Project title: European Federation for Cancer Images

Project acronym: EUCAIM

Grant Agreement: 101100633

Call identifier: DIGITAL-2022-CLOUD-AI-02

D6.4: Report on the implementation and technical validation of the Federated analysis infrastructure

Partner(s): UB, BSC

Author(s): Josep Ll. Gelpí, Carles Hernandez-Ferrer, Laura G. Antiga, Laia Codó

Contributor(s): José Almeida, Esther Bron, Alexadre Calado

Date of delivery: December 29th, 2025

Version: 1

Table of contents

1. Executive summary 4
2. Introduction 4
2.1. Motivation, and requirements 4
2.2. Overall architecture 5
3. Federated Execution Manager (FEM) 7
3.1. FEM-Orchestrator 7
3.1.1. Orchestrator Database 7
3.1.2. Messaging Layer 8
3.3. FEM-Client 8
3.3. FEM Workflows 9
3.3.1. Tasks (Atomic Actions) 9
3.3.2. Bundles 9
3.3.3. Workflows 10
3.3.4. Benefits for EUCAIM 10
3.4. FEM installation requirements 10
3.4.1. Hardware Requirements 10
3.4.2. Tier 3 Deployment Compliance 11
3.4.3. Clarifying Key Points 12
4. Software onboarding 12
4.1. Software onboarding process 12
4.1.1. Software card and bio.tools 13
4.1.2. Software Container 13
4.2. Legal Evaluation 13
4.3. Technical Evaluation 14
4.3.1. Technical Risk Review 14
4.3.2. Execution and Container Validation 14
5. Data Materialization 14
5.1. Interaction with EUCAIM CDM 15
5.1.1. The EUCAIM Clinical Data Model (CDM) 15
5.1.2. The EUCAIM File System Structure (FS) 15
5.2. The DMT from the FEM side 16
5.3. The DMT from the DH slide 17
5.3.1. Materializing datasets upon request 17
5.3.2. Constructing the EUCAIM-standard file structure 17
5.4. EUCAIM CDM, EUCAIM FS, and dataset checkers 18
5.4.1. extract-info-dataset 18
5.4.2. eucaim-cdm-federated-unbalance-checker 18
6. Real time Data holders monitoring 18
7. Federated Processing Dashboard 21
7.1. Components 22
7.1.1. OpenVRE 22
7.1.2. FEM-RUNNER 22

7.2. User Journeys....23
7.2.1. Journey A: Monitoring the Federated Network....23
7.2.2. Journey B: Launching a Federated Experiment....24
7.2.3. Journey C: Monitoring job and retrieving Results....28
7.3. Deployment....29
Annex 1. DH side launcher.json file....30
Empty launcher.json....30
Examples of filled lanugher.json....31
Annex 2. FEM Workflow definition....32
A2.1. FEM Task....32
A2.2. Bundle....33
A2.3. Workflow....34
A2.4. Example....35
Annex 3. Software revision....37
A3.1. Prohibited AI practices....37
A3.1.1. AI Act Article 5, Prohibited AI practices....37
A3.1.2. Prohibitions in the European Health Data Space (EHDS) Regulation....38
A3.1. Self-risk assessment....38
A3.1.1. General Risks....38
A3.1.2. Risks during AI model training....39
A3.1.3. Risks related to using an AI model trained on EUCAIM data....39
Annex 4. Registered software....40

1. Executive summary

This deliverable reports on the implementation, integration, and technical validation of the EUCAIM Federated Analysis Infrastructure, a core component of the project's mission to enable secure, large-scale cancer image analysis across distributed European data holdings. Building on the architecture defined in D6.1, WP6 has developed a production-ready Federated Execution Manager (FEM) that coordinates analytical and federated learning workflows across heterogeneous Data Holders while ensuring that patient data never leaves institutional boundaries.

The document details the current status and functionality of the FEM-Orchestrator, FEM-Client, and the associated workflow model, which now includes declarative definitions for Tasks, Bundles, and multi-node federated Workflows. These components enable flexible execution strategies, runtime assignment of nodes, and strong reproducibility guarantees, supporting both distributed analysis and complex federated learning protocols.

A comprehensive software onboarding process has also been implemented. It combines descriptive documentation, legal compliance checks under the AI Act and EHDS Regulation, and technical validation of containerised tools. This ensures that only compliant, well-documented, and technically robust software becomes available in the EUCAIM Toolbox and can be executed within the FEM ecosystem.

Given the central role of harmonised data, the deliverable further defines the EUCAIM data materialization pipeline. All *Data Holders* are required to expose harmonised clinical and imaging data according to the EUCAIM Common Data Model and File System structure. WP6 developed complementary Data Materialization Tools: one triggered by FEM to coordinate dataset preparation, and one deployed at each Data Holder to extract required metadata and generate the EUCAIM-standard dataset hierarchy from their local PostgreSQL ETL output. Additional dataset inspection and imbalance-checking tools are recommended as best practices prior to running federated workflows.

Finally, the deliverable describes the integration of monitoring and execution services into the Federated Processing Dashboard (OpenVRE), providing real-time visibility of node status, resource availability, and software deployment. Together, these developments mark a major milestone toward a secure, interoperable, and scalable federated infrastructure supporting secondary use of data of EU-wide cancer data. Outputs are not intended to directly drive individual patient management decisions without additional clinical validation and local governance..

2. Introduction

2.1. Motivation, and requirements

The EUCAIM project is building a large European-based infrastructure for the storage and analysis of cancer image data. This objective relies on two main pillars: i) the collection of a large dataset of images and ii) the development of a secure infrastructure for analysis of such images. Due to the sensitive nature of medical images, a significant part of the EUCAIM dataset will be subjected to restriction on the distribution and even on the analysis itself. For this reason, the analysis framework developed for the EUCAIM infrastructure should be designed in a way that transmission and access restrictions can be honoured following a "privacy-by-design" paradigm. To this end, EUCAIM's WP6 has been conceived to develop such analysis infrastructure based on a federated approach. This will allow to perform the envisioned analysis including AI training and inference while keeping data in their original location, under the specific regulations of each data site. Although some of the data available on EUCAIM will be accessible directly on the reference nodes through Secure Processes Environments (SPEs) (see D4.6),

the most restricted approach that serves any scenario is a federated approach. The overall description of the infrastructure and its components and the outline of the process followed was reported in EUCAIM's "D6.1. Design of the architecture and APIs and modules' specification of the federated analysis infrastructure"[1]. The present document reports on the present status of the development of the platform, including overall structure, components, and protocols to cover *Data Holders* and software onboarding. Part of this information, especially regarding documentation of end users is also available at EUCAIM's user guide[2] and Handbook[3]

The main assumptions taken to define the Federated Processing infrastructure are:

1. This infrastructure will work as part of the overall EUCAIM infrastructure and relies on a series of core services (developed at WP4). These services include Authentication and Authorization, Data and tools Catalogues, Negotiation for data access, and software container repositories.
2. The infrastructure will not decide on the internal organisation of data sites, or specific computational power (except for the minimum resources). Instead, we will assume a uniform data access procedure and rely on data sites to provide the necessary interfaces.
3. Authorization policies are managed by the Negotiator service and implemented at site level. Data sites will ensure that access requests' credentials are valid to perform the process.
4. Federated processing infrastructure and frameworks together with the software responsible for data processing and analysis will constitute the project's toolbox Software included will be annotated with the necessary metadata, including documentation, potential risks and deployment strategies.
5. Analysis tools will work in "federated/distributed" mode, i.e. they should perform the required task without requiring moving any original data outside of the site. Only analysis results will be transmitted to a central site.
6. Likewise, analysis tools will not require establishing incoming network connection to data sites. In the case that tools do require such connections, the necessary site-dependent arrangements could be made (e.g. VPNs), but these are not considered here.

2.2. Overall architecture

The high-level architecture of the proposed federated processing system involves a series of core services that coordinate the collaboration between the entities. These core services provide access to the remaining elements like user authentication and user authorization. The overall architecture was originally described in D6.1 and is summarized below in Figure 1.

[1] D6.1: Design of the architecture and APIs and modules' specification of the federated analysis infrastructure: https://cancerimage.eu/wp-content/uploads/2024/02/D6.1_final.pdf
[2] EUCAIM end user guide: https://eucaim.gitbook.io/enduserguide
[3] EUCAIM Handbook: https://eucaim.gitbook.io/handbook/

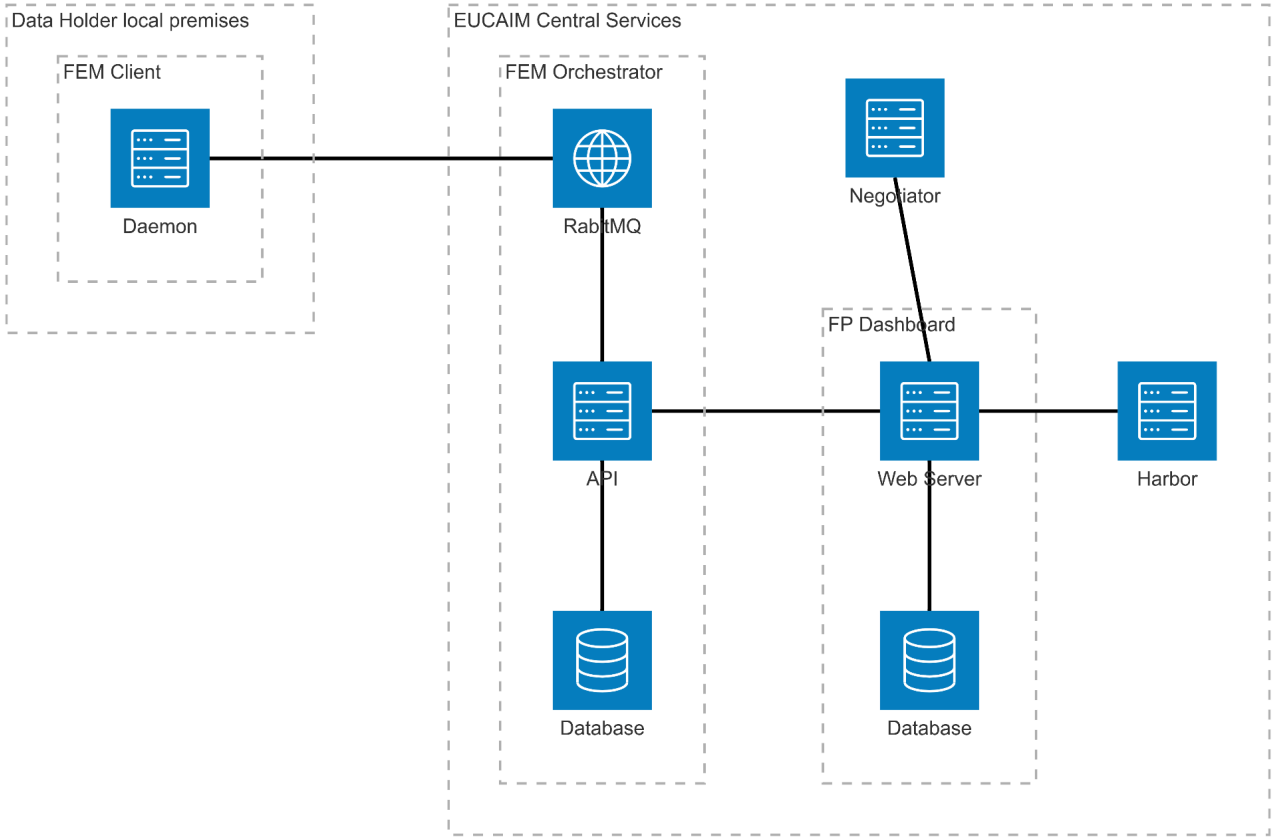

Figure 1. *depicts the internal architecture used to implement the federated processing platform. These components make use of services from the EUCAIM Central services such as the Negotiator and the Toolbox (Harbor).*

Federated Processing Dashboard (FP Dashboard). User interface that helps the final user define and configure a federated experiment. Consuming the information available from users, extracted from the Negotiator, it displays the datasets that the final user has been granted access to. Consuming the Toolbox (implemented with Harbor), the dashboard allows us to monitor which tools are applied on which datasets. The web interface also consumes the API from **FEM-orchestrator**, triggering the federated experiment.

FEM Orchestrator. The "core" of the federated processing infrastructure is the one that contains the information needed to run a software. It is also the one to distribute the different "tasks" to the nodes (aka, *Data Holders*) of the federations. It is also the one in charge of scheduling the multiple tasks of a federated experiment (aka, workflow). To do so, it relies on a message broker, a RabbitMQ server that centralizes all the communication on the federated network. The federated nodes will access their specific queues on the message broker to obtain the assigned tasks and report status that will end up to the final user.

FEM Client. A daemon is set up in each federated node that will connect to the message broker to obtain the assigned tasks and initiate the required software execution. If a mechanism is provided by the *Data Holders* (Docker, Kubernetes, batch queues, ...), the daemon may know the resources. The job, then, is submitted if the resources match or denied if the final user allocated a set of resources the *Data Holder* cannot provide.

3. Federated Execution Manager (FEM)

The **Federated Execution Manager (FEM)** is the component responsible for coordinating and executing federated experiments within EUCAIM, including both federated analysis and federated learning workflows. FEM is composed of two main elements: the **FEM-Orchestrator**, deployed centrally, and the **FEM-Client**, deployed locally at each *Data Holder* site. Together they enable secure, privacy-preserving execution of computational tasks without moving patient data outside institutional boundaries. The orchestrator exposes an API, manages experiment metadata and software descriptions, and handles messaging. The client runs as a lightweight daemon inside the *Data Holder* infrastructure, connects outward to the orchestrator, retrieves assigned tasks, and executes them using the site's preferred execution environment.

