UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dept. of Computer Systems and Computation

Analysis and improvement of a Kubernetes deployment of the EUCAIM platform services

Master's Thesis

Master's Degree in Cloud and High-Performance Computing

AUTHOR: Montoliu Rico, Pablo

Tutor: Blanquer Espert, Ignacio

Experimental director: Lozano Lloret, Pau

ACADEMIC YEAR: 2023/2024

ACKNOWLEDGMENTS:

I would like to dedicate this work to all cancer patients, to whom I sincerely hope both my work and the efforts of everyone involved in the EUCAIM project will be of use in the near future.

Additionally, I would like to thank, first and foremost, Professor Ignacio Blanquer Espert and my colleague Pau Lozano Lloret for their kindness, their patience in teaching me, and for everything I have learned while working with them. I would also like to thank my family and friends, without whom none of this would have been possible.

Lastly, I would like to give special mention to those who have dedicated their lives to study and knowledge, to those who live their profession with absolute passion, and whose tireless efforts and curiosity continually push the boundaries of knowledge beyond the known.

Abstract

The present Master's Thesis will analyse the services deployed within the framework of the EUCAIM platform, proposing improvements in accordance with best practice guidelines for the creation of Kubernetes manifests and Dockerfiles, as well as the inclusion of scripts for configuration automation. It will also explore the possibility of using a CI/CD environment for the verification and deployment of solutions, integrated with the GitHub platform.

Keywords: Kubernetes, Orchestration, Containers, Helm, Deployment, Manifest, EUCAIM, Platform, CI/CD, Federated repositories, Data repositories.

Table of Contents

1. Introduction

Over the past few years, the use of artificial intelligence tools has expanded into numerous fields, causing a significant impact on how problems are addressed. In the clinical domain, these tools have enabled the improvement of diagnostic procedures, treatment, and response time in identifying the need for preventive medicine.

The enhancement and development of artificial intelligence tools require the use of a large volume of data and infrastructures that are often unavailable for most users. Moreover, the management of medical data imposes important restrictions that difficult access and transfer of data.

Regarding the use of these AI tools in the fight against cancer, Europe faces a significant challenge; our clinical data (especially medical imaging) is fragmented and underutilised. Accessing clinical data volumes involves overcoming numerous obstacles, such as obtaining the necessary authorizations or credentials to access data stored by clinical centres, ensuring proper anonymization of images or data, obtaining data in the appropriate format, ensuring that data is used appropriately and always within the legal framework, etc.

Regarding the necessary infrastructure for building AI tools against cancer, training AI models within a reasonable time frame usually requires a significant computing infrastructure with considerable computing capacity. This reality discourages most users from the possibility of training their own models due to the enormous costs involved in accessing and managing such infrastructures.

In this context, the EUCAIM project aims to create a federated infrastructure that brings together all cancer images under a secure and compliant framework. This cancer image atlas will drive the development of advanced AI tools. The central hub of the EUCAIM project will govern and streamline the flow of information, and the project commitment to the FAIR data principles will ensure that the data will be findable, interoperable, accessible, and reusable. This infrastructure goal is to provide the necessary means for the validation and development of artificial intelligence tools focused on precision medicine that will support and enhance the diagnostic procedures, treatment, and identification of the need for predictive medicine.

The federated infrastructure is governed by the federation core services that are run on the Central Hub. These services are key for the findability, accessibility and reuse of the data shared in the infrastructure. Therefore, it is key to have a reliable and performant platform to manage the services. The objective of this work will be to migrate and deploy these core services to an environment that can provide the resilience and performance required, as well as to facilitate operation and maintenance.

1.1 Motivation

The EUCAIM platform, consisting of a wide range of services and components within a complex environment, requires a structured architecture for integration and ongoing maintenance to ensure seamless operation. Software containers and container management platforms are key to simplify the creation and management of these services. In this context, tools like Kubernetes and Helm are invaluable. These tools benefit both developers and platform managers by enabling efficient and scalable deployment and maintenance.

Now that we understand the utility of Kubernetes and Helm, it's important to highlight two key aspects: good practices and automation. The importance of good practices in creating Kubernetes manifests is crucial to ensuring deployments are efficient, scalable, and reliable. Automation plays a vital role in maintaining consistency across environments like EUCAIM, accelerating deployment processes, facilitating scalability, and enhancing overall system reliability. Automatic validation is another critical element, ensuring that all Kubernetes configurations meet the required standards before deployment. This validation process helps to avoid configuration errors and potential disruptions in service, making it an essential step in the CI/CD pipeline.

Furthermore, Kubernetes offers the potential for managing multiple deployments, enabling the EUCAIM platform to effectively manage and scale its diverse services. These multiple deployments can cater to various aspects of the platform, ranging from testing and development environments to production deployments, each tailored to specific requirements. This flexibility ensures the platform can continue to evolve, allowing for the addition of new services or the updating of existing ones without disrupting overall operations.

1.2 Objectives

In this work, we aim to improve various aspects of the previous deployments of the platform. The improvements we have proposed for our new deployment of the EUCAIM Core Services are:

1. Increasing automation and facilitating the deployment of the platform.
2. Achieving reproducibility of the environment to obtain the same results regardless of the machine on which the platform is deployed.
3. Enhancing the security of the deployment by applying best practices in the development of the Infrastructure as code recipes.
4. Overall optimization of deployment processes, ensuring efficient, reliable, and secure updates to the platform.

1.3 Methodology

To achieve a solution that best meets the objectives outlined in the previous section, a classic work methodology was followed. This methodology consisted of the following points or sections:

1. **Analysis**

   First, a detailed analysis of the previous infrastructure and all its components was carried out.

2. **Architectural design**

   Once the previous deployment was analysed, its functionality and architecture were documented to serve as a basis for future deployments.

3. **Study of best practices**

   A study of best practices for managing the Kubernetes environment and creating new manifests for deployments was conducted.

4. **Proposal of improvements**

   Defects or areas for improvement in previous deployments were identified, and various changes were proposed to enhance the security of the configurations and automate the environment.

5. **Deployment**

   In this phase, the updated configurations were created and deployed in the new environment.

6. **Validation**

   Load tests, usage tests, and analysis of the results were performed.

In the next Figure 0 you can see the Gantt chart derived from this planning.

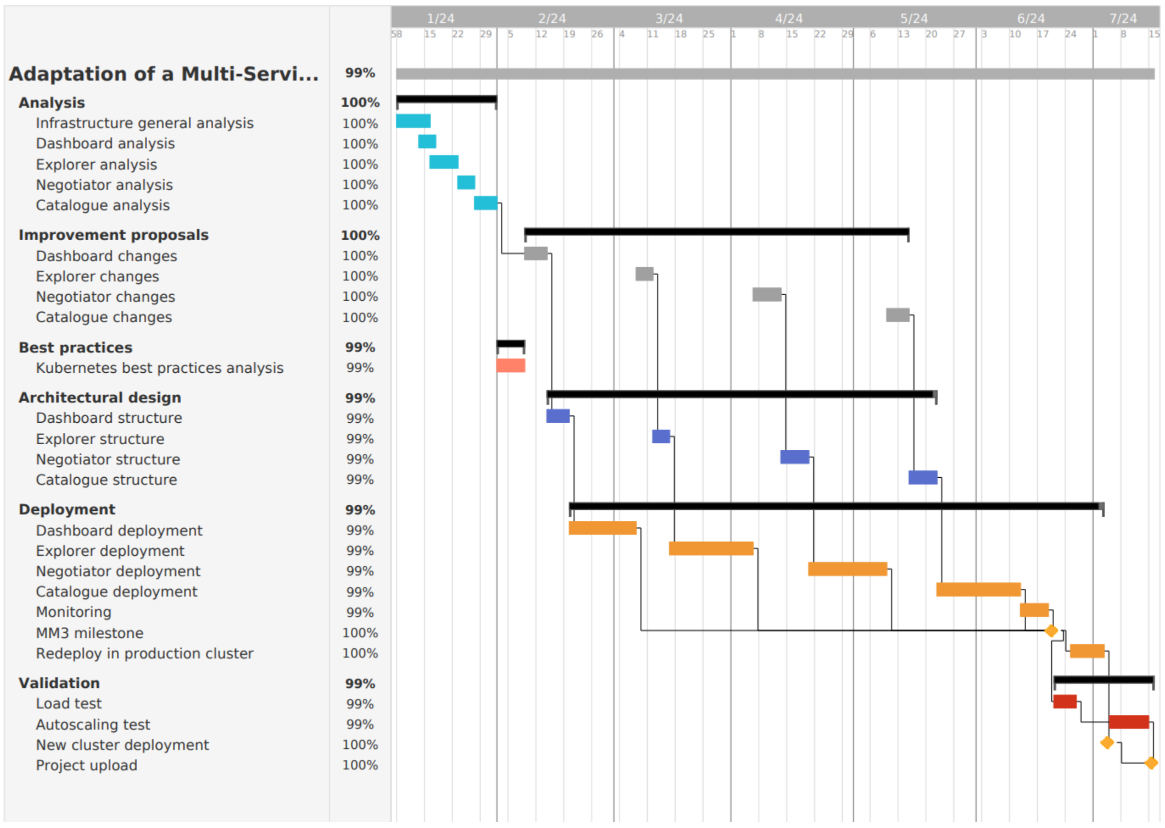


Figure 0. *Projects Gantt Chart*

1.4 Document Structure

The following document is divided into the following sections:

State of the Art

In this section, the reader will find a brief introduction to the core technologies and methodologies used in this project, as well as the institutional context from which this project originated.

Problem analysis

This section provides a detailed analysis of the problem at hand and establishes the requirements that the application must meet to address the identified needs. Additionally, it includes the specification of requirements along with a detailed explanation of the proposed solution.

Solution Design

Throughout this section, the reader will find aspects such as the architecture of the solution, the components it consists of, and a more detailed explanation of the application's design.

Solution Development

In the section dedicated to the development of the proposed solution, there is a summary of the configuration necessary for the system to function correctly. In addition to this configuration, the reader can find code snippets that have been deemed most relevant, accompanied by various explanations regarding their importance and structure.

Testing

This section includes the load tests carried out to verify the correct behaviour of the application under high load situations, as well as a brief explanation of how to perform these load tests.

Conclusions and Future Plan

This section presents the conclusions drawn from this work, the relationship between the work and the studies undertaken, and a thorough analysis of aspects that could be improved in future revisions of the work, which will be addressed in the workspace.

1.5 Conventions

- The source code is written in Courier New font.
- All recipes mentioned are available at: https://github.com/EUCAIM/k8s-deployments.
- Direct quotes from external works will appear in quotation marks.
- References to documents will be cited.
- References to official web pages will include only the access date and will be noted as footnotes.

2 State of the Art

2.1 Institutional Context (EUCAIM ECII Project Framework)

In 2021, the European Commission presented Europe's Beating Cancer Plan[1], a project aimed at leveraging the latest technologies, research, and innovation for patient benefit. [1]

This project can be divided into a series of sub-projects or initiatives, among which is the European Cancer Imaging Initiative [2](ECII). ECII's objective is to make the most of the potential of data and digital technologies such as Artificial Intelligence (AI) or High-Performance Computing (HPC) to combat cancer. The cornerstone of the European Cancer Imaging Initiative will be a federated European infrastructure for cancer image data, developed by the European Federation for Cancer Images (EUCAIM[3]).

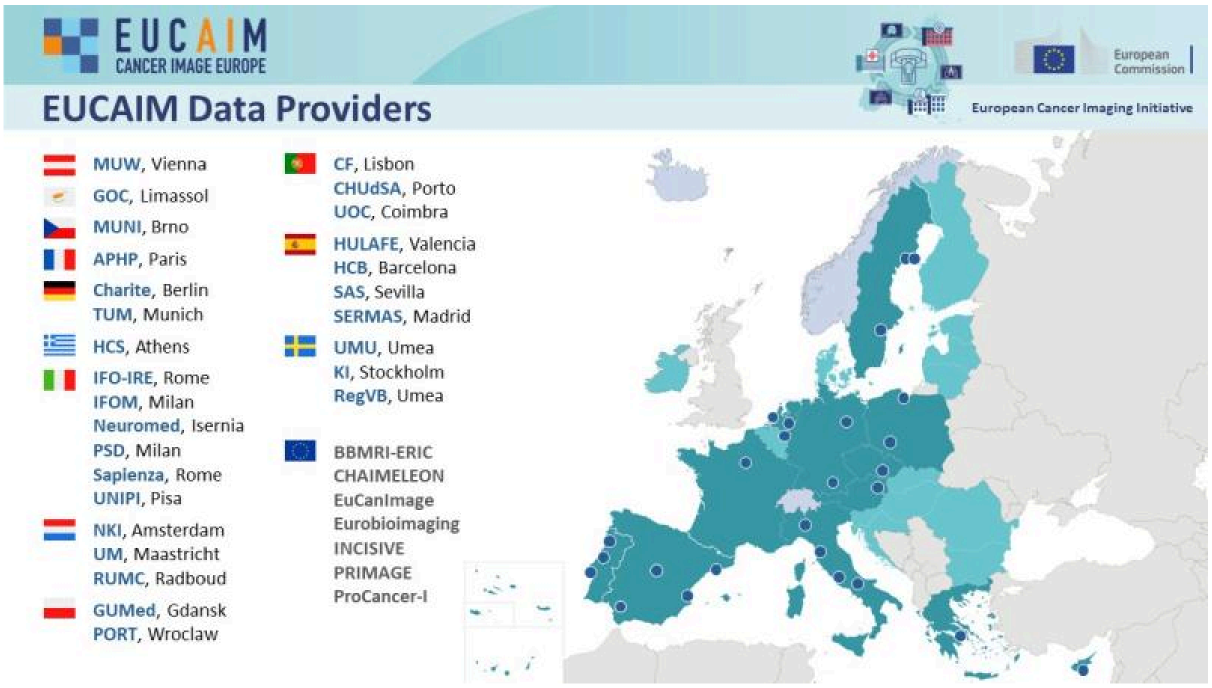


Figure 1. *EUCAIM Data Providers*

1 https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/promoting-our-european-way-life/european-health-union/cancer-plan-europe_en

2 https://digital-strategy.ec.europa.eu/en/policies/cancer-imaging

3 https://cancerimage.eu/

This project, composed of 76 partners, started with 21 clinical sites from 12 countries and aims to have at least 30 distributed data providers from 15 countries by the end of the project. The project will provide a central hub that will link EU-level and national initiatives, hospital networks, as well as research repositories with cancer images data. Clinicians, researchers, and innovators will have cross-border access to an interoperable, privacy-preserving, and secure infrastructure for federated, distributed analysis of cancer imaging data.

2.2 EUCAIM

The project EUropean Federation for CAncer IMages (EUCAIM), started in 2023, is the cornerstone of the European Commission-initiated European Cancer Imaging Initiative, and a crucial element of the Europe’s Beating Cancer Plan (EBCP), which as we previously said aims to foster innovation of digital technologies in cancer treatment and care to achieve more precise and faster clinical decision making, diagnostics, treatment and predictive medicine for cancer patients.

Health data, including medical images, are highly distributed and fragmented in Europe. Artificial Intelligence (AI) offers a paradigm shift towards data-driven decision making that is revolutionising medicine. In medical imaging, an increasing number of studies are appearing in which AI tools are making important contributions towards a more accurate diagnosis or a more reliable treatment response and prognostic prediction. However, there can be a significant difference between the design and the real-world performance of algorithms, leading to ethical and safety issues. The aim of EUCAIM is to address this issue: to **deploy a federated, pan-European research infrastructure** in order to address the fragmentation of the health data and medical images in Europe.

