villarroel Felipe Cruz, Colin John McMurtrie, Axel von Arnim
Description:	
The Neurorobotics Platform (NRP) is a tool for studying models for brain, body, and environment in closed perception-action loops through interactive in-silico experiments. It effectively allows scientists to virtualize brain and robotics research. In the NRP, web-enabled and interactive in-silico experiments connect a brain simulation and an environment simulation in advanced closed-loop experiments. Full models of robot and environment are part of an interactive computer simulation, where the simulated sensors of the robot relay environment information to a simulated nervous system that models a biological brain at different levels of detail, which in turn controls the robot. The user of the NRP can then control the simulation via a web-browser interactively and in real-time. In terms of the platform utilization, the platform service expects that once it enters production that the number of users can fluctuate between 5 to 50 concurrent users. Where each simulation would be distinctive depending on the application and user, as such, the NRP project expects that the complexity of the brain models and robots used in simulations to be wide-ranging, starting from small (brain models with thousands of neurons and simple robots) to large scales (hundreds of thousands of neurons with complex	

robotic environments). From a system requirement perspective, the complexity of the models translates to computational and resource requirements that scale quadratically with the number of neurons.

Estimating the total mix of simulations (small-, medium-, or large-scale) is difficult, however, the expected average number non-trivial large simulations per week is estimated to be close to 14, for a total of 700 non-trivial simulations per year.

Table 21: UC16_BlueBrain

Use Case ID:	UC16_BlueBrain
Title:	Blue Brain Project Micro column
Scientific Owner:	Markram, Felix Schuermann
Technical Contact:	Markram, Felix Schuermann
Description:	

User: Sam – a scientists who wants to run a microcircuit simulation

Preconditions:

Microcircuits are registered and referenced in the HBP Knowledge Graph.

Microcircuits data are stored in the HBP Knowledge graph Object Storage (currently CSCS Object storage)

The user has an HBP account and an account in the HPC Centre where he wants to run a simulation.

Success scenario:

- Sam selects a microcircuit from the ones referenced in the HBP Knowledge graph from a web GUI.
- Sam selects a HPC Centre where he wants to run his simulation from a web GUI.
- Sam selects a configure the simulation he wants to run and the parameter for the HPC job he wants to run (number of nodes for instance) from a web GUI.
- Sam launches the simulation from a web GUI, waits for confirmation that the job has been queued and logout.
- Later, Sam checks the job status from a web GUI.
- Sam sees that the simulation finished.
- Sam configures and launches pre-canned analysis jobs; Sam may select a different compute centre.
- Sam waits for completion of the analysis job and he can check the status in a web GUI.
- Sam visualizes the analysis results in a web GUI.
- Sam wants to interactively analysis the simulation results in the HBP collaboratory in a jupyter notebook.
- Sam wants to visualize interactively the simulation. The compute resource for the visualization may or may not be in the same location than the compute resource.
- Sam registers his simulation in the HBP Knowledge graph
- Sam simulation reports are stored in the HBP Object storage
- Sam does not need his HPC allocation anymore

Table 22: UC18_Macaque

Use Case ID:	UC18_Macaque
Title:	Multi-area macaque NEST simulation with live visualization and interaction
Scientific Owner:	v. Albada, Diesman
Technical Contact:	Espen Hagen, Jari Pronold, Johanna Senk
Description:	

Release of multi-area model of macaque visual cortex, improved using new connectivity and activity data. Account at cellular resolution for properties essential for cortical function, focusing on excitability and feedforward-feedback interactions.

Construct multi-layered multi-area models of the cortex relating the local microscopic connectivity to the macroscopic connectivity of the brain. On the local level, this leads to models with a higher degree of self-consistency than previously possible, because the origins of synapses from remote sources are included, and the lower parts of the power spectrum of neuronal activity missing in purely local models can be investigated. On the global level, the bottom-up and top-down flow of activity between cortical areas in these hierarchical models are investigated.