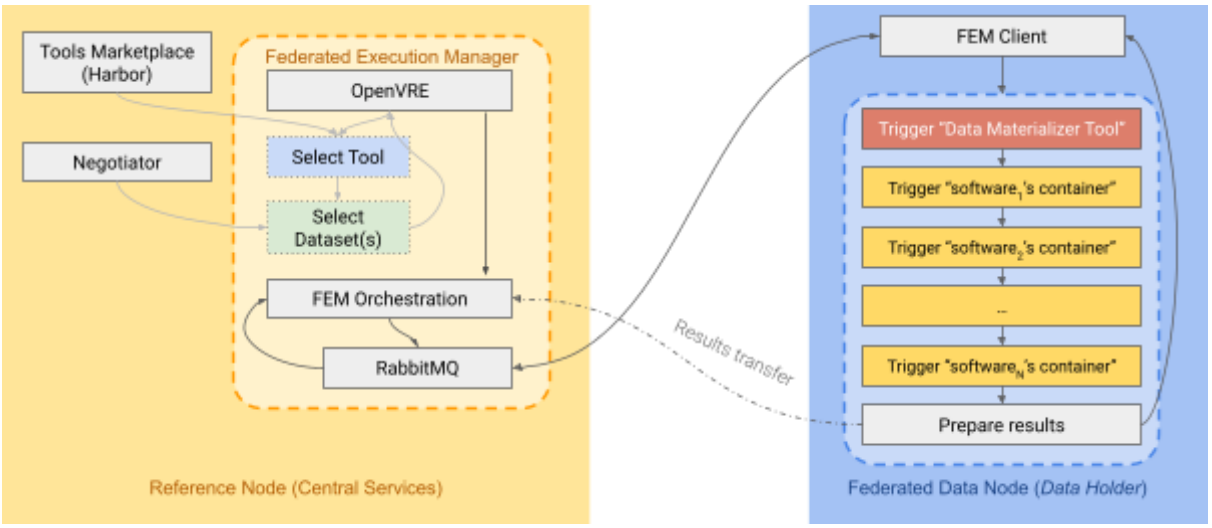

Figure 2. *High level description of Federated Execution Manager (FEM) and the interaction of its components: Dashboard, Orchestrator, communication channel (RabbitMQ) and Client. It also depicts the interaction of the Dashboard with EUCAIM's services as Tools Marketplace (Harbor) and Negotiator (data access).*

3.1. FEM-Orchestrator

The **FEM-Orchestrator** is the central coordinating component of the **FEM**. It exposes a REST API[4] through which federated experiments are defined, scheduled, and monitored, and it serves as the single point of coordination between all registered *Data Holders* participating in a federated workflow. The orchestrator manages the lifecycle of each federated execution step, including task preparation, parameter validation, job dispatching, collection of results, and status monitoring.

To support these functions, the orchestrator integrates three key subsystems:

3.1.1. Orchestrator Database

The **FEM-orchestrator** maintains a **MongoDB** database storing all metadata required to describe and execute software and workflows in a federated setting. Each registered software is described through a structured document containing:

[4] https://fl.bsc.es/eucaim-fem/API/v1/docs

- **Execution metadata:** how the software is executed, including command templates, execution mode, expected inputs, and produced outputs.
- **User-configurable parameters:** parameters that can be safely exposed to end users when preparing a federated experiment.
- **Image information:**
    - A **container image or set of images,** according to the needs of the analysis software, provided by the software provider, and validated by EUCAIM policies.
    - A mechanism allowing each *Data Holder* to define its own **local default image**, enabling use of a private registry or pre-validated images required by local IT policies.

This dual strategy ensures interoperability across the federation while respecting the heterogeneous infrastructure and security constraints of participating institutions.

3.1.2. Messaging Layer

The orchestrator includes a **RabbitMQ**[5] message broker that manages all communication with the **FEM-Clients**. It provides asynchronous, reliable, and scalable distribution of job requests, status updates, and execution logs. Importantly, communication is designed to be client-initiated (outbound only), ensuring that *Data Holders* do not need to expose inbound ports and can operate behind firewalls or restrictive network configurations.

3.3. FEM-Client

The **FEM-Client** is a lightweight software component installed within each *Data Holder*'s secure environment. Its purpose is to receive and execute federated tasks locally, ensuring that all computation happens inside the institutional perimeter and that no patient-level data leaves the site. The client operates as a long-running daemon and establishes an outgoing-only connection to the orchestrator's **RabbitMQ** server. Through this channel, it periodically pulls job requests addressed to the *Data Holder*, executes them using the local infrastructure, and pushes back status updates and results.

The **FEM-Client** is designed to be highly adaptable to the diversity of infrastructures across the EUCAIM network. It supports multiple execution backends, including:

- **Docker** or other OCI-compliant container runtimes
- **Kubernetes** clusters
- **SLURM** or other HPC job schedulers
- **Direct command-line execution** on the host, including **Custom scripts** defined by the *Data Holder*

All site-specific details are captured in a comprehensive configuration file, which allows institutions to fine-tune how each job is handled, define local container registries, specify resource constraints, and impose additional security controls.

FEM-Client Configuration File Template

The following template illustrates the structure of a typical **FEM-Client** configuration file. It is provided as a reference, and each Data Holder may adapt it according to its internal policies and infrastructure (see Annex 1).

[5] https://www.rabbitmq.com/

This configuration file enables each *Data Holder* to define its preferred execution environment while maintaining a consistent interface with the orchestrator.

3.3. FEM Workflows

FEM provides a unified mechanism to define, compose, and execute analytical workflows for distributed analysis, federated analysis, and federated learning. While the system is currently being customized for the needs of EUCAIM, the underlying workflow model is intentionally designed to be general, declarative, and portable, ensuring long-term interoperability with existing workflow standards such as **Common Workflow Language (CWL)**.

In short, FEM CWL-based workflow definition matches at the "task" level with CWL and almost matches at the "workflow" level, adding the "bundle" intermediate level that allows specifying a set of tasks to be run in a local, distributed, or federated way. In more detail, **FEM** workflows are structured around a three-level abstraction that cleanly separates *atomic computation*, *software logic*, and *full analytical pipelines*.

This architecture allows EUCAIM data holders and software providers to design reusable building blocks, describe complex multi-node execution patterns, and run the same workflow on different sets of nodes at execution time.

3.3.1. Tasks (Atomic Actions)

At the lowest level, **FEM** represents each atomic action as a **Task**, which corresponds to a **CWL** `CommandLineTool`. Tasks encapsulate:

- The executable command (and arguments).
- Input/output specifications.
- Runtime environment (e.g., Docker images, env variables, mounted volumes, resource requirements).
- Strong typing of inputs/outputs for validation and portability.

Tasks contain no federated logic: they do not know whether they run on a server, a client, or a distributed worker. This makes them fully reusable across workflows and execution contexts.

3.3.2. Bundles

A **Bundle** is a logical grouping of one or more **Tasks** that describes how they participate in a federated or distributed computation. **Bundles** define:

- The **roles** required (e.g., server, client, worker).
- The **execution mode** (federated, distributed, local).
- How **Tasks** map to **roles** (e.g., which task runs on clients, which on the server).
- Global and per-role input/output interfaces.
- Metadata required by the orchestrator (network requirements, deployment mode, etc.).

Bundles correspond to the notion of *tools* in earlier **FEM** versions but are now redesigned to support richer semantics and reusable declarative definitions.

This level is where federated logic emerges: the **Bundle** defines how many nodes (aka, Data Holders) participate, what each node is required to execute, and how data (aka, results of the tasks) flows across roles.

3.3.3. Workflows

A **Workflow** is the top-level **FEM** orchestration entity. It composes multiple **Bundles** into a complete analytical process and defines the execution graph (dependencies) between them. Workflows specify:

- The set of bundles to run and the order in which they must be executed.
- Workflow-level inputs and outputs.
- How workflow inputs map to bundle inputs.
- Data flow between bundles.
- **Execution node assignment**, done at runtime: roles (server, client, worker) are mapped to specific *Data Holders* when the workflow is launched.

This approach enables reusable analytical templates: the same **Workflow** (e.g., federated learning, distributed analysis) can be executed on any subset of EUCAIM nodes (aka, *Data Holders*) without modification, just customization of the parameters of the software in its **Tasks**.

3.3.4. Benefits for EUCAIM

This architecture enables:

- **Runtime assignment of nodes**, ensuring flexibility and allowing a single workflow definition to be reused in multiple clinical or technical contexts.
- **Consistent support for federated, distributed, and standalone modes**, defined declaratively at the bundle level.
- **Automatic translation to CWL-compatible structures**, ensuring portability and long-term maintainability.
- Future **graph-based execution visualisation**, simplifying understanding and debugging of complex analyses.
- Enhanced **modularity** and reuse across tasks and workflows.

Overall, **FEM** workflows offer a scalable and extensible framework for defining analytical processes across the EUCAIM federation, balancing portability, expressiveness, and strict separation of roles and responsibilities.

3.4. FEM installation requirements

The **FEM-Client** is designed to operate reliably across heterogeneous infrastructures, but certain minimum hardware and software requirements must be met to ensure stable participation in federated analysis or federated learning workflows. The following specifications outline the minimum (Option1) and recommended (Option 2) configurations for *Data Holders* deploying **FEM-Client**.

3.4.1. Hardware Requirements

Table 1 shows the two baseline hardware options are proposed. These have been selected to fulfill the needs of both data processing required to set a data holder site, and federated processing, covering minimum resources for federated AI training.

***Table 1**. Summary of the hardware requirement for FEM.*

Component	Option 1	Option 2	Description / Rationale
CPU	Minimum: 16 cores ≥ 1.8 GHz	Minimum: 12 cores ≥ 3.0 GHz	For nodes without GPU acceleration, a high-core-count server-grade CPU is essential for Second Prototype workloads. If not comparable by core count, aim for 24+ threads.
RAM	64 GB	64 GB	DDR5 is preferred for bandwidth. ECC memory is strongly recommended for stability, especially under continuous operation.
Motherboard	4+ RAM slots	4+ RAM slots	Verify chipset compatibility with the selected CPU. For DDR5, confirm motherboard support.
Storage	521 GB SSD (OS), NVMe M.2 Gen4 or SATA III	1 TB+ data drive, SSD or HDD	The OS should be installed on a fast SSD (NVMe preferred). Data storage requirements depend on dataset size: 1 TB minimum, with many DHs planning 2 TB+. SSD improves speed but is not mandatory.
GPU	NVIDIA Quadro	NVIDIA RTX 30 Series	Recommended for AI workloads. 12 GB VRAM+ preferred. Priority should be placed on maximizing Tensor Cores. Ampere or Volta architectures are ideal.
Operating System	Linux	Linux	Any modern Linux distribution is acceptable (Ubuntu recommended). Windows is not allowed, except in exceptional cases where Linux cannot be deployed.
Power Supply	Site-specific	Site-specific	Must provide sufficient wattage for chosen hardware, with extra headroom for future upgrades.
Internet Connection	≥ 100 Mbps	≥ 100 Mbps	Higher throughput is strongly recommended. Network stability directly affects federated training; insufficient bandwidth may prevent successful workflow execution.

3.4.2. Tier 3 Deployment Compliance

The following steps describe the standard procedure for deploying the **FEM-Client**, the component responsible for connecting the *Data Holder*'s infrastructure to the EUCAIM federated network.

Step 1: Express Your Interest

Data Holders begin the process by contacting the *FEM technical team* to indicate their intention to join the federated ecosystem.

Step 2: Initial Guidance

A member of the UB/BSC team will reply with access to the repository: **https://gitlab.bsc.es/fl/fem-client**

The repository README contains essential details, including:

- Prerequisites
- Installation instructions
- Configuration guidelines

Step 3: Credentials Delivery

Once the *Data Holder* is ready to deploy, the *FEM team* will provide the node-specific credentials required for secure communication with the orchestrator and RabbitMQ.

Step 4: Final Setup & Testing

After the **FEM-Client** is installed and configured, the technical team performs a validation session to verify:

1. **Network connectivity** with the orchestrator and message broker
2. **Execution capability**, ensuring the client can launch Docker-based tools under the site's infrastructure
3. **Data materialization readiness**, confirming that the FEM-Client can access the local dataset structure and sandbox environment

Once all tests pass, the node is approved to participate in EUCAIM federated workflows.

3.4.3. Clarifying Key Points

- The **FEM-Client** is typically installed on a single dedicated host or VM, isolated from internal hospital networks and capable of running Docker containers.
- The client requires only outbound connections to:
  - The **FEM-Orchestrator**
  - The **RabbitMQ** message broker
  - All communication is encrypted and authenticated via node-specific credentials.
- No inbound ports need to be opened.
- No peer-to-peer communication occurs between nodes.
- Data never leaves the host machine: only non-sensitive computational outputs (e.g., model weight updates) are transmitted.
- During installation, two local paths must be defined:
  - **$DATA_PATH (read-only)** → location of datasets materialized according to EUCAIM standards
  - **$SANDBOX_PATH (writable)** → location where tools will produce intermediate and final outputs
- All tools will run as Docker containers or Kubernetes pods are retrieved from EUCAIM's central registry, conforming to EUCAIM's container security and reproducibility requirements. An exception is done in case a *Data Holder* wants to host a mirror of the Tools Repository, allowing, then, to pull the containers and pods from local deployment.