This research infrastructure will support the development and validation of AI tools toward Precision Medicine. Such AI tools will support and enhance the (cancer) diagnosis procedure, treatment and the identification of the need for predictive medicine. Overall, this infrastructure will provide a platform for developing AI tools that aim to enhance cancer treatment and the diagnosis procedure for patients.

EUCAIM is building on the already existing, but fragmented data repositories of cancer images developed through the projects of the AI4HI initiative. Furthermore, it is also defining the legal grounds for the operation of the federated data repository on the European scale by adapting to the particularities of the data management regulations of the different European countries. Preserving the data sovereignty of the data providers is a high priority in EUCAIM.

2.3 Medical Data Situation in Europe

If we talk about data regulations in Europe, we are inevitably talking about the Data Act[4]. In February 2022, the European Union introduced a proposal for the Data Act, a cornerstone of Europe's Digital Decade vision for 2030. The Data Act is expected to significantly impact data holders and third parties handling data—both personal and non-personal—from users of connected products and related services, particularly those generated by Internet of Things (IoT) devices. [2]

The Data Act aims to specify who can access, transfer, and use EU users data, with the goal of boosting competitiveness and innovation within the Union, as well as ensuring sustainable economic growth. Key features of this new regulation include:

- **Legal Security:** The Data Act provides legal certainty by establishing the rights and obligations of users and data holders concerning access, use, and sharing of data from products and services, including necessary metadata.
- **Service Switching and Interoperability:** It facilitates switching data processing services, particularly among cloud service providers, and enhances data interoperability and data exchange mechanisms.
- **Intellectual Property and Confidentiality Protection:** The act ensures the protection of intellectual property and trade secrets, as well as safeguarding data that could expose companies to vulnerabilities, such as cybersecurity incidents.
- **Regulated Data Sharing:** It governs the conditions under which data holders may share data with third-party recipients, including measures to prevent unauthorised use and disclosure of data, and outlines possible compensations agreements.
- **Abusive Clauses:** The regulation identifies clauses that are always considered abusive and those presumed abusive in business relationships concerning data access and usage.
- **Data for Public Bodies:** It allows the provision of data to public bodies and certain authorities under specific exceptional circumstances.

Additionally, the Data Act complements the Data Governance Act, which promotes the creation and development of European common data spaces in strategic sectors such as health, energy, and finance, involving both public and private entities.

The Data Governance Act sets up rules to help share data between different sectors and countries, while making sure everything follows EU laws. It aims to make more and better data available, encouraging new ideas and decisions based on data. By providing clear guidelines for handling data, the act supports transparency, security, and responsible use of data across the European Union.

[4] https://digital-strategy.ec.europa.eu/es/policies/data-act

2.4 Examples of Current Federated Repositories

The EU views data in its strategy not only as a source of wealth but also as a key geostrategic element, particularly when discussing the concept of "digital sovereignty." Being relevant in the world involves controlling and exploiting your own data (at least your own). This strategic interest is also reflected in legislation on data protection, interoperability, and the directive on open data and the reuse of public sector information.

In recent years, various European projects, such as EUCAIM, have been created with the objective of providing practical solutions to the digital sovereignty problem and to achieve better data utilisation.

Some examples of these projects are:

GAIA-X

Gaia-X[5] is a European initiative designed to create a secure, federated data infrastructure, promoting digital sovereignty and interoperability. Here's a brief overview:

- **Objective:** Foster European digital sovereignty and reduce dependency on non-European cloud providers.
- **Federated Services:** Connects various cloud service providers and users through a common framework, ensuring interoperability.
- **Data Sovereignty:** Empowers data owners with control over their data, ensuring privacy and compliance with European regulations like GDPR (General Data Protection Regulation). [3]
- **Interoperability and Standards:** Establishes common standards for data exchange, security, and compliance.

[5] https://gaia-x.eu/

- **Open Source and Transparency:** Promotes open-source technologies and operational transparency to build trust.

GAIA-X is one of the providers for the middleware that will be used to implement the Common European Data Spaces, an initiative to foster data sharing among industries to boost innovation. Data Spaces are being build to facilitate the transactions and accessibility to data generated mainly in the European industry, or data that is key for the development of innovative products. [4]

EOSC (European Open Science Cloud)

The EOSC (European Open Science Cloud)[6] is an initiative by the European Commission aimed at creating a digital environment where European researchers can store, manage, analyse, and reuse scientific FAIR (Findable, Accessible, Interoperable, Reusable) data. [5] Here are the key aspects of EOSC:

- **Objective:** Facilitate access to and reuse of scientific data in Europe, promoting open science and innovation.

- **Federated Services:** Provide a federated and secure infrastructure for storing, analysing, and reusing scientific data, integrating various data infrastructures and services across Europe.

- **Interoperability:** Ensure that research data and services are interoperable, following common standards to facilitate information exchange and collaboration across disciplines and countries.

- **Open Science:** Promote open access to scientific data and publications, aligned with the FAIR principles (Findable, Accessible, Interoperable, Reusable), so Open Science become the “new normal”.

- **Collaboration and Community:** Foster collaboration among researchers, institutions, and sectors, creating an active community that contributes to and uses the EOSC, ensuring the recognition and rewarding of open science practices.

- **Governance and Sustainability:** Establish a clear and sustainable governance framework for the EOSC, ensuring its long-term maintenance and evolution for the benefit of the European scientific community.

[6] https://www.csic.es/es/internacional/eosc

- **Innovation and Competitiveness:** Drive scientific and technological innovation in Europe, enhancing the continent's research competitiveness and capacity.

In essence, the EOSC aims to create a federated, collaborative, and open digital ecosystem for European research, facilitating data sharing and promoting scientific advancements.

Dataspace INE + AEAT + GISS + BDE + SEPE

The officials of the National Institute of Statistics (INE)[7], the State Tax Administration Agency (AEAT)[8], the Social Security (General Treasury, National Institute of Social Security, Social Institute of the Navy, and IT Management), the Bank of Spain, and the State Public Employment Service (SEPE[9]) have signed an agreement to allow combined and integrated access to the records and databases they manage. This access will be granted to researchers conducting scientific work in the public interest. [6]

Access to the data will be carried out using procedures that ensure the confidentiality of the information, the protection of personal data, compliance with current legislation, and statistical secrecy at all times, guaranteeing that it is impossible to directly identify any individuals or units whose information is contained in the databases.

In this way, the Spanish administration takes a significant qualitative leap, aligning itself with the most advanced countries in the management of public data, by making one of its main assets, data, available to researchers. This is intended to be used in analytical projects that can inform decision-making processes as well as the design and implementation of public policies.

The reuse of data provided by citizens, companies, and entities to public institutions for scientific research purposes enhances its added value, significantly contributing to the economic and social progress of countries. Access to cross-referenced information from different institutional databases also maximises the value that this information can bring to research.

European Health Research Infrastructures (RIs)

[7] https://www.ine.es/en/

[8] https://sede.agenciatributaria.gob.es/Sede/en_gb/inicio.html

[9] https://www.sepe.es/HomeSepe/es/

Research Infrastructures (RIs)[10] are European structures aimed at facilitating networked research within the European Research Area. They concentrate knowledge and innovation and are positioned at the centre of the research-innovation-education triangle. Recognized as a key aspect for European competitiveness, they enable researchers to enhance their work by providing access to updated and sophisticated services that would otherwise be difficult to obtain. Additionally, they promote the creation of international collaboration networks.

The Instituto de Salud Carlos III (ISCIII)[11] acts as the central coordinator and manager of national participation, representing Spain in six European biomedical research infrastructures.

These infrastructures provide services and resources to the research community to enhance their international visibility, support the European Research Area[12], and foster innovation.

Some examples of these infrastructures are:

1. **ELIXIR[13]**: A distributed infrastructure for life-science information, supporting research data management, analysis, and interoperability.
2. **BBMRI[14]**: The Biobanking and Biomolecular Resources Research Infrastructure, which provides access to quality-controlled human biological samples and associated data.
3. **EURO-BIOIMAGING[15]**: A research infrastructure offering open access to imaging technologies in biological and biomedical sciences.
4. **EATRIS[16]**: The European Infrastructure for Translational Medicine, facilitating access to high-quality services and expertise to advance medical discoveries into patient benefits.
5. **ECRIN[17]**: The European Clinical Research Infrastructure Network leverages federated data to enhance multinational clinical research in Europe, supporting data integration, privacy, interoperability, collaborative research, resource sharing, quality assurance, and scalability.

[10] https://research-and-innovation.ec.europa.eu/strategy/strategy-2020-2024/our-digital-future/european-research-infrastructures_en

[11] https://www.isciii.es/Paginas/Inicio.aspx

[12] https://research-and-innovation.ec.europa.eu/strategy/strategy-2020-2024/our-digital-future/european-research-area_en

[13] https://elixir-europe.org/

[14] https://www.bbmri-eric.eu/

[15] https://www.eurobioimaging.eu/

[16] https://eatris.eu/

[17] https://ecrin.org/

These Research Infrastructures (RIs) support the research community by providing cutting-edge resources and fostering collaboration across Europe. They enable researchers to effectively leverage these infrastructures to advance their work and contribute to the global scientific community. Spain's commitment to these infrastructures is evident, with active participation throughout the entire process of identifying and developing biomedical products.

2.5 Tools

2.5.1 Kubernetes

As of today, Kubernetes[18] has a large developer community that supports the tool and makes up a large and rapidly growing ecosystem.

Kubernetes (k8s) is an open-source system for automating deployment, scaling, and management of containerized applications. It provides a container-centric management environment, orchestrating computing, network, and storage infrastructure so that user workloads don't have to. This offers the simplicity of Platform as a Service (PaaS) with the flexibility of Infrastructure as a Service (IaaS) and enables portability across infrastructure providers.

2.5.1.1 Alternatives to Kubernetes

While Kubernetes has become the de-facto standard for the orchestration of container orchestration, there are other popular tools in this space. Next you can find a brief list of other Container Orchestration Tools:

Docker Swarm

Docker Swarm[19] is native to Docker and turns a pool of Docker engines into a single virtual Docker engine. It is easy to set up and manage, and it features native clustering, a decentralised design, and load balancing. However, it lacks the extensive feature set and community support of Kubernetes.

Apache Mesos

Apache Mesos[20] is a high-performance cluster manager that offers efficient resource isolation and sharing for distributed applications. It supports both containerized and

[18] https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/

[19] https://docs.docker.com/engine/swarm/

[20] https://mesos.apache.org/

traditional applications, making it versatile. Mesos can scale to tens of thousands of nodes but is more complex to set up and manage compared to Kubernetes.

Red Hat OpenShift

Red Hat OpenShift[21] is built on Kubernetes and extends its features with additional developer and operational tools. It provides an integrated development environment, automated installation and upgrades, a container image registry, and advanced networking features. OpenShift is particularly strong in hybrid cloud environments but can be more complex to manage.

Nomad

Nomad by HashiCorp[22] is a flexible and modern orchestrator that supports both containerized and non-containerized applications. It is known for its simple and unified workflow, scalability to thousands of nodes, and support for diverse workloads. While it is versatile, it does not have as extensive a community or as many features specifically for containers as Kubernetes.

2.5.1.2 Object Modeling

Kubernetes offers us persistent entities called objects. These entities are used to represent the state of your cluster. Specifically, they can describe:

- What containerized applications are running (and on which nodes)
- The resources available to those applications
- The policies around how those applications behave, such as restart policies, upgrades, and fault-tolerance

These objects will be the foundation of our development and can be classified into eleven different types:

1. Pod

Firstly, a Pod in Kubernetes is the smallest deployable unit that can be created, managed, and scaled. It serves as a container for one or multiple closely related containers that are deployed on a single worker node. Pods facilitate faster communication and easier data sharing between containers. Despite their importance in managing containerized applications, Pods are ephemeral, meaning they can be frequently created, destroyed, and recreated as needed within the Kubernetes cluster. [7]

2. Deployment

[21] https://www.redhat.com/en/technologies/cloud-computing/openshift

[22] https://www.nomadproject.io/

In Kubernetes, a Deployment object is used to manage the lifecycle of one or more identical Pods. It allows users to define the desired state of an application—such as the number of replicas, the container images to use, and necessary resources—in a declarative manner, without needing to specify the steps to achieve that state. Kubernetes autonomously handles the process to match the actual state with the desired state as defined by the Deployment. Deployments are particularly suited for managing stateless applications, where no persistent data or state needs to be maintained across container restarts or replacements, making it easy to replace containers without data preservation concerns. Examples of such applications include web servers and load balancers. [8]

3. Persistent Volume

A Persistent Volume (PV) in Kubernetes is a piece of storage that remains independent of the lifecycle of any particular Pod, ensuring that data persists even when a Pod is deleted. PVs can be backed by various types of storage solutions, including local disks, network storage, and cloud storage options. They are particularly useful for applications that require durable storage, such as databases, where it's crucial to maintain data across Pod deletions and restarts. By utilising PVs, users can ensure their data remains accessible and intact regardless of the state of the Pods that consume the storage. [9]

4. Persistent Volume Claim

A Persistent Volume Claim (PVC) is a request for storage by a user in Kubernetes. PVCs allow users to dynamically allocate storage as needed by specifying the desired size and access mode. The Kubernetes system then matches the PVC to an available PersistentVolume (PV) that meets the criteria. This abstraction simplifies storage management by allowing users to work with storage resources without needing to know the specifics of the underlying storage infrastructure. PVCs ensure that applications have the necessary storage resources and can be easily transferred or resized as requirements change.