Table 23: UC23_MultiscaleModels

Use Case ID:	UC23_MultiscaleModels
Title:	Models across scales, from cellular to network models up to the whole-brain
Scientific Owner:	Viktor Jirsa
Technical Contact:	Marmaduke Woodman
Description:	
Closely related to "UC14_MultiscaleCortex" UC23 details the technical requirements for a specific model	

Table 24: UC26_WorkflowOrchestration

Use Case ID:	UC26_WorkflowOrchestration
Title:	Orchestration of modular interactive workflows for neuronal simulators
Scientific Owner:	Wouter Klijn
Technical Contact:	Wouter Klijn
Description:	
This use-case details the infrastructure work for UC14_multiscaleCortex	

Annex VII: Developer's Questionnaire

The questionnaire for developers-members of HBP can be found at:

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

Annex VIII: User Requirements

The first set of user requirements as extracted from the panels and the different discussions and meetings held until at the time of this writing.

Table 25: Requirement StRS01

ID	Level	Priority	Type
REQ-StRS01	StRS	MAN	FUNC
Name			
Single entry point			
Description			
EBRAINS must provide a single entry point where users can register and discover its offerings			

Table 26: Requirement StRS02

ID	Level	Priority	Type
REQ-StRS02	StRS	MAN	FUNC
Name			
Same credentials			
Description			
EBRAINS users should be able to use the same credentials across all of its offerings			

Table 27: Requirement StRS03

ID	Level	Priority	Type
REQ-StRS03	StRS	MAN	FUNC
Name			
EBRAINS Jupyter Notebooks			
Description			
EBRAINS users should have a single-entry point for interactive-computing, and should be able to select the appropriate notebook environment for their use case. The user should be able to select among the available sites for the execution of the experiment.			

Table 28: Requirement StRS04

ID	Level	Priority	Type
REQ-StRS04	StRS	MAN	FUNC
Name			
Scientific workflows			
Description			
EBRAINS users should be able to define scientific workflows. A user should be able to record and reproduce all the steps in a computational scientific workflow.			

Table 29: Requirement StRS05

ID	Level	Priority	Type
REQ-StRS05	StRS	MAN	FUNC
Name			
Monitoring of EBRAINS services			
Description			
EBRAINS should provide transparent access to metrics and analytics for service providers wishing to track SLA compliance of the infrastructure.			

Table 30: Requirement StRS06

ID	Level	Priority	Type
REQ-StRS06	StRS	MAN	FUNC
Name			
Reproducibility of a scientific workflow and data provenance			
Description			
EBRAINS users want to reproduce a simulation or a scientific workflow and to know where some data came from, or the recipe of a simulation.			

Table 31: Requirement StRS07

ID	Level	Priority	Type
REQ-StRS07	StRS	MAN	FUNC
Name			
Stored data			
Description			
Users should know what data has been stored.			

Table 32: Requirement StRS08

ID	Level	Priority	Type
REQ-StRS08	StRS	MAN	FUNC
Name			
Data assets of EBRAINS users			
Description			
EBRAINS users must be able to discover, download, upload, manage, create, share, and publish data assets and use them across EBRAINS services.			

Table 33: Requirement StRS09

ID	Level	Priority	Type
REQ-StRS09	StRS	MAN	USE
Name			
Easy-to-use			
Description			
EBRAINS should be easy to use for people of different backgrounds (scientists, developers) and different research fields. It should follow best practices in terms of usability.			

Table 34: Requirement StRS10

ID	Level	Priority	Type
REQ-StRS10	StRS	MAN	SUP
Name			
Data and Services security			
Description			
Available data must be securely accessed, stored, transferred and must also be protected against malicious attacks. Sensitive information could also be available, so security is very important.			

Table 35: Requirement StRS11

ID	Level	Priority	Type
REQ-StRS11	StRS	MAN	FUNC
Name			
High Availability			
Description			
EBRAINS should have a defined level of availability and should ensure that it is met.			

Table 36: Requirement SyRS01

ID	Level	Priority	Type
REQ-SyRS01	SyRS	MAN	FUNC
Name			
Container services			
Description			
EBRAINS application and service providers should be able to deploy containerized applications across all available Fenix sites.			