4. Software onboarding

4.1. Software onboarding process

The EUCAIM software onboarding process defines a structured pathway that all software must follow before being accepted into the EUCAIM's ecosystem. It begins with the collection of descriptive, technical, and operational information from the software provider, establishing the foundation for subsequent evaluations. This initial stage ensures that each tool is sufficiently documented, classified, and prepared for integration, enabling WP3 and WP6 to perform the legal and technical validations described in the following subsections. Through this phased approach, EUCAIM ensures that only well-described, compliant, and technically robust software can progress toward deployment within the federated infrastructure.

4.1.1. Software card and bio.tools

As part of the software onboarding process, providers must complete a Software Card that describes their tool in a structured and comprehensive way. This card captures essential metadata such as the tool's purpose, functionality, inputs and outputs, technical requirements, expected behaviour, and any supported analytical modalities. The completed Software Card is then used to register the tool in **bio.tools**. Registering tools with EUCAIM label in the **bio.tools**'s domain ensures they can be automatically discovered as members of the project, supports consistent metadata across the federation, and aligns with FAIR principles by making software descriptions findable and reusable.

4.1.2. Software Container

Successful onboarding also requires that the software be packaged into a container image that meets EUCAIM's quality and interoperability standards. The "EUCAIM Software Packaging Guide"[6] outlines best practices for creating container images (typically using Docker) including a structured repository layout, secure Dockerfile construction, testing, and automated CI/CD pipelines to build, validate, and publish images to the EUCAIM registry.

Providers must include clear command-line interfaces, configure inputs and outputs cleanly, manage dependencies securely, handle runtime constraints, and follow FAIR principles to ensure the container is findable, accessible, interoperable, and reusable within the federated environment. Detailed guidance and examples for packaging, metadata descriptors, CI/CD integration, and descriptor files (such as CWL or Boutiques) are also provided in the EUCAIM End-User Guide for Software Providers[7], supporting consistent deployment of tools across analysis platforms.

A complete, end-to-end example of the software onboarding process was presented during the EUCAIM Contentathon (November 2025). In this session, the "EUCAIM CDM Federated Unbalance Checker" was used as a reference software to illustrate the full workflow: from the creation of a detailed Software Card and its registration in **bio.tools**, through the different stages of self-risk assessment and legal considerations, to the technical process of containerisation following EUCAIM packaging guidelines. This example provided a practical demonstration of how the onboarding protocol is applied in practice and served as a validation of the proposed methodology for integrating new software into the EUCAIM federated ecosystem.

4.2. Legal Evaluation

Following the **self-risk assessment**, all software undergoes a **legal evaluation** performed by **WP3**, focusing specifically on compliance with the AI Act Article 5 (Prohibited AI Practices) and the European Health Data Space (EHDS) Regulation (see Annex 3.1).

The **Prohibited AI Practices**, such as subliminal manipulation, exploitation of vulnerabilities, social scoring, predictive policing, untargeted facial recognition, biometric categorisation, among others, are listed in the onboarding guideline and a mandatory step of the onboarding process. **WP3** reviews the provider's answers in the section of the onboarding form and confirms that:

- The software is **not intended** for any prohibited purpose under the AI Act.

[6]EUCAIM Software Packaging Guide: https://dashboard.eucaim.cancerimage.eu/eucaim_software_packaging_guide.pdf
[7] User Guide for Software Providers https://eucaim.gitbook.io/enduserguide/5-userguideswproviders

- The software does **not violate EHDS restrictions** related to discrimination, detrimental decision-making, advertising, harmful products, or ethical conflicts.

Software that fails this evaluation cannot proceed to the technical stage.

4.3. Technical Evaluation

The final step of the onboarding protocol is a **technical validation**, carried out by **WP6** with support from **WP4** and **WP5**. This stage focuses on confirming that the software is safe, reproducible, and functional within the EUCAIM execution environment.

It includes two complementary components:

4.3.1. Technical Risk Review

WP6 reviews the provider's **self-risk assessment** (see Annex 3.2), ensuring that all mandatory risks have been addressed and that identified issues are compatible with federated execution. This step focuses on operational aspects such as resource usage, dependencies, traceability, potential failure modes, or risks introduced during AI model training (when applicable).

4.3.2. Execution and Container Validation

Once the software has passed both descriptive and legal checks, the provider uploads its containers (runner, aggregator, or multi-role images) to the **EUCAIM's Toolbox** (Harbor), following WP4's packaging guidelines.

WP6 then performs:

- **Local run tests** to verify correctness of inputs, outputs, and expected behaviour in EUCAIM Reference Node.
- **FEM integration tests**, where the software is registered in the FEM database, configured as a Task/Bundle, and executed in a controlled federated workflow.
- Verification that the software respects the expected execution model (batch, case-based, client/server, etc.) and works with the FEM-Client runtime environments (Docker, Kubernetes, Slurm, etc.).
- Validation of logging, traceability, and reproducibility.

This final stage confirms that the software can be safely integrated into the EUCAIM federated ecosystem. Once approved, it will be fully available at the **EUCAIM Toolbox** (Harbor) and scheduled in analytical workflows.

5. Data Materialization

The EUCAIM data materialization process ensures that all datasets used in federated or distributed analytical workflows are exported from the *Data Holder*'s internal PostgreSQL database—produced after executing the EUCAIM ETL—and transformed into the standard **EUCAIM File System (FS)** structure. All *Data Holders* participating in federated experiments are expected to have harmonized their data using the **EUCAIM Extract-Transform-Load (ETL)** pipeline and stored the resulting data in a **PostgreSQL database** compliant with the **EUCAIM Common Data Model (CDM)**, derived from the mCODE

specification and the EUCAIM Hyperontology, as stated in the working document "Data Harmonization meeting minutes".[8]

Data materialization therefore acts as the bridge between harmonized relational data and the file-based dataset structure required for **FEM** workflows, enabling consistent interpretation, indexing, and distributed/federated processing.

5.1. Interaction with EUCAIM CDM

All *Data Holders* must materialize datasets according to two mandatory EUCAIM standards, described in the following subsections.

5.1.1. The EUCAIM Clinical Data Model (CDM)

Defined in the working document "EUCAIM mCODE-based CDM Spreadsheet (v3.1)"[9] and reaffirmed in harmonization discussions[6], the CDM dictates the structure, tables, and controlled vocabularies for clinical and imaging metadata. The ETL process outputs a **PostgreSQL database** structured according to this model.

Additionally:

- A **mandatory clinical metadata view** and a **mandatory imaging metadata view** must be materialized as CSV files for each dataset according to harmonization discussions[6].
- These views correspond to the **minimum clinical** and **minimum imaging** requirements described in Deliverables D4.4[10] and D5.6[11].
- **WP5**'s working document "Guidelines for imaging datasets"[12] provide detailed instructions on extracting DICOM metadata, linking studies to clinical episodes, mapping series descriptions to the EUCAIM vocabulary, and generating mandatory metadata tables.

5.1.2. The EUCAIM File System Structure (FS)

The standard folder hierarchy, harmonization discussions[6], must be followed exactly. A materialized dataset must contain a series of files and folders as depicted in **Table 2.**

[8] EUCAIM Data Harmonization and Interoperability Meeting Minutes: https://docs.google.com/document/d/1yPAkDjL0F8OByidMuSffzhuI_MQU6eSwFbmqmYe2SAg/edit?usp=sharing
[9] EUCAIM_CDM_mCODE_based_v3.1: https://docs.google.com/spreadsheets/d/1ox9PdvfCDxpDmEnFzC1M6OFhUhXpjQzg/edit?usp=sharing&ouid=114064821881985537753&rtpof=true&sd=true
[10] D4.4: Final rules for participation report: https://docs.google.com/document/d/15Q73-9fYZZjFSOJH3Aeb2jLSFzngG6cD/edit?usp=sharing&ouid=114064821881985537753&rtpof=true&sd=true
[11] D5.6. Minimum Data Federation and Interoperability Framework: https://drive.google.com/file/d/1EXcXN1OnKdl8JaOrsOBpzdaLMBfR6HoA/view?usp=sharing
[12] Guidelines on how to provide imaging-related information for dataset: https://docs.google.com/document/d/1mnTkf2fvERgaRyQPDFebZHLwB8aBRaIZRkwlMBr3ZXQ/edit?usp=sharing

Table 2. *Structure of a materialized dataset.*

user-data-space:ro

```
/DatasetID/
    ├── clinical_mandatory_view.csv <-- View with mandatory clinical data
    ├── imaging_metadata_view.csv   <-- View with mandatory imaging metadata
    ├── clinical_data               <-- full clinical data
    │    ├── patient.csv
    │    ├── primary_cancer_condition.csv
    │    └── tumor_marker_test.csv ...
    └── imaging_data                <-- Hierarchical structure for imaging files
        ├── patient-uid1
        │    ├── study-uid1/
        │    │    ├── series-uid1/
        │    │    │    ├── DICOM file 1
        │    │    │    ├── DICOM file 2
        │    │    │    └── DICOM file ...
        │    │    └── ...
        │    └──  study-uid2/ ...
        ├── patient-uid2
        │    └── ...
        └── ...
```

This ensures:

- Consistent dataset interpretation by FEM tools
- Reproducibility of analytical workflows
- Seamless interoperability across EUCAIM nodes
- A direct mapping between CDM metadata and the corresponding imaging artefacts

In summary: All *Data Holders* must materialize their datasets using both the EUCAIM CDM and the EUCAIM FS standards, ensuring complete consistency across the federation.

5.2. The DMT from the FEM side

The **FEM**-side **Data Materialization Tool (DMT)** is a software component executed directly on behalf of the end user and typically integrated in the FEM dashboard or workflow submission process. Its purpose is to coordinate and request dataset materialization tasks across participating Data Holders.

Based on the working document "EUCAIM Data Materialization definition v2"[13], the DMT on the FEM side performs the following responsibilities:

1. Receives dataset selections from the user, stemming from the Negotiator via **FEM-orchestrator**.
2. Generates a materialization plan, specifying:
    - Which datasets must be materialized
    - Temporal storage location
3. Sends materialization requests to *Data Holders* through internal communication channels (usually DMT on DH side).

[13] EUCAIM Data Materialization definition v2: https://docs.google.com/document/d/1h5-tOBrwZKeFsoLMLvBXKZa7A6hKriCF7zq5OiobUng/edit?usp=sharing

4. Monitors materialization progress, reporting status to the **FEM-orchestrator** (and, therefore, to the end user).
5. Ensures the resulting file structure adheres to: **EUCAIM CDM** and **EUCAIM FS**.

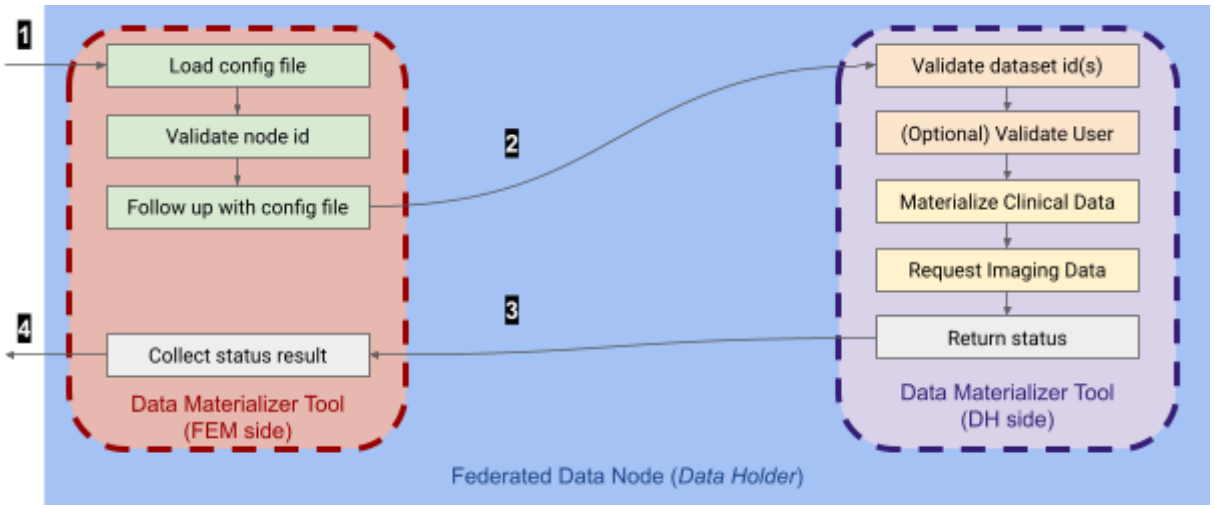

Figure 3. *Conceptualization of the data materialization involving the Data Materializer Tool (DMT). The request of a dataset (1) triggers the execution of the DMT at the FEM side. This follows the procedure described in its configuration file for each requested dataset. The configuration file described the process (2) required by the Data Holder, triggering the DMT on the DH side. The typical steps on the DH side include the validation of the dataset id(s), the validation of the user requesting the datasets, and the materialization and request of clinical and imaging data. Once the datasets have been materialized, or could not be materialized, the status is collected on the FEM side (3) and sent back to the orchestrator to be displayed to the final user (4).*

5.3. The DMT from the DH slide

Following discussions with **WP5**, **WP6** has initiated the development of a ***Data Holder***-side **DMT**. This component is deployed directly at the Data Holder and interfaces with the local PostgreSQL CDM database produced by the ETL process.

5.3.1. Materializing datasets upon request

When the FEM-side DMT issues a materialization request, containing the dataset identifiers selected by the user from the Negotiator, the DH-side DMT:

- Connects to the local PostgreSQL
- Executes the required SQL queries to extract:
  - Mandatory clinical metadata
  - Mandatory imaging metadata (to the PostgreSQL or to other services)
- Retrieves DICOM files according to the patient + study + series structure required by the EUCAIM FS specification.