5. ReplicaSet

In Kubernetes, Deployments manage the rollout and updating of applications by automatically handling ReplicaSets. A ReplicaSet's purpose is to ensure that a specified number of Pod replicas are running at any given time. This is achieved through reconciliation loops, processes that continuously monitor and adjust the current state of the cluster to match the desired state specified by the Deployment. For example, if a Deployment is configured to maintain 3 replicas of a web server Pod and only 2 are currently running, the reconciliation loop will detect this discrepancy and launch an additional Pod to meet the desired state, thus ensuring reliability and availability of the application. [10]

6. StatefulSet

A StatefulSet in Kubernetes is designed to manage stateful applications, which are those that maintain data or state across Pod restarts or replacements. Unlike stateless applications, stateful ones require a persistent storage mechanism to retain data, typically using disks or databases. StatefulSets provide unique, consistent identifiers for each Pod, starting from zero, which helps in maintaining a stable, persistent identity for each Pod and its associated resources. This feature is crucial for ensuring that when a Pod needs to be replaced—due to reasons like node failure—the new Pod inherits the same identifier and attaches to the same persistent volume as its predecessor. This mechanism allows stateful applications to preserve data and state, ensuring continuity and reliability of service, even in the face of Pod recreation or replacement. [11]

7. DaemonSet

A DaemonSet in Kubernetes is used to ensure that a specific Pod runs on all or a selected subset of nodes within the cluster. It is particularly useful for deploying system-level services like logging or monitoring agents that need to operate on every node for comprehensive data collection and analysis. Like ReplicaSets, DaemonSets are managed through reconciliation loops, which continuously monitor the cluster to ensure the current state matches the desired configuration. If a node is added to the cluster, the reconciliation loop automatically deploys the necessary Pod to that node, and if a node is removed, the Pod is also removed, maintaining consistent service operation across the cluster. [12]

8. Namespaces

Kubernetes namespaces are a feature that divides a single Kubernetes cluster into multiple virtual clusters, enabling resource isolation between them. This is particularly useful for managing different environments or versions of applications within the same physical cluster. For instance, separate namespaces can be created for different versions of an application, like "AppA-Namespace" and "AppB-Namespace", to deploy and manage their resources without interference. This approach ensures that even if the applications are nearly identical and use the same services or pod configurations, they remain logically isolated. Namespaces make it possible to test new versions in the same cluster alongside older versions, using the same resource definitions while preventing conflicts and interference, effectively offering the benefits of multiple Kubernetes clusters without the need for additional infrastructure. [13]

9. Service

A Kubernetes Service is a critical component that provides a stable and consistent point of entry for accessing a set of Pods that offer the same functionality within a cluster. Since Pods can be moved, recreated, or their IP addresses can change, directly connecting to them is impractical for clients due to the instability of their locations. A Service solves this problem by having a stable IP address through which clients can connect, with the Service then routing requests to the appropriate Pods. This not only ensures a stable endpoint for client connections but also facilitates simple load balancing by distributing incoming requests across all Pods evenly, preventing any single Pod from being overloaded while others remain idle. This mechanism simplifies application maintenance and update processes, as clients do not need to track changing IP addresses of individual Pods. [14]

10. Job

A Kubernetes Job is designed to execute short-lived, one-off tasks that must be run to completion, ensuring the task is successfully finished. This is particularly useful for tasks like database backups, which are not ongoing but need to be executed fully and correctly once initiated. Jobs are responsible for initiating one or more Pods to perform the task, monitoring their execution, and, if necessary, restarting Pods to ensure the task reaches successful completion. Kubernetes will manage these Jobs, automatically retrying failed tasks by creating new Pods until the job is completed successfully. For tasks that need to be executed periodically, Kubernetes offers CronJobs, which schedule Jobs to run at specified intervals. Jobs can vary in their requirements, with some needing just a single Pod to complete the task, while others may require multiple Pods working in parallel and completing successfully to consider the job done. [15]

11. ConfigMaps

Configuring apps refers to setting various parameters or options that control the behaviours of the apps. In Kubernetes, a ConfigMap is nothing more than a key/value pair. A ConfigMap store's non-confidential data, meaning no passwords or API keys and are used for:

- Kubernetes ConfigMap environment variables
- Kubernetes command-line arguments
- Configuration files in a volume

By using ConfigMaps and Secrets, you decouple the applications running in your Pods from their configuration values. This means you can easily update the configuration of your applications without having to rebuild or redeploy them. [16]

12. Secrets

Secrets are meant to hold sensitive configuration values, such as database passwords, API keys, and other information that only authorised apps should be able to access.

Secrets can be injected into Pods with the help of environment variables, command-line arguments, or configuration files included in the volumes attached to those Pods. [17]

2.5.2 Storage provisioning - NFS

The Network File System (NFS) is a mechanism for storing files on a network. It is a distributed file system that allows users to access files and directories located on remote computers and treat those files and directories as if they were local. [18][19]

For example, users can use operating system commands to create, remove, read, write, and set file attributes for remote files and directories.

Why We Need NFS

We need NFS because we have multiple nodes, and the PV (Persistent Volumes) does not have to be on the same machine as the pods pointing to it. Additionally, NFS provides the possibility of automatic scaling, which a local PV does not offer, and shared access.

Shared Access: NFS allows multiple pods to read and write to the same filesystem simultaneously. This is particularly useful for applications that need a common data source or for sharing files between different components of an application.

Scalability: NFS servers can be scaled to handle increased load by adding more resources or optimising configurations. This scalability helps in accommodating growing storage demands without significant changes to the underlying infrastructure.

In the context of Kubernetes, the Network File System (NFS) can be utilised in a more effective way through the concepts of storage provisioning and StorageClasses. Here's how NFS relates to these concepts and why it was our choice for our Kubernetes environment:

Storage Provisioning and NFS

Kubernetes can dynamically provision NFS volumes using an NFS provisioner. The NFS provisioner runs as a Pod that owns a pool of NFS volumes and handles the creation and deletion of volumes as needed by the Kubernetes environment. This setup simplifies managing shared resources in a distributed environment, which is often tedious with traditional storage solutions. [20]

This nfs provisioner deployment, which manages the automation of creating and deleting volumes as needed by Kubernetes, can be easily deployed using Helm or kubectl in case you need to edit some parameters. [21]

StorageClasses and NFS

In Kubernetes, a StorageClass allows administrators to define how a volume should be provisioned based on the characteristics of the underlying storage. By defining an NFS StorageClass, you can specify that certain types of workloads should be allocated storage that is provisioned from an NFS server. This makes it easy to manage and scale storage independently of Pods and services. [22]

The StorageClass for NFS might look something like this:

```
apiVersion: storage.k8s.io/v1
```

```
kind: StorageClass
metadata:
  name: nfs-negotiator-db
provisioner: fuseim.pri/ifs #Specify the name given to the provisioner that we previously created
reclaimPolicy: Delete
volumeBindingMode: Immediate
allowVolumeExpansion: false
```

2.5.3 Helm

Helm[23] uses a packaging format called charts. A chart is a collection of files that describe a related set of Kubernetes resources. A single chart can be used to deploy something simple, like a Nginx pod, or something complex, like a full web application stack with HTTP servers, databases, caches, etc.

Charts are created as files arranged in a specific directory tree. They can be packaged into versioned files for deployment.

A Helm chart is organised as a collection of files within a directory. The name of the directory is the name of the chart (without version information). Thus, a chart describing WordPress would be stored in a wordpress/ directory.

Within this directory, Helm expects a structure that matches this:

```
wordpress/
  - Chart.yaml          # A YAML file that contains information about the chart.
  - LICENSE             # OPTIONAL: A plain text file containing the chart's license.
  - README.md           # OPTIONAL: A human-readable README file.
  - values.yaml         # The default configuration values for this chart.
  - values.schema.json  # OPTIONAL: A JSON schema to impose a structure on the values.yaml file.
  - charts/             # A directory containing any charts upon which this chart depends.
  - crds/               # Custom Resource Definitions
```

23

```
- templates/            # A directory of templates that, when combined with values, will generate valid Kubernetes manifest files.
- templates/NOTES.txt # OPTIONAL: A plain text file containing brief usage notes.
```

Helm reserves the use of the directories charts/, crds/, and templates/, and the filenames listed. Other files will be left as they are.

Helm offers various advantages over Kubernetes like:

1. **Simplified Deployment**

   If for example you need to deploy a WordPress site with a MySQL database, without Helm, you would need to manually create multiple Kubernetes manifests (YAML files) for WordPress and MySQL, define their respective Services, configure persistent volumes, and ensure they connect properly.

   With Helm, you can use a single command to install a pre-configured WordPress chart that includes all these components:

   ```
   helm install my-wordpress stable/wordpress
   ```

2. **Simplified Upgrades**

   Upgrading applications deployed with Helm is straightforward. When a new version of a chart is available, you can upgrade your deployment with a single command. Helm will manage the upgrade process, ensuring that resources are updated correctly and in the right order:

   ```
   helm upgrade my-wordpress stable/wordpress
   ```

   Helm keeps a history of all releases, so if something goes wrong during the upgrade, you can easily roll back to a previous version with another command:

   ```
   helm rollback my-wordpress 1
   ```

3. **Reproducible Builds**

Helm charts can be versioned and stored in a chart repository, making it easy to share them and ensure consistent deployments across different environments. This versioning allows you to pin a specific version of a chart, ensuring that the same configuration and application versions are used every time you deploy.

4. **Dependency Management**

   Charts can define dependencies on other charts. For instance, a web application chart might depend on a database chart. Helm handles the dependency management automatically, ensuring that all required charts are installed in the correct order.

2.6 Best Practices for Creating Recipes

The creation of well structured and secure Kubernetes manifests is crucial for managing and deploying applications effectively in a Kubernetes environment. Here are some good practices that we have applied while creating our manifests:

Use of Environment Variables

Using environment variables allows you to separate configuration data from the application code. This means you can change the behaviour of your application without altering the codebase.

Kubernetes has two different resources for managing data that can be accessed by applications running in pods. These resources are ConfigMaps and Secrets.

- **ConfigMaps**

  ConfigMaps are used to store non-confidential data that can be consumed by pod containers or used to set environment variables. They are particularly useful for keeping your containerized applications portable and configurable without rebuilding your images.

- **Secrets**

  Secrets are similar to ConfigMaps but are specifically intended for storing sensitive information such as passwords, OAuth[24] tokens, and SSH keys. The data in a Secret is stored in Base64 encoding, which can provide basic obfuscation but should not be considered secure on its own—access to Secrets should be tightly controlled.

[24] https://auth0.com/es/intro-to-iam/what-is-oauth-2

To securely provide Secrets to pods, they should be mounted as volumes. Mounting Secrets as volumes is generally safer than exposing them as environment variables. Environment variables can inadvertently be exposed in crash dumps or logs, while file-based Secrets can utilise filesystem permissions for better access control. This approach minimises the risk of secret exposure and is a recommended practice over granting broad RBAC permissions to a pod's service account to access Secrets directly. To follow the good practices on kubernetes manifests, it is crucial to manage sensitive information in the correct way, and that means using Secrets instead of ConfigMaps for storing your most confidential data.

Use of Namespaces

Kubernetes namespaces are a fundamental concept used to organise and segregate cluster resources within a single Kubernetes cluster. By using namespaces, you can divide cluster resources between multiple users, applications, or environments.

Namespaces provide the following functionalities:

1. **Resource Management:** Namespaces help manage resources under a common naming scheme. This is particularly useful in environments with many users and teams.
2. **Scoping:** Namespaces provide a scope for naming resources, allowing you to reuse names across different namespaces.
3. **Access Control:** With namespaces, you can implement access controls using Kubernetes Role-Based Access Control (RBAC) more effectively. [22]
4. **Resource Quotas and Limits:** Administrators can apply resource quotas to namespaces, restricting the amount of compute resources (like CPU and memory) that can be consumed by resources in a specific namespace.

It's important to have in mind that even though the use of Namespaces provides a level of isolation by segregating resources, they do not offer complete isolation. Network policies and other security mechanisms should be used to enforce stricter isolation if needed.

Use of StatefulSets, ReplicaSets and Deployments

Using high-level controllers in Kubernetes, such as ReplicaSets, Deployments, and StatefulSets, instead of directly managing individual Pods, offers several key advantages:

Automated Resilience: Controllers automatically handle the lifecycle of pods, ensuring that the specified number of pods are always running. They replace failed pods without manual intervention, enhancing the resilience and availability of applications.

Simplified Scaling: Controllers allow for easy scaling of applications. You can increase or decrease the number of pods dynamically, without needing to manage each pod individually.

Efficient Updates and Rollbacks: High-level controllers manage the rollout of updates and can automatically rollback to a previous version if an issue arises during deployment, ensuring continuous availability and minimising downtime.

Overall, using high-level controllers significantly simplifies the management of Kubernetes resources, enhances application reliability, and provides robust tools for handling the deployment and scaling of applications in a cluster.

Automating with Workflows

The use of workflows and CI/CD (Continuous Integration/Continuous Deployment) processes substantially enhance the effectiveness, reliability, and security of software development and deployment. [23]

In this project, we have utilised the GitHub Actions platform to achieve these purposes. Thanks to this, in addition to facilitating testing and deployment of recipes, we have also been able to implement automatic code quality reviews. These reviews inform us whether the code we intend to publish on GitHub—and that will be automatically deployed to our cluster—is truly ready, or if it needs to be revised in any way.

We achieve this using different tools provided by the GitHub Actions platform that would be explained more in detail in the section 5.4 Automation in a CI/CD environment.

2.7 Extern tools

2.7.1 Required tools and images

This section mentions some of the main tools used to carry out this deployment. While they are not indispensable, as they can be replaced by other tools performing similar tasks, their functionality is absolutely necessary in our case because these tools enable us to perform operations that are crucial for the correct functioning of our deployment. This tools are the following:

HASHICORP VAULT

HashiCorp Vault is an identity-based secrets and encryption management system[25]*.* It provides encryption services that are gated by authentication and authorization methods to

[25] https://developer.hashicorp.com/vault/docs/what-is-vault

ensure secure, auditable and restricted access to secrets.
A secret is anything that you want to tightly control access to, such as tokens, API keys, passwords, encryption keys or certificates. Vault provides a unified interface to any secret, while providing tight access control and recording a detailed audit log.

In this project, Vault has been used to generate, sign and store various certificates, providing us with a secure authentication mechanism. In a distributed and federated environment like ours, where multiple platforms coexist, this is essential to prevent potential attacks or fraud.

Upon initialization, Vault provides you with an access key that you must use from that point forward to perform all configuration tasks for Vault and to carry out actions such as certificate generation and signing. This key ensures that only authorised individuals have access to the certificates. These features and some more are discussed in greater detail in section 5.1.1 Vault.

ELASTICSEARCH - ELK STACK

“Elasticsearch is a distributed RESTful search and analytics engine capable of addressing an increasing number of use cases. As the core of the Elastic Stack, it centrally stores your data for lightning-fast search, refined relevance, and powerful analytics that scale effortlessly.”[26]

Some of its key features include:

- Indexing and Search: ElasticSearch stores data in indexes and allows for complex and fast searches on those indexes using a powerful search API.
- Scalability: It is designed to be horizontally scalable, meaning it can handle large amounts of data distributed across multiple servers.
- Real-Time Analysis: It enables real-time data analysis and aggregations, which is useful for use cases such as monitoring, log analysis, and time series analysis.
- High Availability: It supports data replication, ensuring high availability and fault tolerance.
- RESTful API: It provides a RESTful interface for interacting with data, making it easy to integrate with other systems and applications.

In our application Elasticsearch is used by Molgenis for indexing the data layer and by the Monitoring service that provides an overview of the status of the different EUCAIM components. The Monitoring service architecture is composed of 6 components of the technological stack Elasticsearch-Logstash-Kibana (ELK stack), which are deployed in a

[26] https://www.elastic.co/es/elasticsearch

Kubernetes cluster using the operator pattern[27]. More information about this deployment can be found at the section 4.7 Monitoring

MOLGENIS

"MOLGENIS is a modular web application for scientific data.[28]". It has a frontend and a backend. You can develop on them separately. If you want to develop an API and an App simultaneously you need to checkout both[29].

The MOLGENIS[30] application is meant to provide researchers with user-friendly and scalable software infrastructures to capture, exchange, and exploit the large amounts of data that is being produced by scientific organisations all around the world.

In this project, we have used both (frontend and backend) Molgenis images to create our image catalogue, where metadata is stored to provide descriptive information about the available datasets to users. Both images can be found at: https://hub.docker.com/r/molgenis/

NEGOTIATOR

The Negotiator is another key tool within the core services of EUCAIM. Developed by BBMRI-ERIC[31], the Negotiator provides an efficient communication platform for biobankers and researchers requesting samples and/or data. It substantially simplifies the communication steps necessary to obtain information on the availability of relevant samples/data, especially if researchers need to communicate with multiple candidate biobanks. In our project it is in charge of allowing EUCAIM registered users to get an overview of the previously selected datasets. You can find a description of the setup and the latest versions of the Negotiator at: https://gitlab.bbmri-eric.eu/negotiator-deployment/negotiator-deployment-template/-/tree/main

POSTGRESQL

PostgreSQL[32], also known as Postgres, is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads.