Table 37: Requirement SyRS02

ID	Level	Priority	Type
REQ-SyRS02	SyRS	MAN	FUNC
Name			
EBRAINS Identity and Access management services (AAI)			
Description			
Integration of Authorization and Authentication services for the management of EBRAINS users. Integration with Collaboratory IAM layers (HBP account) and mediation with Fenix/FURMS layer.			

Table 38: Requirement SyRS03

ID	Level	Priority	Type
REQ-SyRS03	SyRS	MAN	FUNC
Name			
Automated capture of provenance data and reproducibility			
Description			
Tools / services for automated capture of provenance metadata (VMs, Jupyter notebooks, HPC systems, neuromorphic systems) and for reproducibility.			

Table 39: Requirement SyRS04

ID	Level	Priority	Type
REQ-SyRS04	SyRS	MAN	FUNC
Name			
Automation of integration steps			
Description			
EBRAINS component/service providers must use services to facilitate the continuous integration of software components/services on EBRAINS.			

Table 40: Requirement SyRS05

ID	Level	Priority	Type
REQ-SyRS05	SyRS	MAN	FUNC
Name			
EBRAINS Resource allocation tracking services			
Description			
Services and tools to monitor infrastructure resources, components, and data operations			

Table 41: Requirement SyRS06

ID	Level	Priority	Type
REQ-SyRS06	SyRS	MAN	FUNC
Name			
EBRAINS Data management services			
Description			
Interfaces to information about data stored in Fenix archival data repositories.			

Table 42: Requirement SyRS07

ID	Level	Priority	Type
REQ-SyRS07	SyRS	MAN	FUNC
Name			
EBRAINS Workflow management platform services			
Description			
Workflow management system for scalable and reproducible workflows of EBRAINS services and applications.			

Table 43: Requirement SyRS08

ID	Level	Priority	Type
REQ-SyRS08	SyRS	MAN	FUNC
Name			
EBRAINS Federated JupyterHub service			
Description			
JupyterHub service that can adaptively utilise all Fenix sites and ensure high availability.			

Table 44: Requirement SyRS09

ID	Level	Priority	Type
REQ-SyRS09	SyRS	MAN	FUNC
Name			
EBRAINS image registry			
Description			
Registry for the available container images of EBRAINS.			

Table 45: Requirement SyRS10

ID	Level	Priority	Type
REQ-SyRS10	SyRS	MAN	PERF
Name			
Scalability of the resources			
Description			
Scalable & cost-effective: minimize future effort/costs for production, operation, administration and maintenance; scale according to demand, use resources efficiently, identify cost centres			

Table 46: Requirement SRS01

ID	Level	Priority	Type
REQ-SRS01	SRS	MAN	FUNC
Name			
EBRAINS Provenance capture API			
Description			
Provenance API to capture provenance metadata. Provenance metadata may be captured into the Knowledge Graph.			

Table 47: Requirement SRS02

ID	Level	Priority	Type
REQ-SRS02	SRS	DES	FUNC
Name			
Quality enforcement through SLA management			
Description			
EBRAINS RI should use an SLA (Service Level Agreement) model to monitor the quality of service (QoS) of the different operations.			

Table 48: Requirement SRS03

ID	Level	Priority	Type
REQ-SRS03	SRS	MAN	FUNC
Name			
Interoperability and interconnection of components based on standards			
Description			
Components can intercommunicate (exchange data) with standardised protocols and data formats.			

Annex IX: Software Quality Guidelines

The Software Quality Guidelines document was the output of EBRAINS Software Quality Working Group (ESQ-WG) containing guidelines and best practices for software quality. The text from the document has significantly influenced the section containing the guidelines for Software Quality. It has not been included in the current version of D5.3 in order to reduce the length of the document, and can be found at:

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

Annex X: Software Delivery Guidelines

The document that was the product of EBRAINS Software Delivery Working Group (ESQ-WG) containing guidelines and best practices for software delivery (Work In Progress). The text from the document has significantly influenced the section containing the guidelines for Software Delivery. It has not been included in the current version of D5.3 in order to reduce the length of the document, and can be found at:

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