5.3.2. Constructing the EUCAIM-standard file structure

The software produces exactly the structure defined in section "5.1.2. The EUCAIM File System Structure (FS)". Which can be summarized as:

- `/DatasetID/clinical_mandatory_view.csv`
- `/DatasetID/imaging_metadata_view.csv`

- `/DatasetID/clinical_data/*.csv`
- `/DatasetID/imaging_data/<patient>/<study>/<series>/DICOM`

5.4. EUCAIM CDM, EUCAIM FS, and dataset checkers

Although not yet integrated as an automated step within FEM workflows, it is strongly recommended that final users execute two diagnostic tools at the beginning of any federated or distributed workflow. These tools help assess dataset completeness, internal consistency, and potential imbalances that may affect downstream analyses or federated learning performance.

In addition to technical checks, basic clinical plausibility checks for key variables (e.g., completeness/consistency of stage, treatment line, outcomes when available) are highly encouraged to be taken into consideration by final users. The goal is to reduce any possible clinically misleading analyses.

5.4.1. `extract-info-dataset`

Repository: https://github.com/EUCAIM/extract-info-dataset

This tool performs a systematic inspection of a materialized dataset, extracting high-level statistics and structural information, including:

- Dataset size and composition
- Distribution of patients, studies, and imaging modalities
- Presence and completeness of mandatory metadata views
- Basic consistency checks between CDM tables and the imaging file hierarchy

The tool is particularly useful for verifying that materialized datasets comply with the **EUCAIM CDM** and the **EUCAIM FS** before they are consumed by workflows.

5.4.2. `eucaim-cdm-federated-unbalance-checker`

Repository: https://gitlab.bsc.es/fl/eucaim-cdm-federated-unbalance

This software evaluates potential cohort imbalance across federated nodes. It computes harmonized statistics by running:

- A runner module locally on each *Data Holder*
- An aggregator module that combines results to produce a global unbalance report

Current checks include distribution of clinical variables across nodes: age, sex, and ethnicity (developers are open to include extra covariables according to EUCAIM CDM). These diagnostics are especially important for federated learning workflows, where unbalanced data may significantly distort gradient aggregation or model generalizability.

6. Real time Data holders monitoring

FEM-orchestrator keeps an updated status of *Data Holder*'s sites. To this end, **FEM-orchestrator's API** provides two specific endpoints. First, `/hosts/heartbeat/{node_id}` monitors the state of the connection between FEM clients and the RabbitMQ broker. This is done locally at the **FEM-orchestrator**

site, querying RabbitMQ to avoid network delays. The response from heartbeat (example below ***Table 3***) includes technical details of the connection to RabbitMQ (see 3.1.2. Messaging Layer) and its state.

Table 3. *Output from heartbeat endpoint for node BSC.*


```
"BSC": {
    "user": "BSCrmq",
    "node": "BSC",
    "host": "212.128.226.71",
    "port": 51536,
   "state": "running"
}
```


`Heartbeat` is used as an immediate test, prior to executing any tool in a distributed manner, and helps to update the number of available clients for those analysis software (and especially federated learning tools) that rely on datasets located in multiple nodes (aka, *Data Holders*).

The second monitoring tool is triggered by the `/hosts/resources/{node_list}` endpoint. In this case, execution is done by a specific monitoring software installed at **FEM-client**, and implies a true distributed operation. This tools monitors at the execution time:

- Host total resources: Number of CPUs, Total Memory, available GPUs and type.
- Installed software: Presence and version of Docker engine, installed container images (with checksum control for version check).
- Installed SSL certificates and expiration dates.
- Immediate resources: Available CPUs and Memory.
- Data Availability: Presence of expected /data and /sandbox folder and contents.

A typical response from this endpoint is shown in ***Table 4***.

Table 4. *Information conveyed by the resources endpoint.*


```
{
  "BSC": {
    "docker_version": "Docker version 28.5.1, build e180ab8",
    "public_ip": "212.128.226.71",
    "gpu_check": {
      "gpu_detected": false,
      "message": "No Nvidia GPU or driver error"
    },

    "resources_check": {
      "CPU Count": 8,
      "CPU Percent": 1.2,
      "Memory Available": "28.69 GB",
      "Memory Total": "31.34 GB",
      "GPU Count": 0,
      "Has Internet": true
    },
    "paths_check": {
```

```
      "sandbox_path_exists": true,
      "data_path_exists": true,

      "sandbox_contents": [
        "6ad81195-4946-40d2-9b6f-4f09135695ec",
        "Bf0b4f00-ee26-4438-9d01-31d92601758a",
        "Flcore_superlink",
        "9d552988-7959-4e5d-bb70-61a801b6ce86",
        "Flcore",
        "Round-3-weights.npz",
        "Eucaim-fl-dl-server",
        "Img",
        "Eucaim-fl-dl-client",
        "Round-2-weights.npz",
        "Log.txt",
        "0e8f509e-3d43-4346-96da-aa744e7e8a75",
        "Round-1-weights.npz",
        "E51f0d40-a011-4a55-bd8d-6c009fa830e2"
      ],
      "data_path_contents": [
        "Clinical.csv",
        "site_2_BSC.zip",
        "Test_study",
        "3362-2045-1944"
      ]
    },
    "docker_images_installed": [
      {
        "repository": "registry.gitlab.bsc.es/fl/eucaim-fl-dl-server",
        "tag": "v2.0",
        "image_id": "2b6ccffe8c9c",
        "Sha256_digest":
"Sha256:7bafb74babeb76e697479d171578cfdb90292b367a46c153c4c4a13dd7949f44",
        "warning": "Hash mismatch: expected image_hash_value, found
Sha256:7bafb74babeb76e697479d171578cfdb90292b367a46c153c4c4a13dd7949f44"
      },
      {
        "repository": "registry.gitlab.bsc.es/fl/flcore",
        "tag": "demo",
        "image_id": "03d6c41d6742",
        "sha256_digest":
"sha256:8e4ed82ab11fda99cbe71861eb820e4dc2476003cc8f7312b5524d26eb2112bb"
      },
      {
        "repository": "registry.gitlab.bsc.es/fl/flcore",
        "tag": "demo",
        "image_id": "03d6c41d6742",
        "sha256_digest":
"sha256:8e4ed82ab11fda99cbe71861eb820e4dc2476003cc8f7312b5524d26eb2112bb"
      },
      {
        "repository": "federated-unbalance-runner",
        "tag": "latest",
        "image_id": "5f2cf434b6bb",
```

```
        "sha256_digest": "<none>",
        "warning": "Hash mismatch: expected , found <none>"
      },
      {
        "repository": "federated-unbalance-aggregator",
        "tag": "latest",
        "image_id": "6cdd6930344a",
        "sha256_digest": "<none>",
        "warning": "Hash mismatch: expected , found <none>"
      }
    ],
    "pem_certificates": {
      "server_cert.pem": {
        "expiry_date": "2026-12-30 13:42:46 UTC",
        "status": "Valid"
      },
      "BSC_client_cert.pem": {
        "expiry_date": "2026-12-30 13:43:00 UTC",
        "status": "Valid"
      },
      "rootCA_cert.pem": {
        "expiry_date": "2026-12-30 13:42:46 UTC",
        "status": "Valid"      }
     }
  }
}
```


Combined information of both heartbeat and resources check is also accessible from the Federated Processing Dashboard (see section 7. Federated Processing Dashboard (OpenVRE)).

7. Federated Processing Dashboard

The **Federated Processing Dashboard** (FP Dashboard), provides the graphical user interface (GUI) through which EUCAIM researchers and data scientists (aka, end users) configure, launch, and monitor federated experiments. Built on top of the OpenVRE framework, it offers a unified and secure environment that connects researchers with the **FEM-orchestrator** and some of the EUCAIM central services. Its goal is to make complex federated workflows accessible to end users through a streamlined, GUI-based experience, while enforcing authentication, authorization, and dataset-access constraints.

The **FP Dashboard** plays three complementary roles within the platform:

- User interface for federated experiments execution by exposing a simplified end-to-end workflow for assembling, launching, and reviewing federated experiments.
- Middleware to the FEM-orchestrator by translating all user actions into authenticated REST calls to FEM, which manages the scheduling and execution across FDNs.
- Integrator of required EUCAIM services by interacting with Life Sciences AAI (OIDC login), the Negotiator (dataset authorization), the Data Catalogue (dataset metadata), and FEM monitoring endpoints (node health and resources).

7.1. Components

The **FP Dashboard** is composed of two core modules: OpenVRE and the FEM-RUNNER. Together, they provide the user interface, service connectors, and execution bridge required to run federated experiments within the EUCAIM federated network.

7.1.1. OpenVRE

Repository: https://github.com/EUCAIM/fed-computing-dashboard

OpenVRE is an open-source platform for building and operating Virtual Research Environments (VREs). It provides a modular backend and a customizable web interface designed to integrate analytical tools, data services, and compute resources under a unified environment. The platform manages user sessions, job scheduling, access to data resources, and orchestration of containerised or HPC-based tools.

For EUCAIM, OpenVRE has been extended to support the specific requirements of federated experiments:

- **Customized GUI.** The interface has been adapted to EUCAIM's look and feel and expanded to include functionality needed for federated experiments execution. This includes a real-time view of the federated network (see 6. Real time Data holders monitoring) and guided selection of participating nodes based on their dynamic availability and tool installation status.
- **Integration with EUCAIM AAI.** Authentication is handled through Life Sciences AAI (EUCAIM VO) using OIDC. This enables seamless propagation of user identity and authorization through OAuth2 tokens across EUCAIM services - here, the **FP Dashboard**, **FEM-orchestrator**, and Negotiator. Where stricter scopes are required, the Dashboard performs a transparent access-token exchange.
- **EUCAIM data-service connector.** A dedicated connector has been implemented to allow browsing and selection of datasets for which the user has already been authorized. Dataset URIs are currently retrieved from **FEM-orchestrator** endpoint, but the connector is designed to be eventually interoperable directly with the Negotiator as well as the EUCAIM public catalogue (Molgenis) to improve dataset exploration.
- **Execution bridge.** A dedicated **FEM-RUNNER** component has been developed as the OpenVRE executor to interact programmatically with the **FEM-orchestrator**. Through this dedicated adapter, OpenVRE remains decoupled from **FEM-orchestrator** internals while ensuring reliable, traceable, and reproducible interaction with the federated executions (see next section).

7.1.2. FEM-RUNNER

Repository: https://gitlab.bsc.es/fl/AI-Dashboard-FEM-runner

The FEM-RUNNER is a lightweight Python component embedded within the FP Dashboard's backend. It encapsulates all communication with the FEM-orchestrator's API. Its primary function is to translate user actions performed in the GUI into the orchestrator's API calls and to manage the lifecycle of federated experiment submissions. **Table 5** summarizes the sequence of actions carried out by **FEM-RUNNER** on each federated experiment run.

Table 5. *FEM-RUNNER connection to FEM-orchestrator endpoints used during a federated experiment run.*

Step	Step Description	FEM Orchestrator API Endpoint(s)
Node Heartbeat Check	Verify that each participating node is connected to RabbitMQ and responsive.	`GET /hosts/heartbeat?client_nodes={node_id}`
Node Resource Check	Retrieve dynamic node information: CPU/RAM availability, GPU presence, Docker version, dataset/sandbox paths.	`GET /hosts/resources?client_nodes={node_name}` `GET /hosts/resources?client_nodes={node_list}`
Submit Federated Run	• Build the JSON payload specifying workflow definition, tool ID, datasets URIs, parameters, and node assignment that vary at runtime. • Submit the federated experiment to FEM, triggering workflow resolution and dispatching.	`POST /workflows/run_workflow`
Poll Execution Status	Periodically query FEM for global and per-node job status (queued, running, completed, failed).	`GET /execution/status/{exec_id}`
Fetch Node Logs	Live print of the logs of the execution (all nodes involved in a run).	`/execution/logs/{exec_id}`
Retrieve Results (Logs)	Obtain final outputs generated by the federated workflow (e.g., model, metrics, reports).	`/data/download_files?execution_id={exec_id}` `/data/download_files?execution_id={exec_id}3&file={file_name}&node={node_name}`

7.2. User Journeys

FP Dashboard supports multiple user journeys. The three relevant ones for EUCAIM are (A) Monitoring the federated network, (B) Launching a federated experiment, and (C) Monitoring Job and Inspecting Results.

7.2.1. Journey A: Monitoring the Federated Network

The **FP Dashboard** provides a dedicated view for monitoring the EUCAIM Federated Network to assess network readiness prior to launching federated analyses. It is accessible from the main navigation menu under “EUCAIM Network”. This view lists all participating nodes (*aka* Data Holders) and presents their current operational status.

Information displayed on this page is obtained from the real-time monitoring endpoints exposed by the **FEM Orchestrator** (see 6. Real time Data holders monitoring). To ensure responsiveness, cached monitoring data is displayed immediately upon page load and refreshed at the beginning of each user session. A manual Reload action is also available to request an on-demand update. To avoid necessary delay, resources endpoint is only executed after a successful heartbeat call (online/offline).