[27] Kubernetes operator pattern: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/

[28] https://molgenis.gitbook.io/molgenis

[29] https://github.com/molgenis/molgenis

[30] https://molgenis.org/

[31] https://www.bbmri-eric.eu/

[32] https://www.postgresql.org/about/

It is a crucial tool in our platform as it is responsible for managing the Negotiator and Molgenis databases.

The image used in our deployment can be found at: https://hub.docker.com/r/bitnami/postgresql

SAMPLY BEAM

Samply.Beam[33] is a distributed task broker designed for efficient communication in strict network environments. It offers commonly used communication patterns, end-to-end encryption, and digital signatures, along with certificate management and validation, all accessible via a user-friendly REST API. Besides task/response capabilities, Samply.Beam supports high-performance applications through encrypted, low-level direct socket connections.

Despite the obvious difficulties in configuring an environment like ours, combining Vault and Samply Beam to achieve a federated search environment like the Explorer, we have installed Samply Beam in our Explorer deployment. This is because it offers some of the most notable features —end-to-end encryption and signatures, as well as certificate management and validation— for building a federated environment like the Explorer.

GRAFANA

Grafana[34] is open-source software licensed under the Apache 2.0 licence, which allows for the visualisation and formatting of metric data. It enables the creation of dashboards and graphs from multiple sources, including time series databases such as Graphite, InfluxDB, and OpenTSDB.

In this work, Grafana has been used to display the metrics of the load test results obtained through Prometheus.

PROMETHEUS

Prometheus collects and stores its metrics as time series data, meaning that metrics information is stored with the timestamp at which it was recorded, along with optional key-value pairs called labels. It is an open-source systems monitoring and alerting toolkit that was initially developed at SoundCloud. Prometheus is now an independent open-source project, maintained separately from any company. To emphasise this and clarify the project's governance structure, Prometheus joined the Cloud Native Computing Foundation in 2016 as the second hosted project, following Kubernetes.

For this project, we have used Prometheus to extract metrics from our Kubernetes cluster and send them to Grafana for visualisation.

[33] https://github.com/samply/beam

[34] https://grafana.com/

Backend solutions

OPENSTACK

In this project we used OpenStack as a virtual machine management platform.

OpenStack is a cloud operating system that manages extensive pools of compute, storage, and networking resources across a datacenter. It uses APIs with common authentication mechanisms for management and provisioning. Additionally, it offers a dashboard that allows administrators to control resources and enables users to provision them through a web interface. Beyond the standard infrastructure-as-a-service capabilities, OpenStack includes additional components for orchestration, fault management, and service management, ensuring high availability for user applications[35].

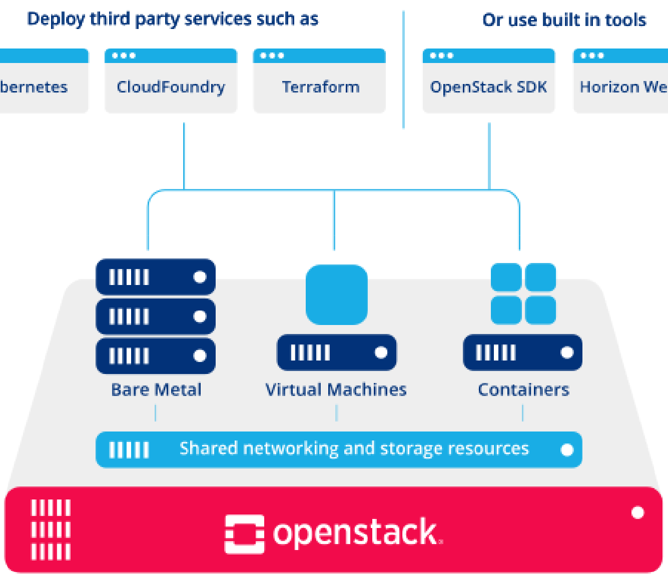


Figure 2. *OpenStack schema*

[35] https://www.openstack.org/software/

3 Problem Analysis

3.1 Proposed Solution

The proposed solution to achieve the objectives outlined in section 1.2 Objectives includes the following key aspects:

Firstly, to increase the level of automation in this project, a CI/CD solution has been integrated using GitHub Actions. This will facilitate the tasks of the developers by providing automated workflows for building, testing, and deploying the application. With GitHub Actions, every change in the codebase will trigger a series of automated steps, ensuring that code quality is maintained and deployments are consistent.

Secondly, by implementing best practices in our deployment processes, we ensure that our infrastructure is not only reproducible but also secure. When we refer to best practices, we mean:

- Replacing environment variables in the manifests with ConfigMaps or Secrets.
- Using Namespaces to ensure organisation and segregate cluster resources.
- Replacing Pods with high-level controllers in Kubernetes, such as ReplicaSets, Deployments, and StatefulSets, to ensure automated resilience, efficient updates and rollbacks, and simplified scaling.

Apart from this, the aforementioned use of workflows will also substantially enhance the effectiveness, reliability, and security of software development and deployment.

Lastly, the deployment process has been optimised, ensuring efficient, reliable, and secure updates to the platform. To achieve this, the architectural solution known as blue-green deployments has been applied, which allows us to gradually transfer user traffic from a previous version of an app or microservice to a nearly identical or updated new release—both of which are running in production.

All these aspects and changes have been introduced in the previous deployment of EUCAIM and have been transferred to a new production machine.

All the details about the solution's architecture and its implementation are provided in the following sections. Thank you very much, and I hope you enjoy the reading.

3.2. Requirements Specification

Due to the complexity of the platform and the wide range of actions that users can perform, this section will provide a concise explanation of the most relevant actions users can undertake. It will also include a brief distinction between the different types of users who can access the platform and the roles corresponding to each based on their functions.

User Roles

Depending on their position and functionalities within the platform, a user can have the following different roles:

1. **Data Holder[36]/Data Provider/Data Controller**

   EUCAIM Data holders are the entities with the right or ability to make data available, such as research organisations, clinical centres, and biobanks. They can either set up a federated node or upload anonymized data to Central Storage, adhering to EUCAIM's specifications and data sharing agreements. Data Holders must ensure data quality, compliance with privacy regulations, and use standard schemas for interoperability, supported by a Local Data Manager for operational tasks.

2. **Software Provider:**

   A Software Provider in the EUCAIM platform refers to entities such as startups, enterprises, research institutions, and nonprofits that contribute processing tools, services, and applications for federated processing or data pre-processing. They must ensure compliance with technical standards, GDPR, and risk analysis, providing evidence of adherence to data protection and trustworthy AI guidelines. Tools must include clear documentation, be compatible with federated data, and respect privacy, security, and ethical standards. Providers must also comply with terms related to intellectual property, licensing, and revenue sharing.

   Example: A startup offering an AI explainability platform for real-time model analysis, bias detection, and data validation.

3. **Data User-Researcher**:

   EUCAIM Data User-Researchers are the individuals that explore the public catalogue of available (meta)data and also can request access to data for processing with platform tools or their own AI tools. Data access requests must be made under an approved R&D project or an observational study proposal, both subject to evaluation

[36] Proposal for a REGULATION OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL on the European Health Data Space COM/2022/197 final

by the Access Committee. Data User-Researchers aim to generate new knowledge in medicine and publish findings, often requiring specialised datasets or tools for complex analysis. Data Scientists, a specific profile within this group, focus on developing and testing statistical or machine learning models using specialised data mining and analysis skills.

Example of a Data User-Researcher:

A researcher leading a project on prostate cancer, aiming to improve treatment allocation based on baseline MR images, incorporating AI tools for result validation and clinical application.

4. **Research Communities**

   Research Communities refer to groups or entities with a common research goal, typically formed through completed, ongoing, or newly emerging projects. They aim to use EUCAIM's research environment to continue and expand their initial research efforts. Such communities, like consortia, must agree to share their collected data and developed tools with EUCAIM's Central Repository. This collaboration allows the research community to remain connected via EUCAIM, furthering their work and initiating new projects within a secure and interoperable environment. EUCAIM includes these datasets in its catalogue, facilitating new collaborations with other connected partners.

   Example of Research Communities: AI4HI[37] is a European group of researchers, data scientists, and healthcare professionals focused on applying artificial intelligence techniques to improve healthcare outcomes. As a Research Community within EUCAIM, AI4HI contributes entire finished projects, such as PRIMAGE[38], that involve developing AI algorithms for medical image analysis, clinical decision support systems, and predictive models for two types of paediatric cancer diagnosis, prognosis, and therapies follow-up. Thus, the entire finished project could be transferred to the Central Hub, so that the data can still be used and they can further contribute to the project. They share their expertise, data, and tools with other members of the EUCAIM federation, promoting collaboration and innovation in the field of AI-driven healthcare solutions through new research projects, engaging new partnerships.

Management Profiles

In addition to user roles, there are two management profiles that will interact with the platform, described as follows:

1. **EUCAIM Platform Manager**:

   The EUCAIM Platform Manager is a technical expert or team responsible for operating the core services of the EUCAIM platform. This role includes managing

[37] https://ai4hi.net/

[38] https://www.primageproject.eu/

and maintaining the platform's technical infrastructure, such as central storage, servers, databases, and other resources. Key responsibilities include:

User Management: Managing user accounts and access permissions.

Application Deployment: Deploying applications and services, uploading new applications to the marketplace (provided by Software Providers), and ensuring their proper integration.

Federated Processing Orchestration: Collaborating with Data Holders and Software Providers to integrate metadata, tools, and services, and ensuring that Data User-Researchers' queries are properly executed.

Support and Documentation: Providing user support, responding to inquiries, providing documentation, and troubleshooting platform issues.

Example of an EUCAIM Platform Manager:

A system administrator employed by the institution operating the central storage, possessing technical and project management skills, along with knowledge of data management principles such as data storage, data integration, and data governance. They understand data privacy and security regulations and have a basic understanding of machine learning and artificial intelligence, including federated learning principles. This enables them to support the orchestration of federated learning processes.

2. **Governing Body**:

   The Governing Body is a decision-making board within EUCAIM that plays a crucial role in the coordination, governance, and operation of the federated infrastructure. The composition and decision-making processes of the Governing Bodies may be subject to modifications throughout the project's lifespan.

   Example of a Governing Body: EUCAIM Access Committee, this committee is responsible for accepting or rejecting data access requests from Data Users-Researchers. For instance, when a request to access specific datasets from the Atlas of Cancer Images is submitted, the Access Committee receives an email notification via the Negotiator service. The committee then follows the established procedure to evaluate the request, based on the type of agreement signed with the Data Holder organisation. Any rejection by the Access Committee is thoroughly justified in writing, following objective criteria defined in its internal procedures.

3.2.1 Functional Requirements

Within the functional requirements, we find the set of use cases that describe all the interactions that users will have with the software. In these use cases, we identify all the services that the system will need to provide once completed. As described in the previous section, our system comprises a wide variety of users. The following sections explain the use cases for Data User-Researchers and Access Committee users in the central core services of EUCAIM platform.

Exploration of datasets from the Public Catalogue

Data User-Researcher

Preconditions:

- Valid user account, registered and validated through the authentication and authorization service (AAI).
- Metadata datasets must have been uploaded by Data Holders onto the Public Catalogue

Postconditions:

- Datasets meeting the specified criteria are successfully identified by using basic search and filtering options.
- The available metadata from the exposed datasets can be visualised.

Use Case example:

Alice, a researcher from a research group, aims to prepare a research project based on medical images of prostate cancer. As an anonymous user of the EUCAIM platform, she explores diverse metadata datasets provided by federated Data Holders to perform an initial evaluation of their relevance and suitability for her research.

1. **Exploration:**
    - Alice navigates through the Public Catalogue within the EUCAIM platform, using basic search and filtering options.
    - She identifies datasets that align with her research interests, such as datasets containing breast cancer cases where the cohort's age is between 40 and 50.
    - She discovers that the INCISIVE[39] and CHAIMELEON[40] datasets satisfy her criteria.
2. **Metadata and Access Conditions:**
    - Alice views the access conditions for each dataset offered by the Data Holders.
3. **Registration for Full Access:**

[39] https://incisive-project.eu/
[40] https://chaimeleon.eu/

- ○ To view the complete metadata datasets available in the user's catalogue, Alice registers on the platform via the Life Science AAI and accepts the Terms of Usage and Privacy.

This process allows Alice to assess the available datasets and determine their potential use for her research project, ensuring efficient and streamlined access to relevant data. The visual process of this action can be seen in Figure *3*

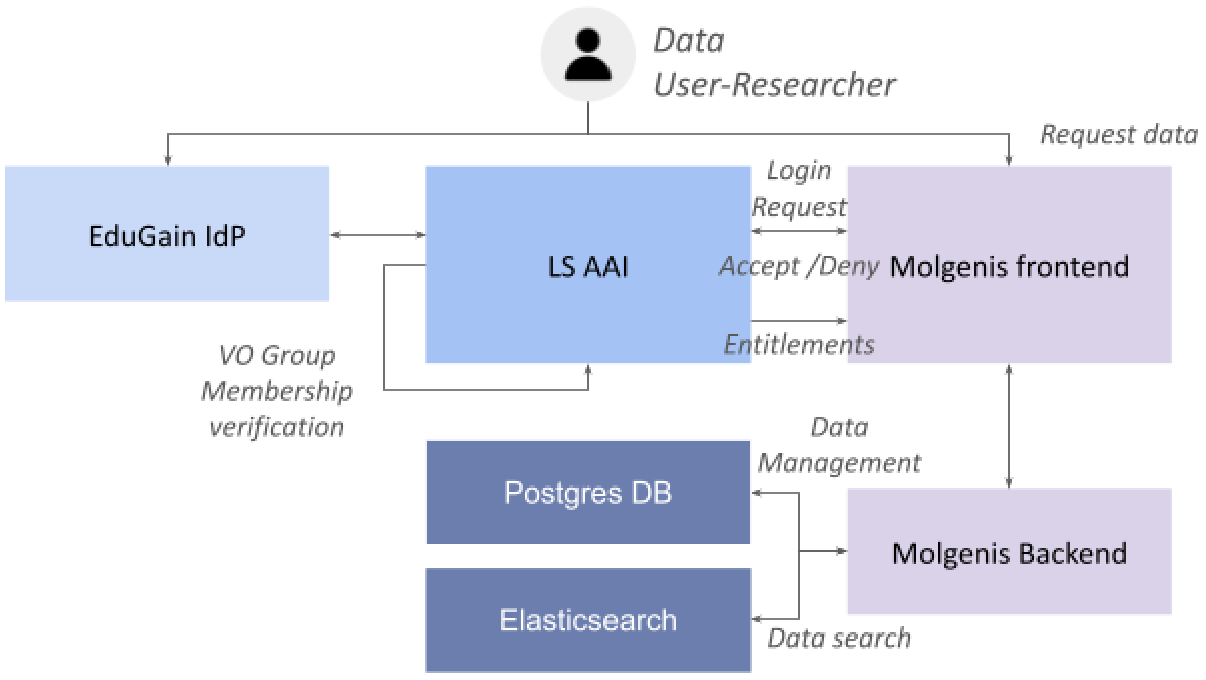


Figure 3. *Exploration of datasets from the Public Catalogue.*

Federated search of aggregated data in the datasets

Data User-Researcher

Preconditions:

- Valid user account, registered and validated through the authentication and authorization service (AAI).
- Federated nodes, including the central repository, are interconnected and functional through the mediator component.