The monitoring view reports the following information for each Data Holder:

- Online/offline status of Data Holders

- Data Holder response time
- Resource summary, including CPU, RAM, GPU, and Docker version
- Tool readiness checks, including Docker image availability and checksum validation
- SSL/TLS certificate validity

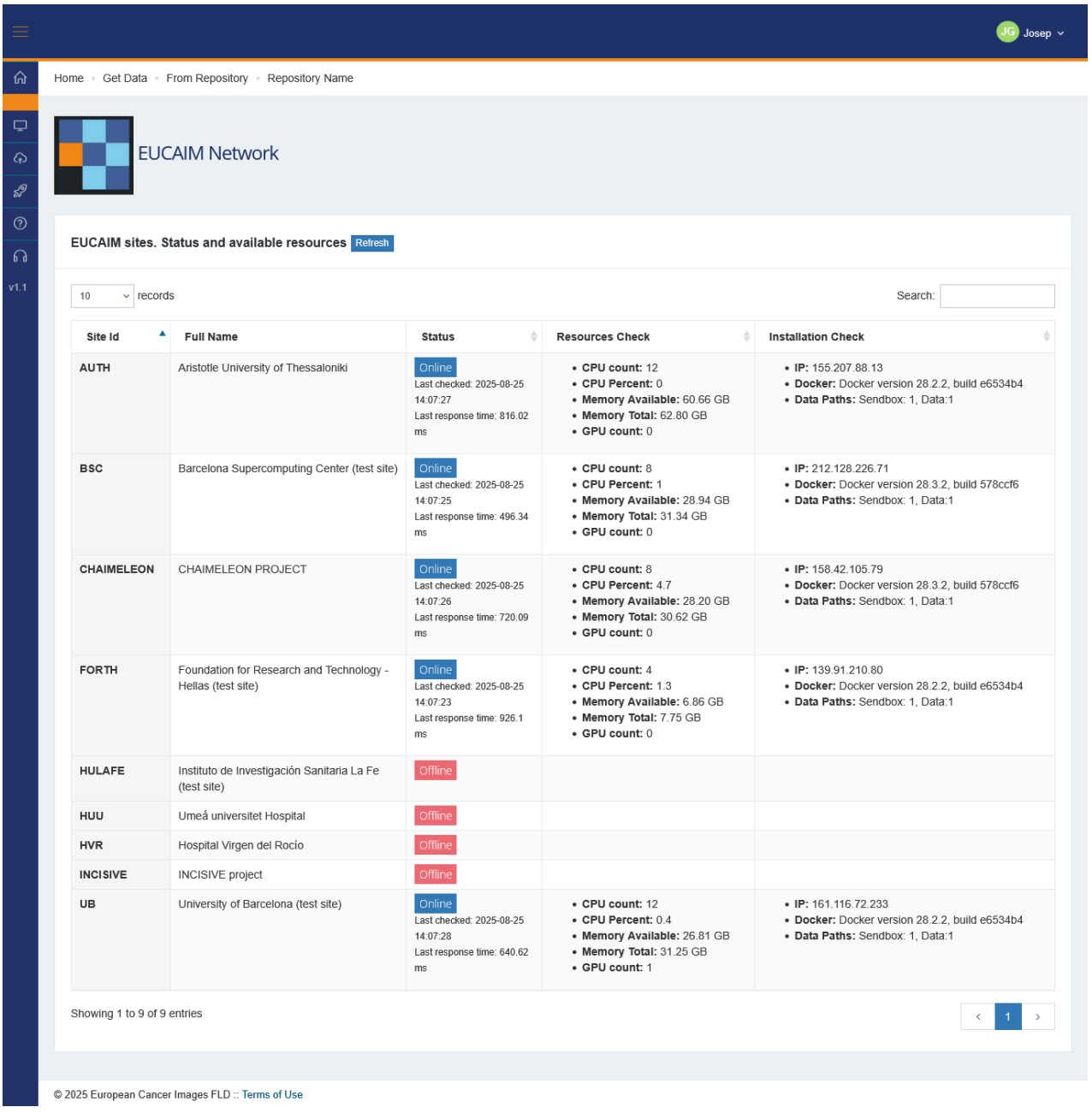

Figure 4. *The EUCAIM federated network as seen at the FP Dashboard.*

7.2.2. Journey B: Launching a Federated Experiment

This section covers a developed end-to-end example of a complete federated analysis including the use of the federated processing dashboard to trigger a FEM execution. The steps covered are:

1. Login in the FP dashboard using LS-Login. Users already logged in EUCAIM need not to issue their credentials as Single-Sign-On is in place. The first login will require to complete the user registration to generate the FP dashboard internal user.
2. Description of FP dashboard interface
3. Selection of data, tool and details of the federated execution (including selection of sites involved)
4. Launch of the execution

Login via Life Sciences AAI

The user authenticates through the EUCAIM Virtual Organisation using OIDC. The OpenVRE frontend redirects to the LS-AAI login page, and upon success, identity and access tokens are stored in the backend for subsequent calls to integrated EUCAIM services (e.g. **FEM-orchestrator**, Negotiator).

(a) (b)

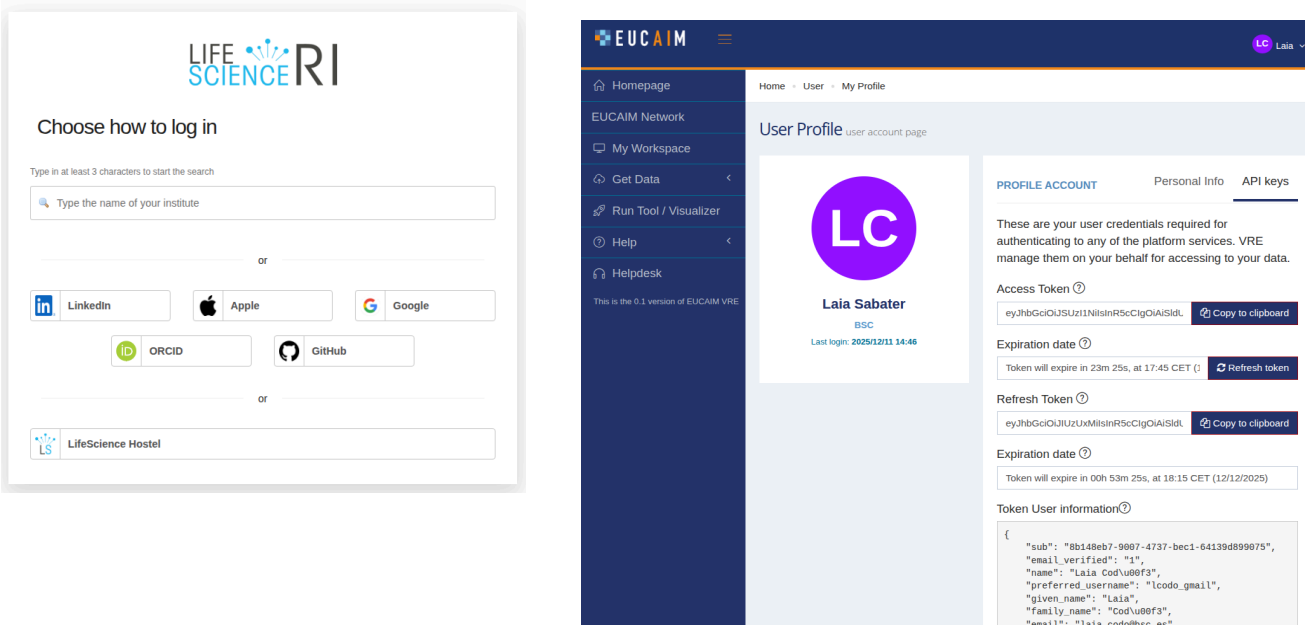

Figure 5. *(a) FP Dashboard login via Life Science AAI; (b) FP Dashboard profile page displaying OIDC issued tokens*

Home page

After login, the homepage displays the list of federated software (FEM Workflows and FEM Bundles) as tiles. Each tile offers a brief description of the tool as well as two shortcuts for either running the software (directly opening the configuration form) or learning more about it by opening the software-specific documentation.

The left navigation menu provides access to:

- **My Workspace**: user's input files, results, and available operations
- **Get Data**: mechanisms to upload or register data
- **Run Tool / Visualizer**: searchable list of all available tools
- **Helpdesk**: dashboard documentation and ticket-based support
- **Admin**: admin-only features

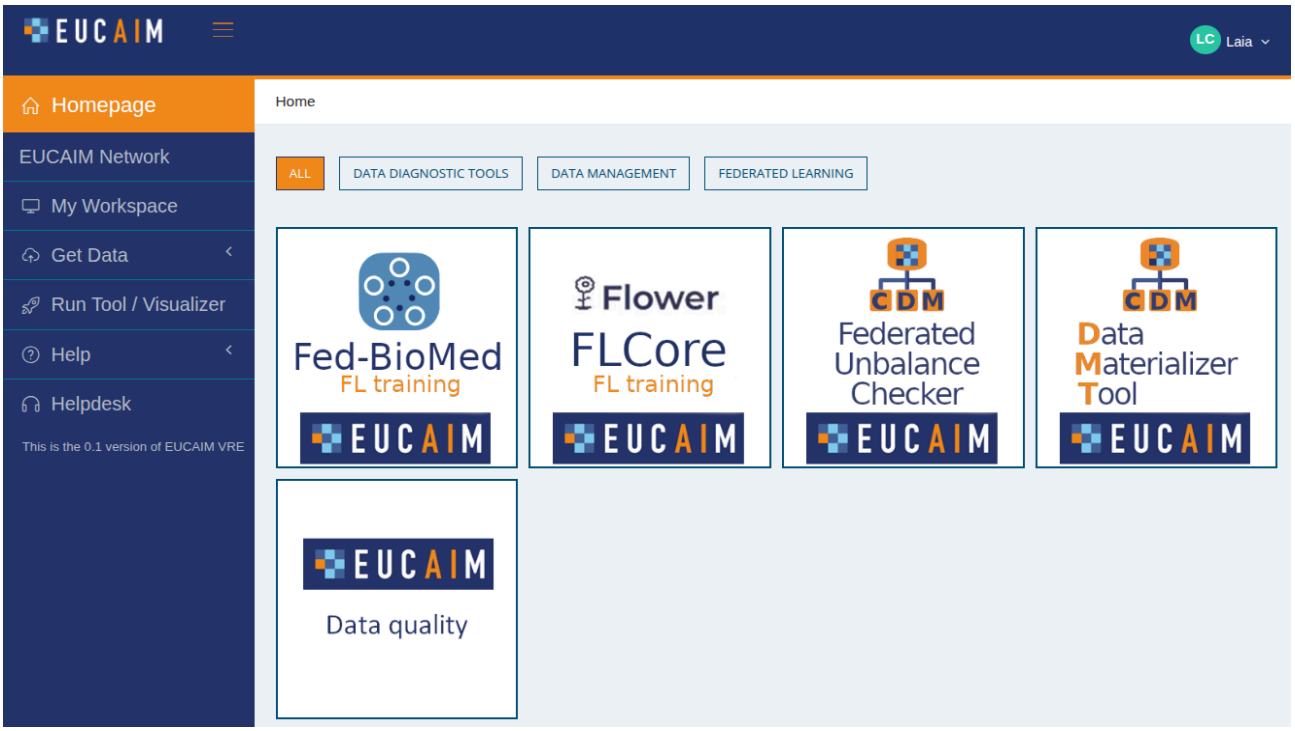

Figure 6. *FP Dashboard home page*

Dataset selection

Input data must be available in the user workspace before it can be bound to a software:

- User-provided files can be uploaded via Get Data → Upload Data.
- Federated datasets from EUCAIM appear under Get Data → My Datasets after successful negotiations in the Negotiator. Selecting a dataset registers its URI (`node_id:node/dataset/path`) into the workspace so it can be used as input.

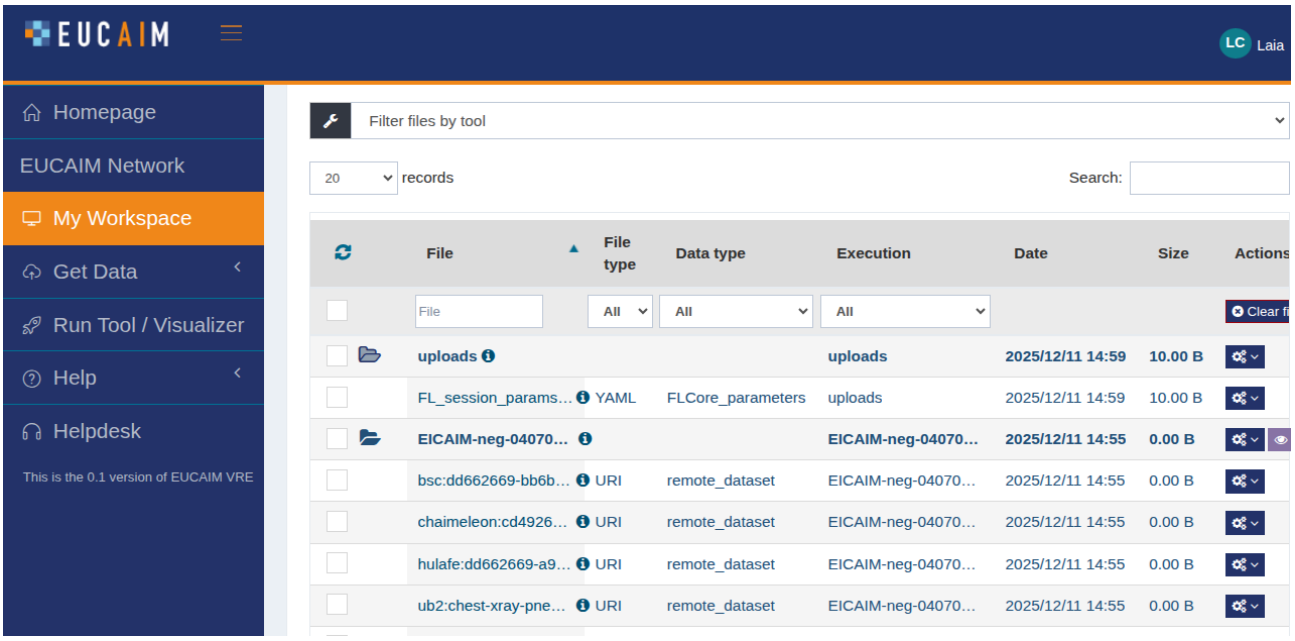

Figure 7. *Remote dataset references (URIs) imported at the FP Dashboard workspace ready to be used in a federated run*

Tool Selection

Tools can be selected in multiple ways:

- From the homepage (tiles).
- From the searchable list under Run Tool / Visualiser.
- From the Workspace: when users have already selected specific inputs, the Dashboard filters the available tools and presents only those compatible with the metadata of the selected files (file type, format, etc.)

Federated Run configuration

The FP Dashboard presents a structured configuration form with three blocks:

- **Project Settings.** Includes FP Dashboard's specific metadata such as "Execution Name". A dedicated workspace folder is created for the run, storing all generated files and logs.
- **Execution Settings.** Allows configuration of FEM-related parameters, including:
  - Generic execution parameters (e.g. job timeout).
  - Task- or container-specific arguments (only whitelisted parameters are exposed for security).
  - FDN selector: choose participating nodes (aka, *Data Holders*). The selector dynamically enables only nodes that are currently online, approved for the selected software, and have the required container (version and consistency check) available.
- **Input Settings.** Lets the user select which workspace files or dataset URIs will be bound to the tool inputs.

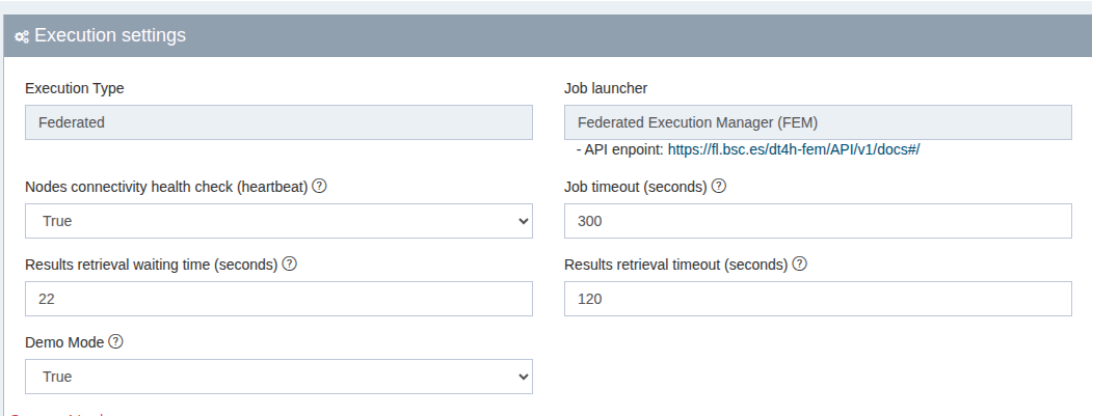

Figure 8. *Generic FEM execution parameters as displayed in the Execution Box of the FP Dashboard*

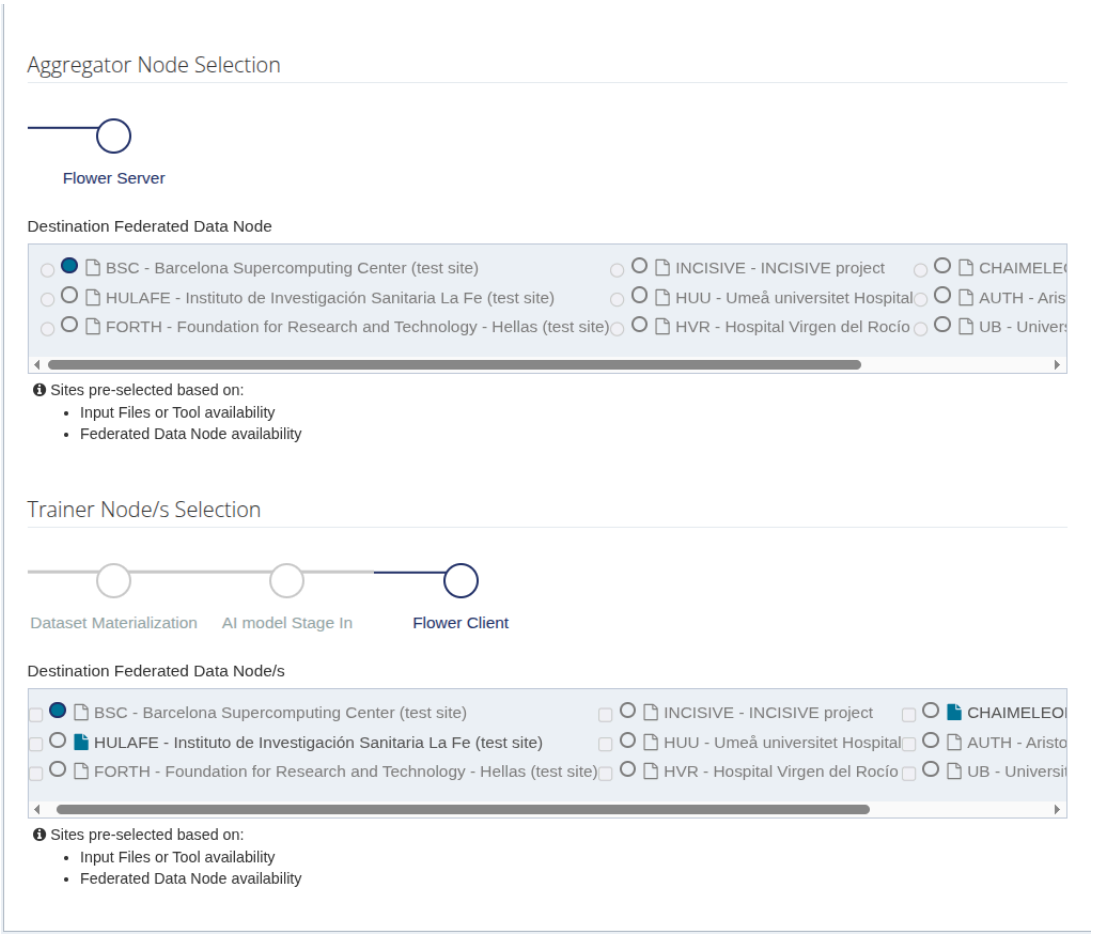

Figure 9. *FDN selectors associated to each FEM task as displayed in the Execution Box of the FP Dashboard*

Execution submission

Once the form is validated, OpenVRE triggers the FEM-RUNNER (see 7.2. User Journeys). While the user is redirected to the monitoring view (see user journey B), FEM-RUNNER and the **FEM-orchestrator** perform the workflow resolution, role assignment, and task dispatching across nodes participating in the federated experiment.

7.2.3. Journey C: Monitoring job and retrieving Results

Once a federated experiment has been submitted, the user is redirected to the Workspace, where the Dashboard provides a consolidated and real-time representation of the experiment's lifecycle. This journey covers the period from submission to final retrieval of results.

Job status

The **FP Dashboard** periodically queries the **FEM-orchestrator** for job state changes (pending, running, completed) and the user can see the progress on the Workspace under the corresponding run folder.

Logs Retrieval

When the execution of an experiment ends, output artifacts (e.g., derived datasets, metrics, etc) that are allowed to be exported, following legal agreements, are shown. Currently, only "Log files" are automatically fetched from each node participating in the federated experiment and displayed to the end

user. Error traces are also displayed in the FEM-orchestrator to support debugging. Users can display or download the files produced by the execution, and re-use them in further analysis.

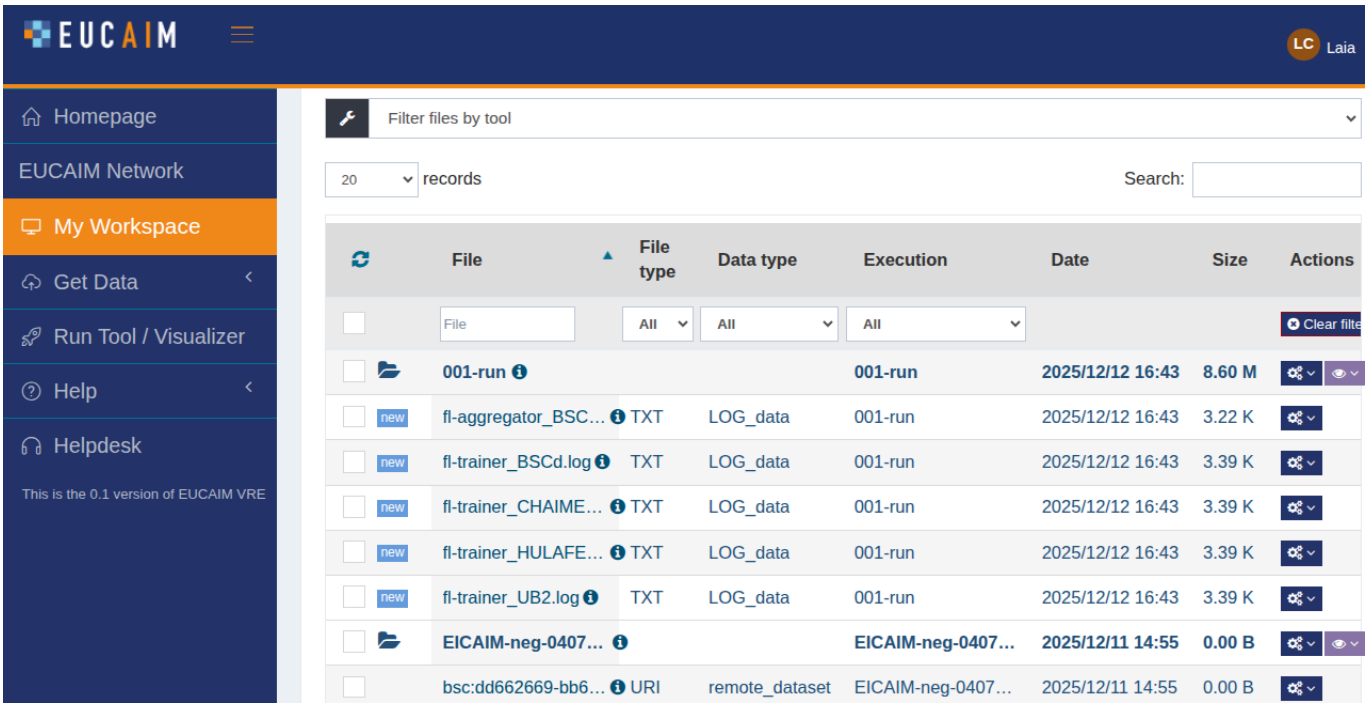

Figure 10. *Log Files generated by a federated run as displayed in the workspace of the FP Dashboard*

7.3. Deployment

The Dashboard is designed for reproducibility and adaptable deployment across development, testing, and production environments. It is distributed as a set of containerised services, following the modular architecture of openVRE[14]. A typical deployment includes:

- **openVRE backend** (PHP)
- **openVRE frontend** (web user interface)
- **MongoDB database** (storing workspace metadata, user sessions, and tool configurations)
- **HashiCorp Vault** (secure storage of secrets and tokens when OIDC is not enabled)
- **SGEcore** (internal openVRE task scheduling engine) with the **FEM-RUNNER** embedded as the execution adapter
- **Reverse proxy (NGINX)** for TLS and routing
- **Shared data volume** (storing user executions)

A reference test deployment is maintained at BSC using Docker Compose. This environment is used for integration testing with the FEM Orchestrator and validation of newly onboarded tools. The production deployment is being installed at the EUCAIM Reference Node using Kubernetes, chosen for its robustness, scalability, and operational resilience. In this setup, all openVRE components run as independent pods, with Kubernetes managing service discovery, scaling, secret injection, and persistent storage.

[14] openVRE docs: https://github.com/inab/openVRE/wiki/Overview#architecture

Table 6. Basic information of current deployments of OpenVRE in EUCAIM.