Postconditions:

- Federated query is successfully performed over multiple sources.

- Aggregated results and a downloadable report are obtained based on the hyperontology.
- Datasets meeting the specified criteria are successfully identified by using advanced search and filtering options, through federated queries.

Use Case example:

After signing up on the EUCAIM platform, Alice logs into the EUCAIM Dashboard (https://explorer.eucaim.cancerimage.eu/) to explore the complete metadata catalogue and perform an advanced search using available filters like cancer type, imaging modality, and annotation availability. She executes a federated query over various nodes, including the central repository, based on hyperontology concepts. This detailed search allows her to identify datasets and the number of cases that meet her criteria. For instance, Alice searches for breast cancer datasets with a specific TNM stage and MR series from a specific manufacturer, discovering 123 cases in the INCISIVE dataset and 400 cases in the CHAIMELEON dataset. She downloads a report of these numeric aggregated results to attach to her research proposal for the EUCAIM access board, which will include her study's objectives and methodology. The schema of this flow can be seen in Figure *4:*

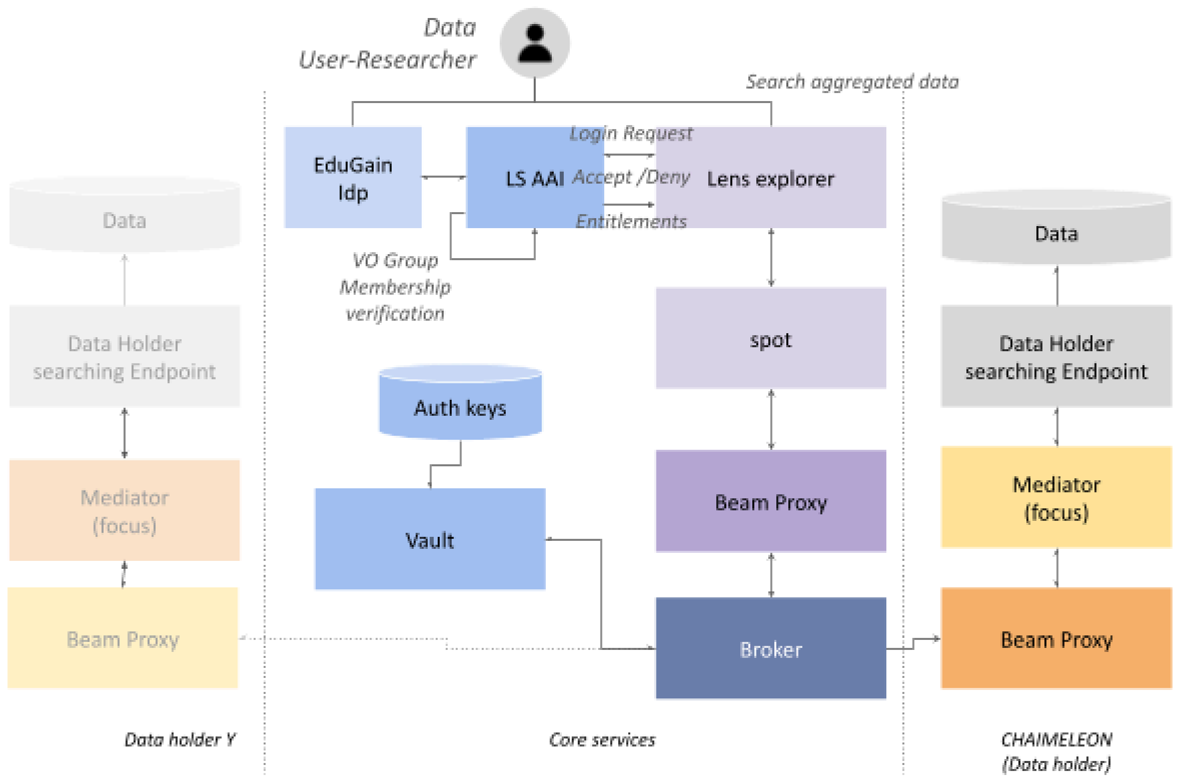


Figure 4. *Federated search of aggregated data in the datasets schema.*

Request access to the datasets from the User’s Catalogue

Data User-Researcher + Access Committee

Preconditions:

- Login is performed using the AAI (Authentication and Authorization Infrastructure).
- Datasets meeting some specified criteria have already been successfully identified.
- A research project approved by an Ethical Committee for submission to the Access Committee, under which the requested data are to be used.

Postconditions:

- The access request application has been submitted correctly through to the Negotiator.
- The Access Committee reviews the access request and makes a decision.
- Once the access request is approved, access to the federated data is granted to the user of the data.

Use Case example:

In the EUCAIM platform, Alice and Leo, both Data Users-Researchers (with Leo having a Data Scientist profile), need access to federated data for their research projects. They identify relevant datasets through the user catalogue and federated queries. They then use the Negotiator tool to submit access requests to the Access Committee. The committee reviews these requests for compliance with the project's purpose, data privacy regulations, and policies. Sometimes, the Data Holder must also approve the request. Upon approval, Alice and Leo gain access to the federated data, enabling them to enhance their research with diverse datasets.

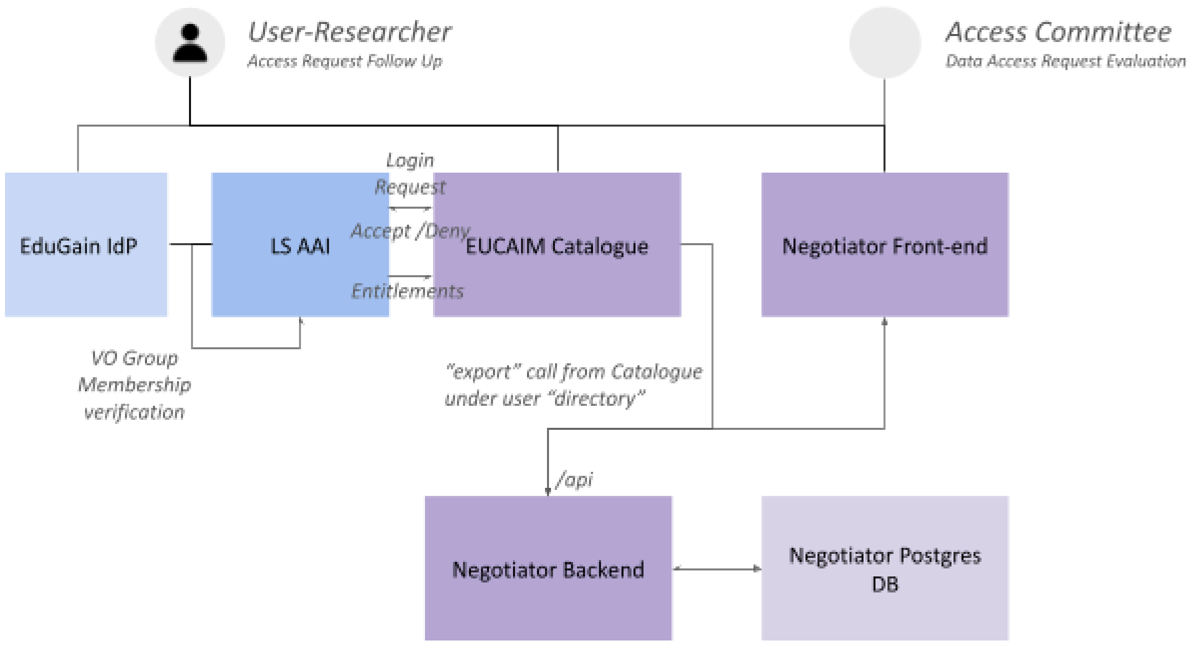


Figure 5. *Request access to the datasets from the User's Catalogue.*

Get an overview of the datasets to which they have been granted access

Access Committee

Preconditions:

- Login is performed using the AAI (Authentication and Authorization Infrastructure).
- Access request applications within the Negotiator have already been evaluated and successfully granted.

Postconditions:

- Metadata and details of datasets granted access can be successfully reviewed, as well as the information on any restrictions or conditions applicable to the datasets.
- Informed decisions regarding the selection and use of the available data resources can be performed by the data user.

Use Case example:

Leo, a Data Scientist and Data User-Researcher on the EUCAIM platform, aims to perform image analysis and processing. She uses advanced search and filtering in the user's catalogue to find datasets that meet her research objectives. After gaining access approval, Leo explores the datasets in the User's Area, selects the necessary subsets, and uses the Processing Services to develop models, conduct detailed analysis, and obtain valuable insights for her research.

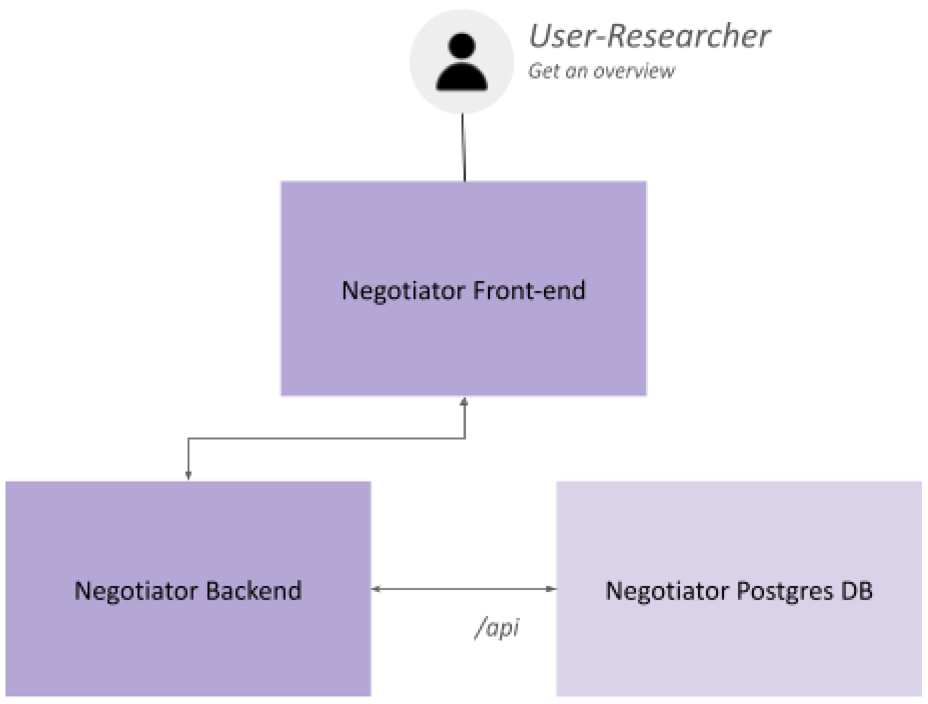


Figure 6. *Get an overview of the datasets to which they have been granted access*

Evaluate and accept/reject datasets from Data Holders

Data User-Researcher + Access Committee

Preconditions:

- An application has been made by a Data Holder

Postconditions:

- The Access Committee accepts/rejects the application.

- If accepted, new data is incorporated into the EUCAIM federation.

Use Case example:

Raul sends an application for providing datasets to be incorporated within EUCAIM. The AC makes a decision of acceptance or rejection, supported by the internal governance bodies on ethics and legal compliance. Raul receives a reply as soon as possible.

Evaluate and accept/reject a data access request application

Data User-Researcher + Access Committee

Preconditions:

- An application has been made by a Data User-Researcher.

Postconditions:

- The Access Committee accepts/rejects the application.
- If accepted, a research project is carried out with data available from EUCAIM.

Use Case example:

Leo (a Data Scientist) or Alice (a Researcher) identifies specific datasets needed for their research project using the Public Catalogue of the Atlas of Cancer Images. They log into the User's Area, review the instructions for the required documents and the response time from the Access Committee (AC). After preparing and submitting the necessary documentation, the AC receives an email notification via the Negotiator service. The AC then follows an established procedure to evaluate the request. If the request is rejected, the AC provides a written justification based on objective criteria from their internal procedures. The AC members are appointed by the EUCAIM Management Board for set terms and are periodically renewed.

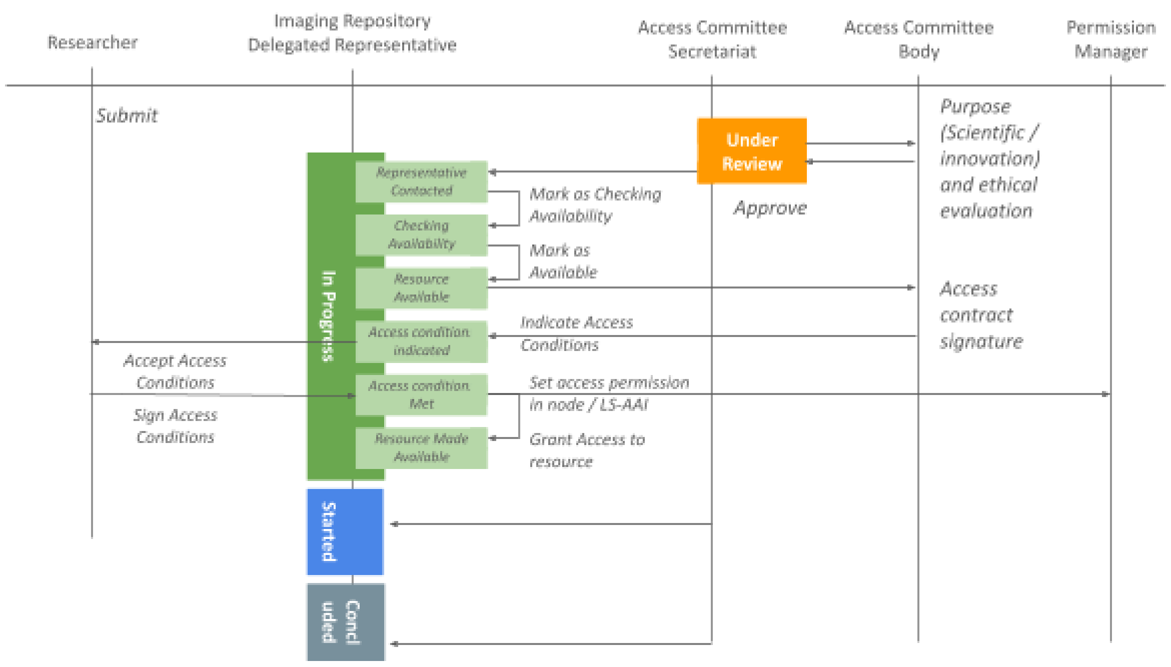


Figure 7. *Evaluate and accept/reject a data access request application*

3.2.2 Non-functional Requirements

3.2.2.1 Security

Authentication and Authorization

In our case, both user authentication and authorization for our platform are managed by LS-AAI, an Authentication and Authorization Infrastructure (AAI) that allows developers to largely delegate the tasks associated with ensuring authentication and authorization. Detailed configuration of this tool in our environment can be found in section 3.2.2.1 Security.

In addition to the necessary authentication and authorization of platform users, it is also crucial to authenticate and authorise data providers. Vault handles this process. By having the certificates of other centres signed within it, Vault can verify whether the hosts attempting to access our services are legitimate. This ensures that all nodes in our network of data holders are who they claim to be, thereby preventing potential attacks.

Monitoring

The Monitoring EUCAIM service provides an overview of the status of the different EUCAIM components by making requests to the associated web services at certain periods of time. In addition to this, the service is also capable of sending notifications to the person in charge of a EUCAIM component when one of the predefined rules is fulfilled. This service is explained in more detail at 4.7 Monitoring

Apart from the Monitoring service, each of the EUCAIM core services has its own volume where logs are stored. These logs enable audits to be conducted as needed to ensure compliance with security policies and to detect potential incidents.

3.2.2.2 Performance

Scalability

Although our system resources do not scale automatically, which could be an improvement to consider for future versions of the infrastructure, as seen in the load tests in section 6.1 Load Testing of Services, the computing capacity of the application meets the estimated load, and if exceeded, it can be easily scaled horizontally by adding new replicas to the Kubernetes deployment.

Response time

The system must respond within an appropriate time frame for usability. This feature is tested in 6.1 Load Testing of Services.

3.2.2.3 Availability

High Availability[41] and Resilience

Our infrastructure relies on the virtual machines where our Kubernetes nodes are running. These Kubernetes nodes consist of 1 frontend node and 2 working nodes. While a failure in any of these virtual machines or the physical node hosting them would result in a connection loss due to the centralised architecture, it would not be costly to restart our infrastructure on any other physical node or cloud alternatives such as AWS[42] or Azure[43].

For these reasons, a deployment test on AKS Azure Kubernetes Services is proposed for future work.

3.2.2.4 Usability

User Interface

[41] https://www.redhat.com/es/topics/linux/what-is-high-availability

[42] https://aws.amazon.com/es/

[43] https://azure.microsoft.com/es-es

The dashboard interface is designed to be as intuitive as possible, allowing users to navigate and find the information they need quickly. The interfaces of the other services are still being improved for upcoming project milestones.

Key usability aspects such as clearer feedback and greater customization options are also being worked on.

Documentation and Support

The platform is compiling a substantial amount of documentation at each milestone, explaining the functioning of each component in detail. Additionally, each participating association or company offers constant support for their respective tools.

3.2.2.5 Interoperability

Compatibility

This platform is highly compatible. As can be seen in its composition, it comprises a wide variety of applications and technologies that integrate efficiently and function together seamlessly.

The platform's metadata must follow the health specification of DCAT-AP[44] (Data Catalog Vocabulary Application Profile for data portals in Europe) based on the W3C Data Catalog Vocabulary[45] (DCAT) standard. DCAT-AP is designed to facilitate the interoperability of data catalogues published on the web, particularly in the context of open data portals in Europe and provides guidelines and standards to ensure that data catalogues from different organisations and countries can be easily discovered, accessed, and used. part from this, the platform's data must adhere to the project's hyper-ontology[46], which serves as an advanced and organised structure that integrates and manages large volumes of data coherently and efficiently, thus facilitating the integration of data from multiple sources and the interoperability of both data and applications from different providers and disciplines through the use of common standards and protocols.

In addition to these aspects, the EUCAIM platform facilitates interoperability between various research infrastructures and medical data across Europe. Its modular design allows the incorporation of new systems and technologies without affecting the platform's stability or performance, ensuring continuous evolution and adaptation to the changing needs of research.

[44] https://datos.gob.es/es/documentacion/dcat-ap-perfil-de-aplicacion-de-dcat-para-portales-open-data-europeos

[45] https://www.w3.org/TR/vocab-dcat-2/

[46] EUCAIM's Hyper-Ontology_V1.0 (zenodo.org)

Integration

Integration is another strong point of EUCAIM. Its infrastructure allows researchers to access a vast amount of information in a coherent and unified manner. This integration is achieved through open standards and common protocols, ensuring that data and applications from different providers and disciplines can interoperate smoothly.

3.2.2.6 Maintainability

Modularity

Thanks to the modular architecture (implemented with Kubernetes) of our platform and the continuous integration and delivery (CI/CD) pipelines, we can update and improve individual components without affecting the overall functioning of the platform, thereby facilitating the implementation of new features and problem resolution.

Documentation

Throughout this work, various aspects of the functioning and composition of this infrastructure are explained. Additionally, throughout the development of this project, each of the applications that compose it and each of the functionalities have been thoroughly documented, allowing future developers and users to interact with the platform more easily and effectively.

3.2.2.7 Legal and Regulatory

Compliance with Regulations and Data Privacy

As mentioned in section 2.3 Medical Data Situation in Europe, this infrastructure is required to comply with essential data protection requirements within the European community.

4 Solution Design

4.1 General Architecture

The general architecture of EUCAIM consists of the following core services:

- Dashboard
- Catalogue
- Federated Search
- Negotiator
- Monitoring

Although each of these services is accessible from its own URL, the main access point to the platform is the Dashboard (https://catalogue.eucaim.cancerimage.eu/#/).

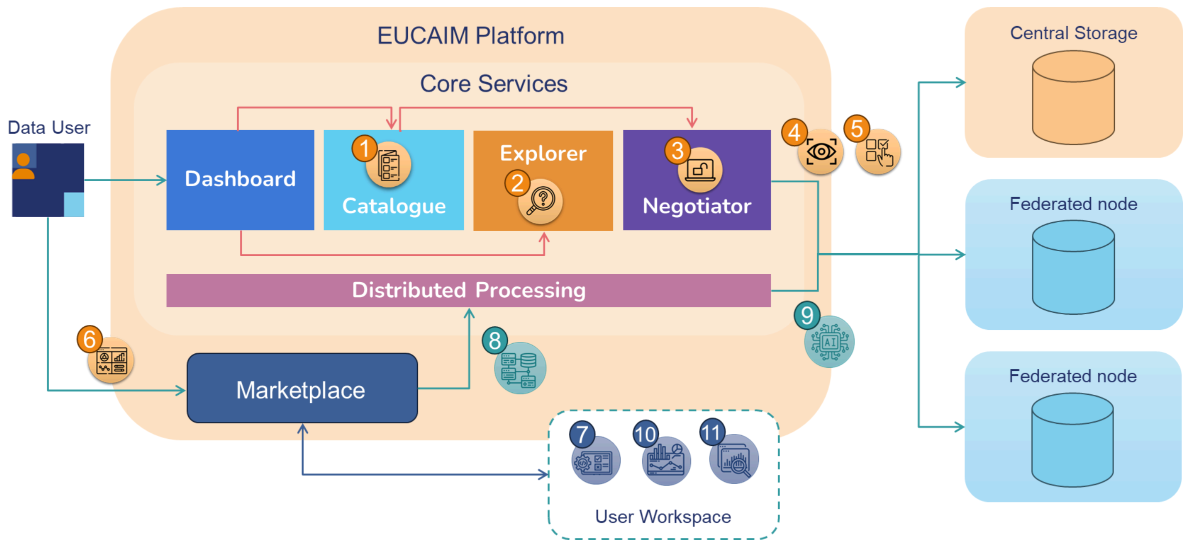


Figure 8. *EUCAIM Core Services Architecture*

To be able to perform tasks on the platform, users must be registered, for which they will need to authenticate via the Life Science Login (LS LOGIN). The Life Science Login enables researchers to use their home organisation credentials or community or other identities (e.g., Google, LinkedIn, LS ID) to sign in and access the data and services they need. It also allows service providers (both in academia and industry) to control and manage access rights of their users and create different access levels for research groups or international projects.

Apart from this, it is important to note technical requirements for integration into the federated infrastructure rely on the model of tiers:

Tier 1: Compliance with the metadata model for the datasets.

Tier 2: Direct (through adoption) or indirect (through a mediator component) compliance with the data model for searching purposes.

Tier 3: Direct (through adoption) or indirect (through a mediator component) compliance with the data model for processing purposes.

Figure 9 shows the diagram of the general architecture as well as the interactions between each of its components.

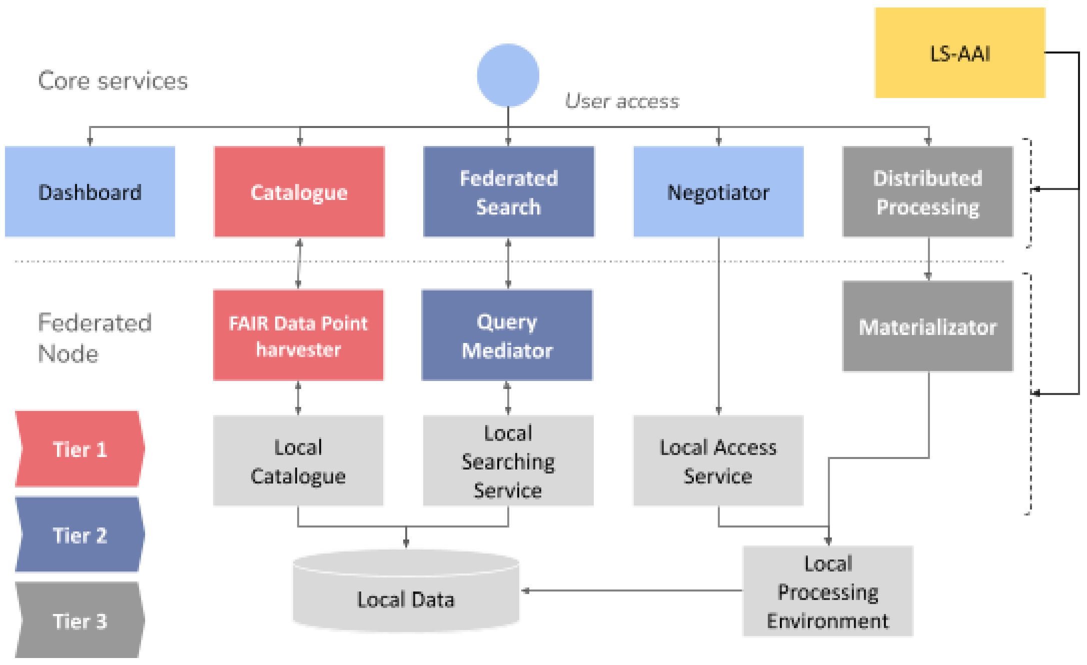


Figure 9. *EUCAIM Core Services detailed schema*

If we analyse the architecture in greater detail, it can be observed in Figure 10 that each of these components is composed of two simultaneous deployments. This technique or application release model is known as Blue-Green Deployment and allows us to gradually transfer user traffic from a previous version of an app or microservice to a nearly identical or an updated new release—both of which are running in production. The old version can be referred to as the blue environment, while the new version can be known as the green environment. So, whenever we have a new version of any service we wish to deploy, we simply need to scale down the old deployment and add the necessary replicas to the deployment that contains the new version.

Once production traffic is fully transferred from blue to green, the blue environment can stand by in case of a rollback, or be removed from production and updated to become the template upon which the next update is made. [24]

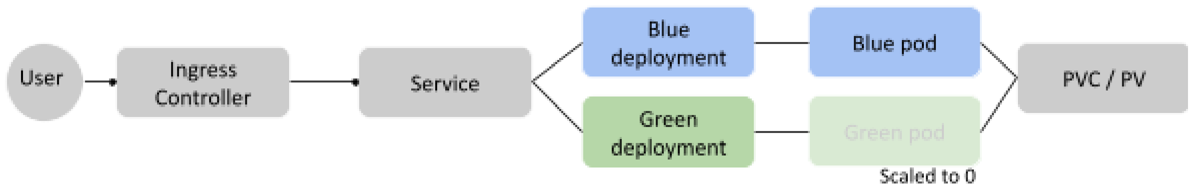


Figure 10. *Green-Blue Deployment*

4.2 Security Model

This section briefly describes how we handle the login process on our platform. EUCAIM services have public (anonymous) and restricted (requiring authorisation) services.

Public services can be accessed without an account. Access to restricted services requires valid and duly authorised credentials.

User authentication and authorization are two key aspects when it comes to accessing sensitive data. In EUCAIM, these two aspects are managed by the LS AAI endpoint. LS-AAI is the commonly agreed AAI (Authorization and Authentication Infrastructure) framework for Life Sciences Research Infrastructures. It relies on the AARC blueprint and supports the eduGAIN Federation (which serves most academic and research organisations in Europe), as well as other public Identity Providers. Authentication is performed through membership in the EUCAIM VO Group[47].

As mentioned in the previous paragraph, a EUCAIM user will need to create a LS-AAI account and to request membership to the EUCAIM group. This process has to be performed only once and can be initiated through the EUCAIM Platform Dashboard (https://dashboard.eucaim.cancerimage.eu).

[47] Enrollment URL for the EUCAIM VO Group
https://signup.aai.lifescience-ri.eu/fed/registrar/?vo=lifescience&group=communities_and_projects:EUCAIM

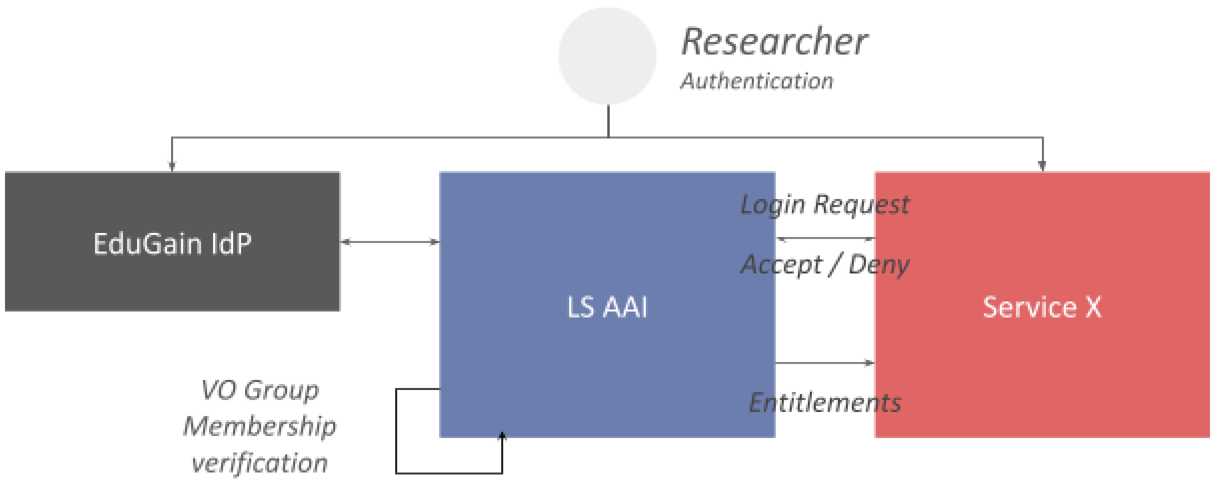
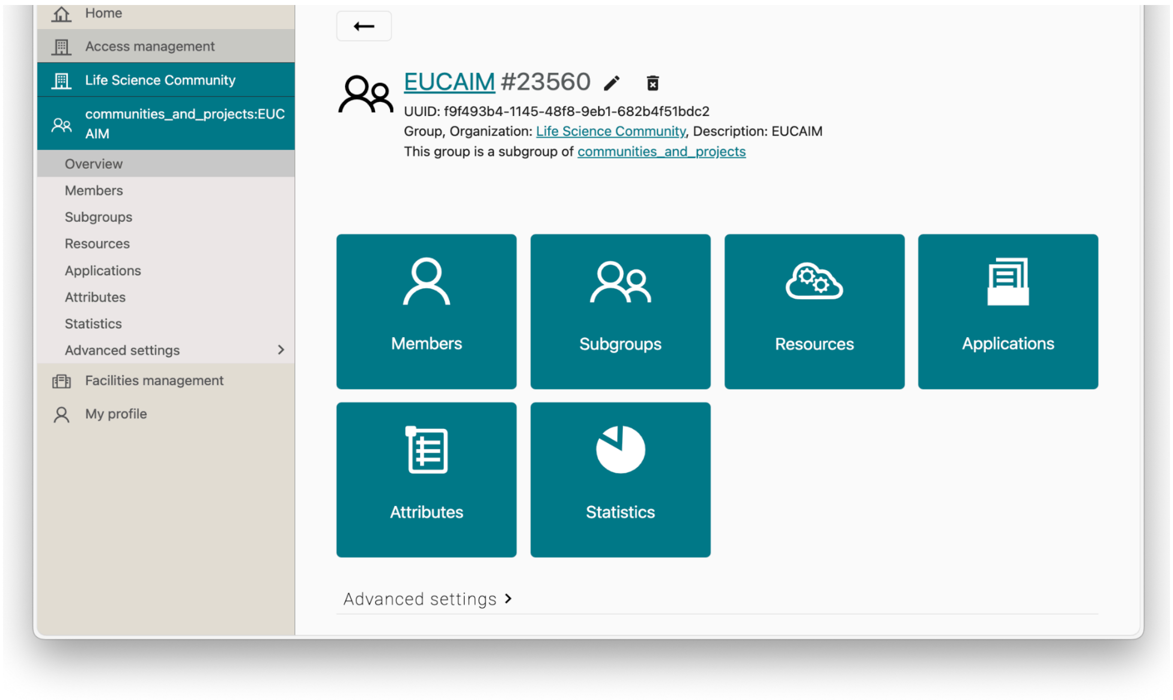