Instance	Node	Deployment Type	Purpose and Access
Integration	BSC	Docker Compose	Integration testing, tool onboarding, continuous development
			Access: https://fl.bsc.es/eucaim-fld
Production	UPV	Kubernetes	Official EUCAIM user instance; scalable, secure federated analyses
			Access: Not yet accessible

Annex 1. DH side `launcher.json` file

Empty launcher.json

```
{
    "submit_run": {
        "cmd": [
            "local command to run a docker container or kubernetes pod (no gpu)"
        ],
        "cmd_gpu": [
            "local command to run a docker container or kubernetes pod (with gpu access)"
        ]
    },
    "cancel_run": {
        "cmd": [
            "local command to cancel or stop a docker container or kubernetes pod"
        ]
    },
    "run_status": {
        "cmd": [
            "local command to cancel or query a docker container or kubernetes pod"
        ]
    },
    "print_images": {
        "cmd": [
            "local command to list all docker images or kubernetes images"
        ]
    },
    "run_report": {
        "cmd": [
            "local command to list the status of the system (docker or kubernates)"
        ]
    },
    "system_var_handler": {
        "list of local mappings",
    },
    "site_vars": {
        "pairs of name-value of local variables",
    }
}
```

Examples of filled lanugher.json

```
{
    "submit_run": {
        "cmd": [
            "CONTAINER_ID=$(docker run -d --network host --shm-size=%%MEMORY%% %%VOLUMES%%
%%PORT_MAPPINGS%% %%VAR_ENVS%%
%%DOCKER_IMAGE_REPO%%%%DOCKER_IMAGE_NAME%%:%%DOCKER_IMAGE_TAG%% %%TASK_COMMAND%%) &&",
            "echo \"{\\\"container_id\\\": \\\"$CONTAINER_ID\\\"}\" | jq '.'"
        ],
        "cmd_gpu": [
            "CONTAINER_ID=$(docker run -d --gpus all --network host --shm-size=%%MEMORY%%
%%VOLUMES%% %%PORT_MAPPINGS%% %%VAR_ENVS%%
%%DOCKER_IMAGE_REPO%%%%DOCKER_IMAGE_NAME%%:%%DOCKER_IMAGE_TAG%% %%TASK_COMMAND%%) &&",
            "echo \"{\\\"container_id\\\": \\\"$CONTAINER_ID\\\"}\" | jq '.'"
        ]
    },
    "cancel_run": {
        "cmd": [
            "(docker kill %%CONTAINER_ID%% >/dev/null 2>&1 || true) && (docker rm
%%CONTAINER_ID%% >/dev/null 2>&1 || true) &&",
            "echo \"{\\\"status\\\": \\\"cancelled\\\", \\\"container_id\\\":
\\\"%%CONTAINER_ID%%\\\"}\" | jq '.'"
        ]
    },
    "run_status": {
        "cmd": [
            "docker logs %%CONTAINER_ID%% 2>&1 | tee
%%SANDBOX_PATH%%/%%user_id%%/%%execution_id%%/%%task_id%%.log |jq -Rsc '{\"container_id\":
\"%%CONTAINER_ID%%\", \"logs\": .}'"
        ]
    },
    "print_images": {
        "cmd": [
            "docker images --digests --format '{{json .}}' | jq -s '[.[] | {repository:
.Repository, tag: .Tag, image_id: .ID, sha256_digest: .Digest}]'"
        ]
    },
    "run_report": {
        "cmd": [
            "CONTAINER_ID=%%CONTAINER_ID%% &&",
            "docker inspect \"$CONTAINER_ID\" >/dev/null 2>&1 &&",
            "EXIT_CODE=$(docker inspect \"$CONTAINER_ID\" --format='{{.State.ExitCode}}')
&&",
            "IMAGE_NAME=$(docker inspect \"$CONTAINER_ID\" --format='{{.Config.Image}}')
&&",
            "IMAGE_SHA=$(docker inspect \"$CONTAINER_ID\" --format='{{.Image}}') &&",
            "START_TIME=$(docker inspect \"$CONTAINER_ID\" --format='{{.State.StartedAt}}')
&&",
            "STOP_TIME=$(docker inspect \"$CONTAINER_ID\" --format='{{.State.FinishedAt}}')
&&",
            "STATUS=$(docker inspect \"$CONTAINER_ID\" --format='{{.State.Status}}') &&",
            "ERROR=$(docker inspect \"$CONTAINER_ID\" --format='{{.State.Error}}') &&",
            "START_TIME_UNIX=$(date -d \"$START_TIME\" +%s) &&",
            "STOP_TIME_UNIX=$(date -d \"$STOP_TIME\" +%s) &&",
            "ELAPSED_TIME=$((STOP_TIME_UNIX - START_TIME_UNIX)) &&",
            "USER_CONFIG=$(docker inspect \"$CONTAINER_ID\" --format='{{json .Config.User}}'
2>/dev/null || echo 'null') &&",
```

```
            "PRIVILEGED_MODE=$(docker inspect \"$CONTAINER_ID\" --format='{{json
.HostConfig.Privileged}}' 2>/dev/null || echo 'null') &&",
            "SERVER_NAME=$(hostname) &&",
            "CONTAINER_PID=$(docker inspect \"$CONTAINER_ID\" --format='{{.State.Pid}}')
&&",
            "MEMORY_USAGE=$(docker stats --no-stream --format '{{json .MemUsage}}'
\"$CONTAINER_ID\" 2>/dev/null || echo 'null') &&",
            "CPU_USAGE=$(docker stats --no-stream --format '{{json .CPUPerc}}'
\"$CONTAINER_ID\" 2>/dev/null || echo 'null') &&",
            "IO_USAGE=$(docker inspect \"$CONTAINER_ID\" --format='{{json
.HostConfig.BlkioDeviceReadBps}}' 2>/dev/null || echo 'null') &&",
            "MOUNTS=$(docker inspect \"$CONTAINER_ID\" --format='{{range
.Mounts}}{{.Source}}:{{.Destination}}:{{.Propagation}} | {{end}}' 2>/dev/null) &&",
            "MOUNTS_JSON=$(echo \"$MOUNTS\" | sed 's/ | $//' | jq -R 'split(\" | \")') &&",
            "CMD_EXECUTED=$(docker inspect \"$CONTAINER_ID\" --format='{{json .Config.Cmd}}'
2>/dev/null || echo 'null') &&",
            "ENV_VARS=$(docker inspect \"$CONTAINER_ID\" --format='{{json .Config.Env}}'
2>/dev/null || echo 'null') &&",
            "JSON_OUTPUT=\"{ \\\"container_id\\\": \\\"$CONTAINER_ID\\\", \\\"exit_code\\\":
$EXIT_CODE, \\\"image\\\": \\\"$IMAGE_NAME\\\", \\\"image_sha\\\": \\\"$IMAGE_SHA\\\",
\\\"start_time\\\": \\\"$START_TIME\\\", \\\"stop_time\\\": \\\"$STOP_TIME\\\",
\\\"elapsed_time_sec\\\": $ELAPSED_TIME, \\\"status\\\": \\\"$STATUS\\\", \\\"error\\\":
\\\"$ERROR\\\", \\\"user\\\": $USER_CONFIG, \\\"privileged_mode\\\": $PRIVILEGED_MODE,
\\\"host\\\": { \\\"server_name\\\": \\\"$SERVER_NAME\\\", \\\"host_user\\\":
\\\"$HOST_USER\\\", \\\"host_uid\\\": \\\"$HOST_UID\\\", \\\"host_gid\\\": \\\"$HOST_GID\\\"
}, \\\"memory\\\": $MEMORY_USAGE, \\\"cpu\\\": $CPU_USAGE, \\\"io\\\": $IO_USAGE,
\\\"mounts\\\": $MOUNTS_JSON, \\\"environment\\\": $ENV_VARS, \\\"command\\\": $CMD_EXECUTED
}\" &&",
            "echo \"$JSON_OUTPUT\" | jq '.' || echo '{ \\\"error\\\": \\\"Container not
found\\\", \\\"container_id\\\": \\\"$CONTAINER_ID\\\" }' | jq '.'"
        ]
    },
    "system_var_handler": {
        "VOLUMES": "-v {host_path}:{container_path}",
        "PORT_MAPPINGS": "-p {host_port}:{container_port}",
        "VAR_ENVS": "-e {key}={value}"
    },
    "site_vars": {
        "ONFHIR_FEAST_BASEURL": "http://onfhir-feast:8087/onfhir-feast"
    }
}
```

Annex 2. FEM Workflow definition

A2.1. FEM Task

```
# task.schema.yaml
cwlVersion: v1.2
class: CommandLineTool

id: string               # Machine-readable name of the task
label: string            # Human-readable name of the task
doc: string              # Optional description of the task

requirements:
  DockerRequirement:
    dockerPull: string    # Full image URL (with tag)
```

```
  EnvVarRequirement:
    envDef:
      VAR_NAME: string    # Any number of environment variables
  ResourceRequirement:
    coresMin: int         # Optional: Minimum CPU cores
    ramMin: int           # Optional: Minimum RAM in MB
  InitialWorkDirRequirement:   # Optional: mount volumes at runtime
    listing:
      - entryname: string
        entry: string

inputs:
  input_name:
    type: string | int | File | Directory | array  # Strongly typed
    inputBinding:
      position: int
      prefix: string
      separate: true | false
      valueFrom: string | expression (optional)

outputs:
  output_name:
    type: File | Directory | array
    outputBinding:
      glob: string         # Pattern to capture output

baseCommand:
  - string                 # Base executable command (array form)

arguments:
  - position: int
    prefix: string
    valueFrom: string | expression (optional)
    separate: true | false

stdout: string             # Optional: path to capture stdout
stderr: string             # Optional: path to capture stderr

metadata:                  # Optional: extra metadata
  author: string
  version: string
  visibility: public | private
  pre_checks: boolean
  persistent: boolean
```

A2.2. Bundle

```
# tool.bundle.schema.yaml

cwlVersion: v1.2
class: BundleOfCommandLineTool

id: string               # Machine-readable name of the bundle
label: string            # Human-readable name of the bundle
doc: string              # Optional description of the bundle

mode:
  - federated               # Federated: multi-node with roles (client/server)
  - distributed             # Distributed mode (multi-node parallel tasks)

tasks:
  - task_id: string             # Reference to Task (CommandLineTool) ID
    role: server | client | worker
```

```
    multiplicity:
      type: single | multiple
      description: string         # Free text for clarity (e.g., "1 server", "multiple clients")
    execution_site: runtime-param  # Always dynamic at execution time

inputs:
  global:
    input_name:
      type: string | int | File | Directory | array
      description: string
      required: boolean
  per_role:
    client:
      input_name:
        type: string | int | File | Directory | array
        description: string
        required: boolean
    server:
      input_name:
        type: string | int | File | Directory | array
        description: string
        required: boolean
    worker:
      input_name:
        type: string | int | File | Directory | array
        description: string
        required: boolean

outputs:
  global:
    output_name:
      type: File | Directory | array
      description: string
  per_role:
    client:
      output_name:
        type: File | Directory | array
        description: string
    server:
      output_name:
        type: File | Directory | array
        description: string
    worker:
      output_name:
        type: File | Directory | array
        description: string

metadata:                  # Optional: extra metadata
  author: string
  version: string
  visibility: public | private
  orchestrator:
    network: overlay | host | bridge
    additional_metadata: object        # Freeform metadata for node-level orchestration
```

A2.3. Workflow

```
# workflow.schema.yaml

cwlVersion: v1.2
class: FederatedWorkflow

id: string                 # Machine-readable workflow ID
```

```
label: string              # Human-readable name of the workflow
doc: string                # Optional description of the workflow purpose

parameters:                # General parameters for the workflow
  runtime_mode: federated | distributed # I don't think this is necessary anymore as
                                        # we have this at bundle level
  description: string

execution_nodes:           # Runtime input: participating nodes and roles
  node_alias:
    node_name: runtime-param   # runtime node reference
    roles:
      - server | client | worker
    bundles:                   # Which bundles this node will participate in
      - bundle_id

bundles:                    # Composition of the workflow
  - bundle_id: string          # Reference to a Tool bundle
    alias: string              # Optional short alias for readability
    inputs:                    # Bind workflow-level inputs to bundle inputs
      global:
        input_name: workflow_input_reference
      per_role:
        client:
          input_name: workflow_input_reference
        server:
          input_name: workflow_input_reference
        worker:
          input_name: workflow_input_reference
    outputs:                   # Map bundle outputs to workflow-level outputs
      global:
        output_name: workflow_output_reference
      per_role:
        client:
          output_name: workflow_output_reference
        server:
          output_name: workflow_output_reference
        worker:
          output_name: workflow_output_reference
    depends_on:                # Dependencies between bundles
      - bundle_id              # Array of bundle ids

inputs:                       # Workflow-level inputs (parameters, files, etc.)
  input_name:
    type: string | int | File | Directory | array
    description: string
    required: boolean

outputs:                      # Workflow-level outputs (final artifacts)
  output_name:
    type: File | Directory | array
    description: string
```

A2.4. Example

A simple "workflow" that runs a heartbeat + health check + FLcore.

```
# federated-workflow.yaml

cwlVersion: v1.2
class: FederatedWorkflow

id: federated-demo
```

```
label: "Federated Workflow: Heartbeat + HealthCheck + FLcore"
doc: "Run heartbeat across all nodes, health-check on selected nodes, and FLcore federated
training."

parameters:
  runtime_mode: federated
  description: "Full federated workflow combining monitoring and federated learning."
  orchestrator:
    container_runtime: docker
    network: overlay
    additional_metadata:
      priority: high

execution_nodes:
  node_bsc:
    node_name: runtime-param   # Injected at runtime
    roles: [ server, client ]
    bundles: [ flcore, heartbeat, healthcheck ]

  node_ub2:
    node_name: runtime-param
    roles: [ client ]
    bundles: [ heartbeat, healthcheck ]

  node_ub3:
    node_name: runtime-param
    roles: [ client ]
    bundles: [ flcore, heartbeat, healthcheck ]

  node_hulafe:
    node_name: runtime-param
    roles: [ client ]
    bundles: [ flcore, heartbeat, healthcheck ]

bundles:

  - bundle_id: heartbeat
    alias: "node-heartbeat"
    inputs: {}
    outputs: {}
    depends_on: []

  - bundle_id: healthcheck
    alias: "node-healthcheck"
    inputs: {}
    outputs: {}
    depends_on:
      - heartbeat

  - bundle_id: flcore
    alias: "fl-training"
    inputs:                            # Updated at runtime
      global:
        model_type: model_type_input   # Workflow-level input
        data_path: global_data_path    # Workflow-level input
      per_role:
        server:
          rounds: training_rounds_input     # Workflow-level input
```

```
        client:
          local_epochs: client_epochs_input # Workflow-level input
          dataset_id: client_dataset_input  # Workflow-level input
    outputs:
      global:
        aggregated_model: final_model_output
    depends_on:
      - healthcheck

inputs:
  model_type:
    type: string
    description: "Model architecture to train."
    required: true

  global_data_path:
    type: Directory
    description: "Path to shared dataset storage."
    required: true

  training_rounds_input:
    type: int
    description: "Number of training rounds."
    required: true

  client_epochs_input:
    type: int
    description: "Number of local epochs per client."
    required: true

  client_dataset_input:
    type: int
    description: "Dataset identifier per client."
    required: true

outputs:
  final_model_output:
    type: File
    description: "Final trained federated model."
```

Annex 3. Software revision

A3.1. Prohibited AI practices

A3.1.1. AI Act Article 5, Prohibited AI practices

The European Union's Artificial Intelligence Act (AI Act) identifies specific AI practices that are prohibited due to their potential to cause significant harm or infringe upon fundamental rights. These prohibited practices include:

1. Subliminal Manipulation: AI systems that deploy subliminal techniques beyond a person's consciousness to materially distort behavior, impairing the ability to make informed decisions and causing significant harm.