Figure 11. *Security Model Schema*

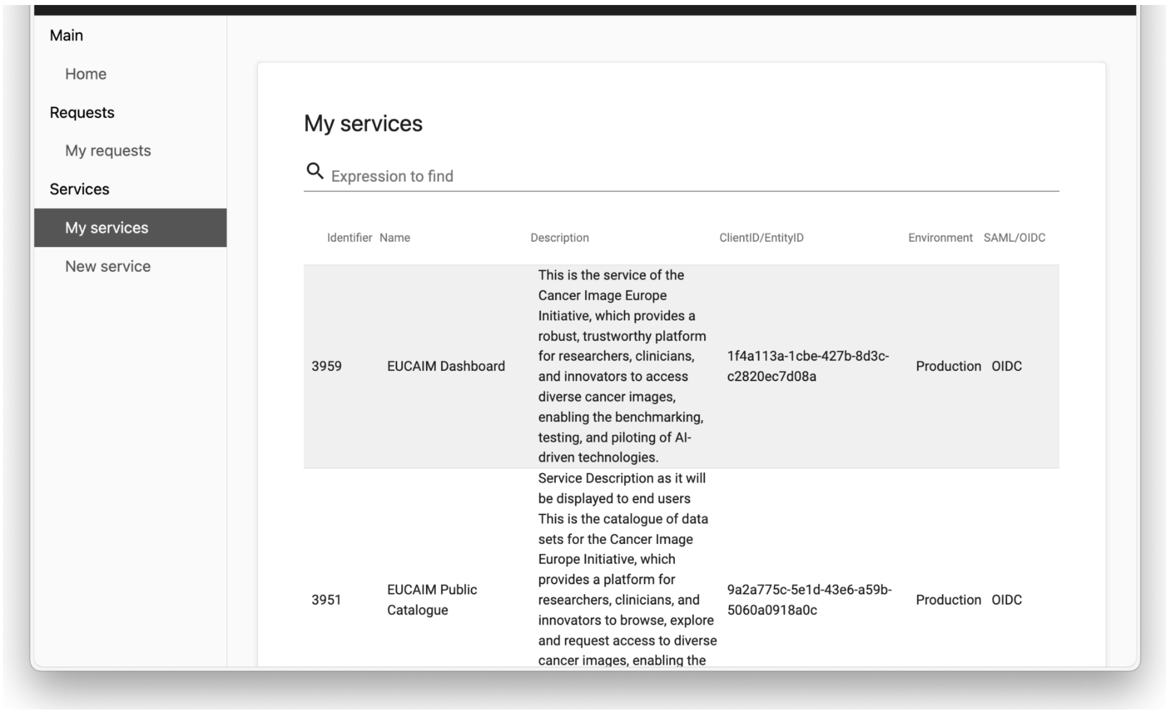
The Dashboard and the Catalogue allows anonymous access. These services provide access to general information, onboarding processes and aggregated data, and for this reason, access requires authentication and authorization.

As shown in Figure 11, each service interacts with the LS-AAI endpoint which delegates the authentication to the institutional IdPs. Along with the acceptance of an authentication request, the LS-AAI service performs the validation of the membership of the users to the EUCAIM VO Group). The LS-AAI service also returns the Entitlements (additional authorisation attributes) of the user including the VO groups membership and roles, which are used in the authorisation process by the Service.

Services in the development platform are authenticated through the same services to facilitate the roll-out of services into production. Figures 12 and 13 shows the management dashboard for the EUCAIM VO Group and the EUCAIM core services.



EUCAIM #23560
UUID: f9f493b4-1145-48f8-9eb1-682b4f51bdc2
Group, Organization: Life Science Community, Description: EUCAIM
This group is a subgroup of communities_and_projects



My services

Identifier	Name	Description	ClientID/EntityID	Environment	SAML/OIDC
3959	EUCAIM Dashboard	This is the service of the Cancer Image Europe Initiative, which provides a robust, trustworthy platform for researchers, clinicians, and innovators to access diverse cancer images, enabling the benchmarking, testing, and piloting of AI-driven technologies.	1f4a113a-1cbe-427b-8d3c-c2820ec7d08a	Production	OIDC
3951	EUCAIM Public Catalogue	Service Description as it will be displayed to end users This is the catalogue of data sets for the Cancer Image Europe Initiative, which provides a platform for researchers, clinicians, and innovators to browse, explore and request access to diverse cancer images, enabling the	9a2a775c-5e1d-43e6-a59b-5060a0918a0c	Production	OIDC

Figures 12 and 13. *Snapshot of the management console of the LS AAI: VO-management panel*[48] *(up) and service management section*[49] *(down)*

[48] EUCAIM VO management: https://perun.aai.lifescience-ri.eu/organizations/3345/groups/23560

[49] EUCAIM Services management page in LS-AAI: https://services.aai.lifescience-ri.eu/spreg/auth/facilities/myServices

4.3 Dashboard

As previously mentioned, Dashboard serves as the main entry point to the EUCAIM environment. Figure 14 illustrates the schema of the Dashboard components, which is comprised of four main components:

- Persistent Volumes (PVs): Two PVs are utilised for the application and database files.
- A MongoDB database operates within a deployment, persisting database files via a Persistent Volume Claim (PVC). It is internally accessible through a service, which is used by the Dashboard application.
- Node.js Server: A Node.js server runs in a deployment, mounting the dashboard application via a PVC and is exposed through the dashb-node-service.
- Ingress Controller: An ingress controller exposes the dashboard service at the root of the dashboard.* DNS.

It is also important to mention that the kube-root-ca.crt ConfigMap is used to store the Kubernetes root Certificate Authority (CA) certificate. This certificate is essential for various internal communications within the Kubernetes cluster. Specifically, it allows Kubernetes components and applications running in the cluster to verify the authenticity of the Kubernetes API server when using internal endpoints such as the internal service IP or Kubernetes service names.

These components collectively ensure the functionality and accessibility of the EUCAIM Dashboard. The templates for a complete dashboard deployment can be found on GitHub[50].

[50] https://github.com/EUCAIM/k8s-deployments/tree/main/dashboard

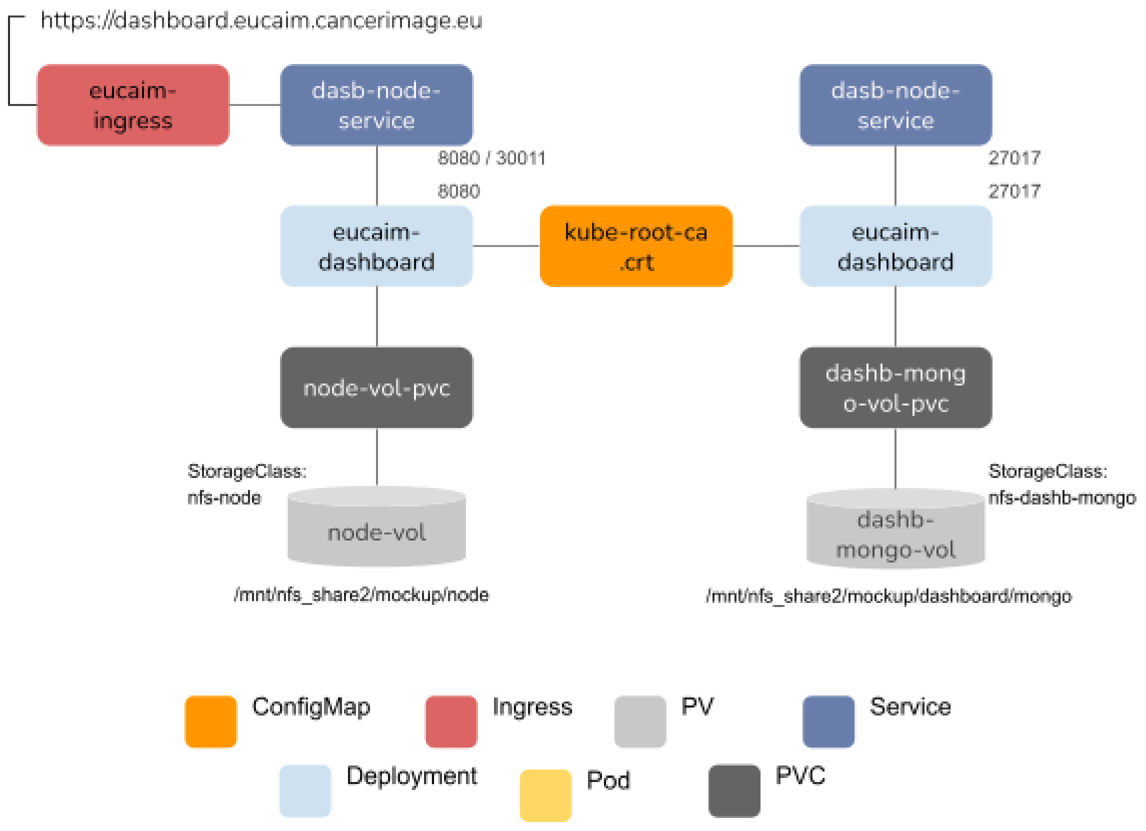


Figure 14. *Dashboard architecture*

The previous components have been listed following the recommended deployment order. To customise the deployment, the following changes should be applied:

- Change the hostname in the root-path-redirect ingress.
- Change the PV NFS endpoints in the node-vol and dashb-mongo-vol PVs
- The configuration of the endpoints should be provided in a file named settings.json and the configuration of the AAI should be changed in the code (client/main.jsx, structure LSConfig). The package should be compiled and uploaded into the Dockerfile of the container images

The provisioning of a certificate for the domain where the service will be exposed is automatically created through an annotation in the ingress controller. This annotation triggers the creation and fetching of a certificate for the domain through Let's encrypt service, which is stored in a secret Kubernetes object.

4.4 Catalogue

The Molgenis catalogue stores the metadata, offering the researchers descriptive information about the available datasets. The manifests that were used for the deployment of the catalogue could be also find in GitHub[51].

The architecture of the solution could be found in figure 15 and figure 16. This deployment consists of four elements:

1. **Molgenis deployment:** This deployment is bounded to two ConfigMaps that configure the service and two Persistent Volumes (PV) where we store logs from the application (audit PV) and the backups and the necessary files for running the application backend (app-data PV).
2. **Frontend deployment:** The frontend deployment runs the molgenis-frontend image. This frontend is accessible via web at the url https://catalogue.eucaim.cancerimage.eu/ thanks to the Ingress attached to it. It also has a persistent volume bounded where are stored the necessary files for running the Molgenis user interface.
3. **Postgres database:** A database, where the catalogue metadata is stored. The Postgres database information is stored in the postgres-vol PV.
4. **Elasticsearch:** Molgenis uses Elasticsearch for indexing the data layer. This deployment also comes with its own persistent volume, allowing elasticsearch to keep the data.

ConfigMaps

As can be seen in Figure 16, the architecture of the federated search contains 5 distinct ConfigMaps. Each of these ConfigMaps has been designed with the following purposes:

- **ConfigMap backend:** Configures an Nginx server to act as a reverse proxy and manage DNS resolutions within the cluster.
- **ConfigMap nginxconf:** Contains the main settings for an Nginx server, including worker processes and logging configurations.
- **ConfigMap tomcat-webxml:** Sets up web application parameters for Tomcat, focusing on file upload configurations and application metadata.
- **ConfigMap tomcat-serverxml:** Provides detailed server settings for Tomcat, focusing on HTTP connector configurations and user authentication.
- **ConfigMap ltsconf:** Manages URL rewrites and redirects in Nginx to optimise content delivery and application functionality.

[51] https://github.com/EUCAIM/k8s-deployments/tree/main/Molgenis

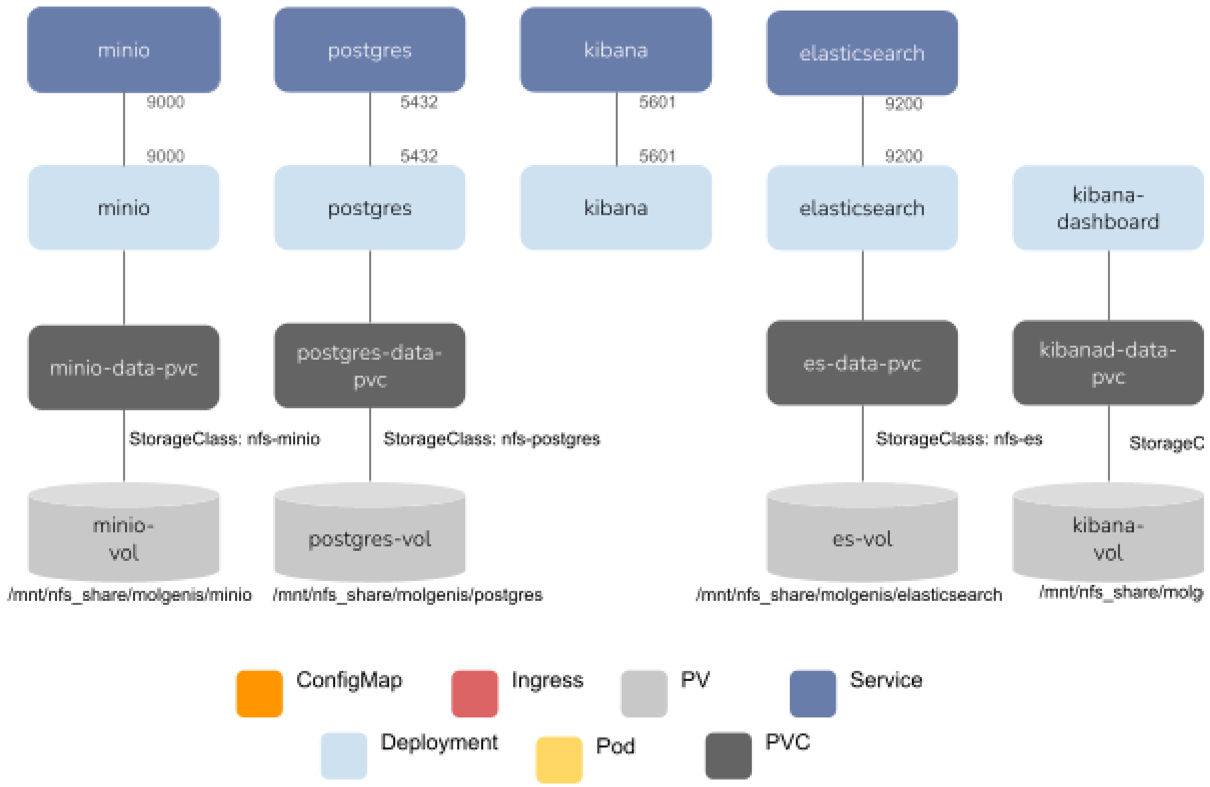


Figure 15. *Deployment schema of the postgres and elasticsearch deployments in the catalogue*