2. Exploitation of Vulnerabilities: AI systems that exploit vulnerabilities of individuals or specific groups based on age, disability, or socio-economic status, leading to behavior distortion and significant harm.

3. Social Scoring: AI systems used for evaluating or classifying individuals or groups over time based on social behavior or personal characteristics, resulting in unjustified or disproportionate detrimental treatment.

4. Predictive Policing: AI systems that assess or predict the risk of individuals committing criminal offenses based solely on profiling or assessing personality traits, without objective and verifiable facts directly linked to criminal activity.

5. Untargeted Facial Recognition Data Collection: AI systems that create or expand facial recognition databases through untargeted scraping of facial images from the internet or CCTV footage.

6. Emotion Recognition in Sensitive Areas: AI systems used to infer emotions of individuals in workplaces and educational institutions, except when intended for medical or safety reasons.

7. Biometric Categorization: AI systems that categorize individuals based on biometric data to deduce or infer sensitive attributes such as race, political opinions, religious beliefs, or sexual orientation.

8. Real-Time Remote Biometric Identification: The use of real-time remote biometric identification systems in publicly accessible spaces for law enforcement purposes, with specific exceptions for serious threats or criminal investigations.

A3.1.2. Prohibitions in the European Health Data Space (EHDS) Regulation

In the latest version of the forthcoming European Health Data Space (EHDS) Regulation, specific prohibitions are outlined to ensure the ethical use of health data:

1. Detrimental Decision-Making: Using health data to make decisions that could harm individuals or groups.

2. Discrimination: Making decisions that disadvantage people in job offers, goods/services, insurance, or loans based on their health data.

3. Advertising or Marketing: Using health data for promotional activities.

4. Harmful Products/Services: Developing products or services that harm individuals, public health, or society (e.g., illicit drugs, weapons).

5. Ethics Violations: Engaging in activities that conflict with ethical standards under national law.

A3.1. Self-risk assessment

A3.1.1. General Risks

1. An (unknown) vulnerability in the tool's code can be exploited by malicious actors, potentially without the knowledge of the tool provider

2. An (unknown) vulnerability in a third-party library can be exploited by malicious actors, potentially without the knowledge of the tool provider

3. The tool running in EUCAIM is not the tool that was actually validated (malicious substitution, or human error)

4. The combination of multiple tools or datasets allows data leaks/reidentification or other harmful consequences

5. A software preempts or uses all of the computational resources available

6. A software causes a crash of the whole system

7. A software corrupts the data

8. The software has been developed based on unethical practices

9. A user executes a software for which they did not have the right according to intellectual property or copyright laws

10. A software introduces algorithmic bias in the pipeline

11. After a harmful event, it is impossible to trace back the history of the software's execution

A3.1.2. Risks during AI model training

1. Training gradients can be "inverted" to identify training data (either by the data scientist, or by other data providers, or by a third-party malicious actor)

2. Code for model training (dynamically provided by data scientist) includes malicious procedures

3. Malfunctions during model training lead to accidental leak of information (e.g. error logs reveal sensitive information about a data sample)

4. Malicious actors with a legitimate role in the training process can "poison" model with ad-hoc data or crafted gradient updates to prevent correct model training

5. Malicious actors can "impersonate" a legitimate data provider to perform attacks such as model poisoning

6. Malicious actors exploit legitimate training protocols that rely on multiple training of similar models (e.g. leave-one-out cross-validation) to perform statistical attacks

7. Training data has been corrupted, leading to a harmful malfunction of the tool

8. Model drift in a continual learning setting leads to harmful consequences

A3.1.3. Risks related to using an AI model trained on EUCAIM data

1. Generative models leak training data samples

2. Trained model can be "inverted" to identify training data

3. A model trained on a data subset (e.g. dataset at time *t*) turns out to leak sensitive information when used for inference on another dataset (e.g. dataset at time *t+delta*, or out-of-distribution data)

4. The trained model discriminates specific data subpopulations

5. The model is trained on a dataset that is too small, thus leading to overfitting that can be exploited for data leaks

6. The trained model is also (maliciously) provided a backdoor that causes it to leak data only under specific circumstances (e.g. when presented with a special kind of input)

7. The legitimately trained model is used for malicious purposes outside of EUCAIM

8. Infringement of intellectual property: a model trained on EUCAIM data is "stolen"

9. The trained model is used to make a harmful clinical decision

Annex 4. Registered software

Name	Version	Topics	Operations
2D Digital Mammography Harmonization (EUCAIM-SW-046_T-01-03-008)	['1.1.0']	Data curation and archival; Computer science; Imaging; Data management	Data filtering; Data filtering
AITANA	['v6.0']	Machine learning	
AutoProSeq - automatic classification of sequence types for prostate mpMRI	['v0.1']	Medical informatics	Image annotation
Biologically motivated normalization techniques	['1.6']	Machine learning; Data visualisation	Image analysis
Brain Glioma Segmenter	['v1.0']	MRI; Machine learning	Image annotation
Brain Metastasis Segmenter	['v1.0']	MRI; Machine learning	Image annotation
Breast Dense Tissue Segmentation	['v1.0.6']	Imaging; Medical imaging	Image annotation
BreastDeformNet	['alfa01']	Surgery; MRI; Machine learning	
Cluster based harmonization (EUCAIM-SW-044_T-01-03-006)	['1.1.0']	Data curation and archival; Imaging	Clustering; Clustering
CT Slice thickness harmonization	['0.1']	Machine learning; Medical imaging	Data filtering
CT-based neuroblastoma tumour detection and segmentation (EUCAIM-SW-022_T-01-02-005)	['1.0.0']	Oncology; Machine learning	
Data Ingestion Tool (UPV Reference Node)	['1.0.5']	Data integration and warehousing; Software engineering; Imaging	
Data Integration Quality Check Tool (DIQCT)	[]	Data quality management	
datmat	['0.2.0']		Data retrieval; Data sorting
Deep features extraction (EUCAIM-SW-059_T-02-01-007)	['1.0.0']	Data architecture, analysis and design	
Deep Learning Noise Reduction (DLNR)	[]	Data quality management	
Denoising-Inhomogeneity Correction Tool	['1.1.0']	Data curation and archival	Data filtering; Data filtering

(EUCAIM-SW-015_T-01-01-015)			
DICOM defacing anonymisation	[]	Medical imaging; Data protection	
DICOM File Integrity Checker (EUCAIM-SW-002_T-01-01-002)	['2.0.0', '2.1.0', '2.1.1']	Computer science; Data acquisition; Data management	Format validation; Format validation
DICOM tags extractor	['1.1']	Data architecture, analysis and design; Medical imaging	Image analysis; Image analysis
DICOM-SEG Annotation	['0.1']	Data quality management	
Diffusion maps extraction (EUCAIM-SW-057_T-02-01-005)	['1.0.0']	Data architecture, analysis and design	
EAPP - Extended a Priori Probability tool	['v1.0.1']	Machine learning; Statistics and probability	Data analysis
EUCAIM CDM Federated Unbalance Checker	['alfa01']	Data quality management	Data analysis; Data analysis
EUCAIM DICOM Anonymizer	['0.12.0']	Medical imaging; Data protection	Anonymisation
EUCAIM DICOMseg Converter	['1.3']		
EUCAIM ETL toolset	['0.1.1', '0.1.2']	Data identity and mapping; Workflows	Data formatting; Data deposition; Format validation
EUCAIM Wizard Tool	['beta1.0']	Medical imaging; Data protection; Oncology	Anonymisation
EUCAIM_DicomSeg_Converter	['1.3']		
EUCAIM_DicomSeg_Converter	[]		
EUCAIM_DicomSeg_Converter	[]		
FAIR EVA for EUCAIM	['EUCAIM_plugin_0.3', 'EUCAIM_plugin_0.3.1', 'EUCAIM_plugin_0.4.0', 'EUCAIM_0.4.2']	FAIR data	Format validation; Validation
FAIR4Health Data Curation Tool	['v1.2.5']	Ontology and terminology; Data submission, annotation and curation	
Feature-based harmonization	['1.4']		
Fed-BioMed	['4.4', '6.1']	Machine learning	Image analysis; Statistical calculation
HCC segmentation tool for contrast-enhanced abdominal CT	[]	Medical informatics	Image annotation; Image annotation
Health-RI XNAT	['1']	Medical imaging	Data deposition; Image analysis
Image Duplicates Checker	['2.0']	Data quality management	
Image Quality Assessment metrics (IQA) for XNAT platform	['1.0']	Data quality management	Image analysis
img2catalog	['1.0.1']	FAIR data	Query and retrieval; Data parsing
Intracranial Meningioma Segmenter	['v1.0']	MRI; Machine learning	Image annotation
IQA DCE tool	['1.0']	MRI; Data quality management;	

		Oncology	
Lethe DICOM Anonymizer	['0.9.12']	Medical imaging; Data protection; FAIR data	Anonymisation
Mainzelliste	['v1.13.7']	Data protection; Data identity and mapping	
MITK	['v2023.04']		Image annotation
ML model for MR series categorisation (EUCAIM-SW-011_T-01-01-011)	['1.1.0']	Computer science; Data acquisition; Data management	Classification
MR-based DIPG tumour detection and segmentation (EUCAIM-SW-020_T-01-02-003)	['1.0.0']	Oncology; Machine learning	
MR-based glioblastoma tumour detection and segmentation (EUCAIM-SW-021_T-01-02-004)	['1.0.0']	Data architecture, analysis and design	
MR-based neuroblastoma tumour detection and segmentation (EUCAIM-SW-019_T-01-02-002)	['2.0.0']	Oncology; Machine learning; Data architecture, analysis and design	Image annotation
MRI Image intensity harmonization	['0.1']	MRI; Machine learning; Medical imaging	Data filtering
N4 Bias Filter	['1.7']	Machine learning; Data visualisation; Data quality management	Data filtering; Image analysis
NestedComBat	['v1']	Machine learning; Statistics and probability	
NLmCED Filter	['1.0']	Imaging; MRI; Medical imaging	Image analysis; Data filtering
nnUNet	['1.0']		Image annotation; Image annotation; Image annotation; Image annotation; Image annotation
OpenEBench	['v3']	Biomedical science; Bioinformatics; Ontology and terminology; Biological databases	Data retrieval
openVRE	['1.0']	Bioinformatics	Service management
Overall survival in glioblastoma multiforme (EUCAIM-SW-078_T-02-04-007)	['1.0.0']	Machine learning; Data architecture, analysis and design	
Overall survival in neuroblastoma by GIBI230 (EUCAIM-SW-079_T-02-04-008)	['1.0.0']	Machine learning; Data architecture, analysis and design	
Pediatric Brain Tumor Segmenter	['v1.0']	MRI; Machine learning	Image annotation
Perfusion maps extraction (EUCAIM-SW-054_T-02-01-002)	['1.0.0']	Data architecture, analysis and design	
Prostate zone segmentation tool	[]	Medical informatics	Image annotation; Image annotation
QP-Insights Uploader	['1.2.1']	Medical imaging; Data integration and warehousing	Data deposition
R1 and T1 maps extraction	['1.0.0']	Data architecture, analysis and design	

(EUCAIM-SW-058_T-02-01-006)			
R2 maps extraction (EUCAIM-SW-056_T-02-01-004)	['1.0.0']	Data architecture, analysis and design	
RACLAHE	['3.0']	Data visualisation; Machine learning	
Radiatus	['v5.0']	Data architecture, analysis and design; Machine learning	
Radiomic features extraction (EUCAIM-SW-055_T-02-01-003)	['1.0.0']		
Radiomics based Lung cancer staging	[]	Machine learning; Medical informatics; Oncology	
RadiomicsEnabler	['01']	Medical imaging; Preclinical and clinical studies	
Sub-Saharan Africa Brain Glioma Segmenter	['v1.0']	MRI; Machine learning	Image annotation
Tabular Data Curator	[]	Data quality management	
Time Coherence Tool (EUCAIM-SW-001_T-01-01-001)	['1.1.0']	Data visualisation; Computer science	Data analysis
to delete	[]		
Trace4Harmonization	['v1.0.00']	Machine learning; Statistics and probability	
Trace4MedicalImageCleaning	['1.0']		
UM - Automated detection and segmentation NSCLC	[]	Data architecture, analysis and design	
UM - Precision Medicine Toolbox	['0.2']	Oncology; Data architecture, analysis and design	
UM - RQS 2.0 with RRLs	['2.0']		
Virtual Imaging Platform	['3.0.1']		Image analysis
Volumetry analysis (EUCAIM-SW-061_T-02-01-009)	['1.0.0']	Data architecture, analysis and design	
XNAT	[]	Medical imaging; MRI; Bioimaging; Tomography; Data management	Data deposition; Data sorting; Query and retrieval; Anonymisation
XNATpy	[]	Data management; FAIR data; Medical imaging	Data retrieval; Database search; Data deposition