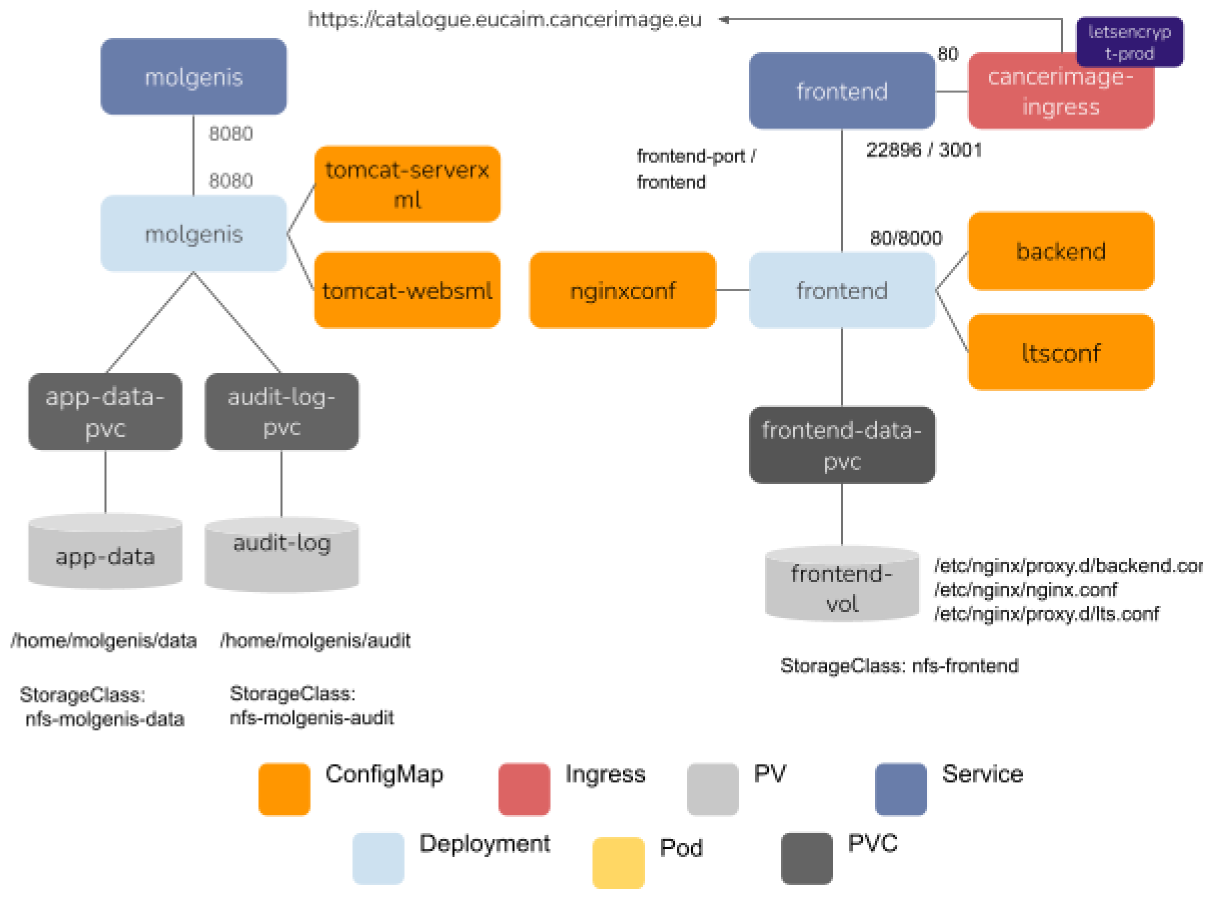


Figure 16. *Deployment schema of the Molgenis backend and frontend deployments.*

Each deployment uses a service to make the services accessible and discoverable. Only the frontend service is made available to outside of the platform through an ingress controller. The catalogue application is available in the repository https://gitlab.com/radiology/infrastructure/projects/catalogue/eucaim-molgenis-app

4.5 Explorer

The explorer, also known as federated search, is responsible for obtaining and displaying the datasets and the number of subjects that fulfil a specific filtering criteria defined by the user. The software is available in https://github.com/samply/beam.

The architecture of the federated search is composed of six different elements that allows it to manage the connections between the different data holders.

1. **Lens:** Lens is responsible for providing the graphical interface to the user, allowing them to interact with the explorer in an efficient way. This deployment is linked to an ingress that makes it accessible at the url: https://explorer.eucaim.cancerimage.eu/

2. **Spot:** Spot is the backend service of the federated search application and is responsible for making the calls to the proxy for retrieving the data that the user specified.

3. **Beam-Proxy:** The Beam-Proxy is responsible for connecting the application service to the broker, thereby allowing the application to access data from other projects.

4. **Broker:** The Broker is in charge of linking all the projects together by managing the requests from the proxies (each project has its own proxy).

5. **Vault:** A Vault meant to store and protect the access to tokens, passwords and certificates. Vault services can be accessed through the Vault API and we can also use the vault for auditing due to it keeps a log of the access to secrets, allowing security administrators to track who accessed which secrets and when. Detailed Explanation in Section 5.1.1 Vault

6. **Oauth 2.0 Proxy:** This Oauth2 proxy acts as a reverse proxy and authentication layer. It allows us to access the application via LifeScience AAI authentication without having to implement a login application.

Figure 17 shows the deployment at the central service and Figure 18 shows the deployment at the data holders, as well as the interaction among the components. As can be seen in Figure 18, each data holder must have a beam proxy service deployed for connecting to the beambroker of the central node, and a Focus service that acts as an intermediary with the particular data service that each data holder has. To facilitate the understandability of both figures, we have not included the pod objects associated with the deployments.

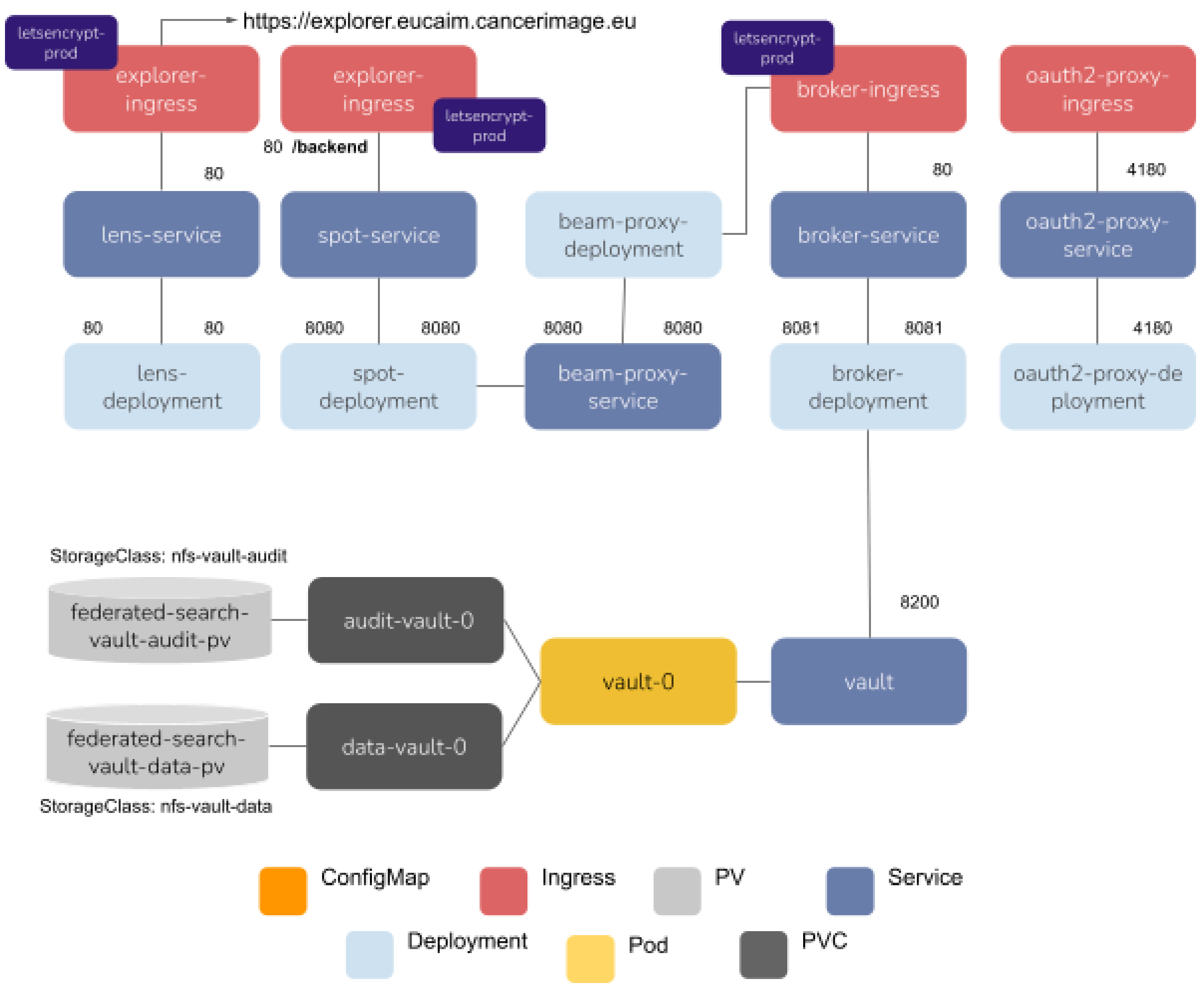


Figure 17. *Deployment schema of the federated search at the core services level*

You can find the manifests and some more information about the deployment of this service at: https://github.com/EUCAIM/k8s-deployments/tree/main/federated-search/eucaim.

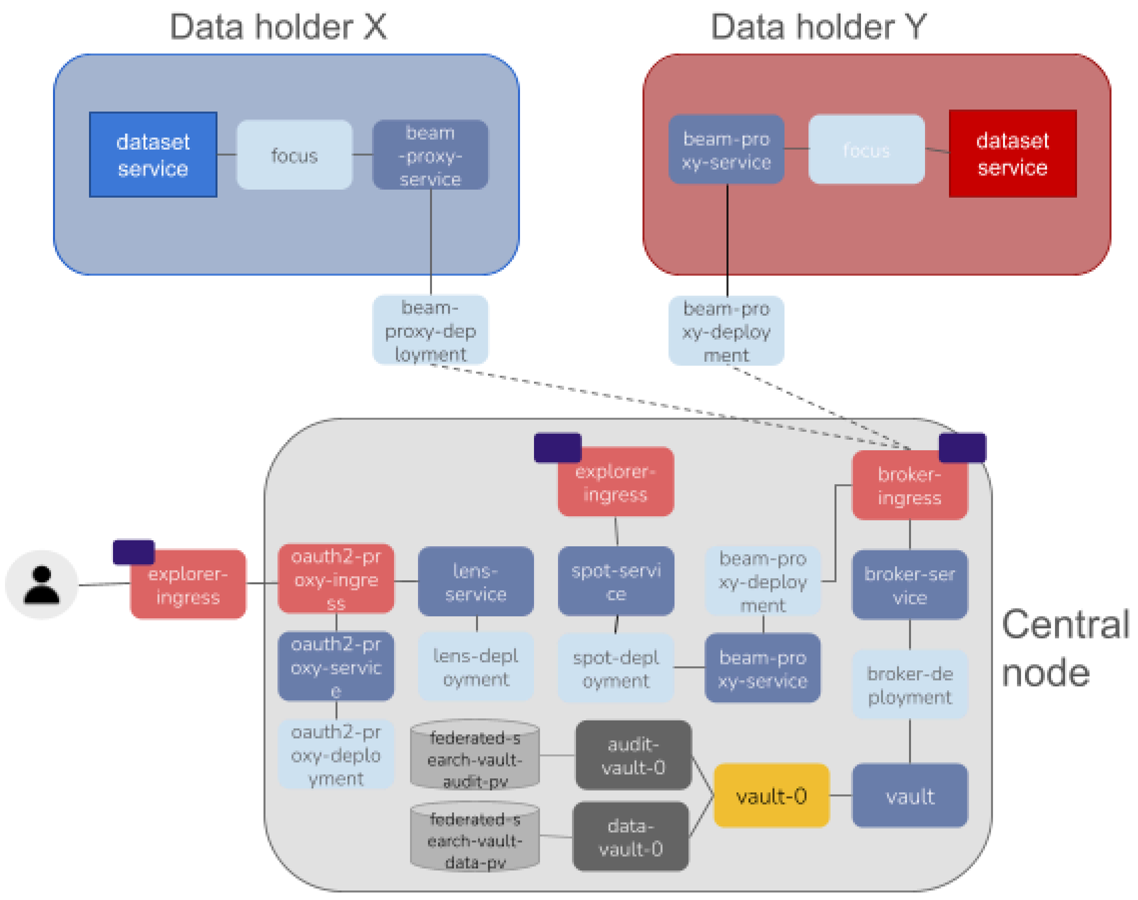


Figure 18. *Deployment schema of the federated search including the interactions with the data holders*

As in the previous core services, a Kubernetes service is created to provide a persistent endpoint for lens, spot, beam-proxy, broker-service and vault. The application does not need persistence except for the case of the access credentials, stored in Vault.

The authentication and authorisation is provided by the OAuth2 proxy ingress service and an OAuth2 configuration in a secret, both available in the GitHub Repository[52]. For the deployment, the following attributes should be updated:

- OAuth2 ingress: the attribute host should include the DNS of the host for discriminating Federated Search requests (explorer.eucaim.cancerimage.eu in the case of EUCAIM production service.
- OAuth2 proxy deployment: The following arguments should be checked and updated:
    - --redirect-url=https://explorer.eucaim.cancerimage.eu/oauth2/callback
    - --oidc-issuer-url=https://proxy.aai.lifescience-ri.eu
    - --scope="openid email profile eduperson_entitlement"
    - --whitelist-domain=explorer.eucaim.cancerimage.eu
- OAuth2 secret: Client ID and Secret access key from the LS-AAI console.

[52] https://github.com/EUCAIM/k8s-deployments/tree/main/federated-search/eucaim

4.6 Negotiator

The negotiator component deployment manifests are available in GitHub[53]. It comprises three main services that implement the UI, the backend and the database. Although the UI is the only service that is directly accessible from users, the backend exposes the API to the web browser application in the /api path. Therefore, the negotiator application has two ingress services that expose the frontend and backend deployments. A third deployment manages a postgres database that is accessible only through the internal network. The persistence of the applications is fully managed through the database, that includes the UI forms. The persistence is provided by a PVC mounting a dedicated NFS PV Volume. Figure 19 describes the names and interactions of the components.

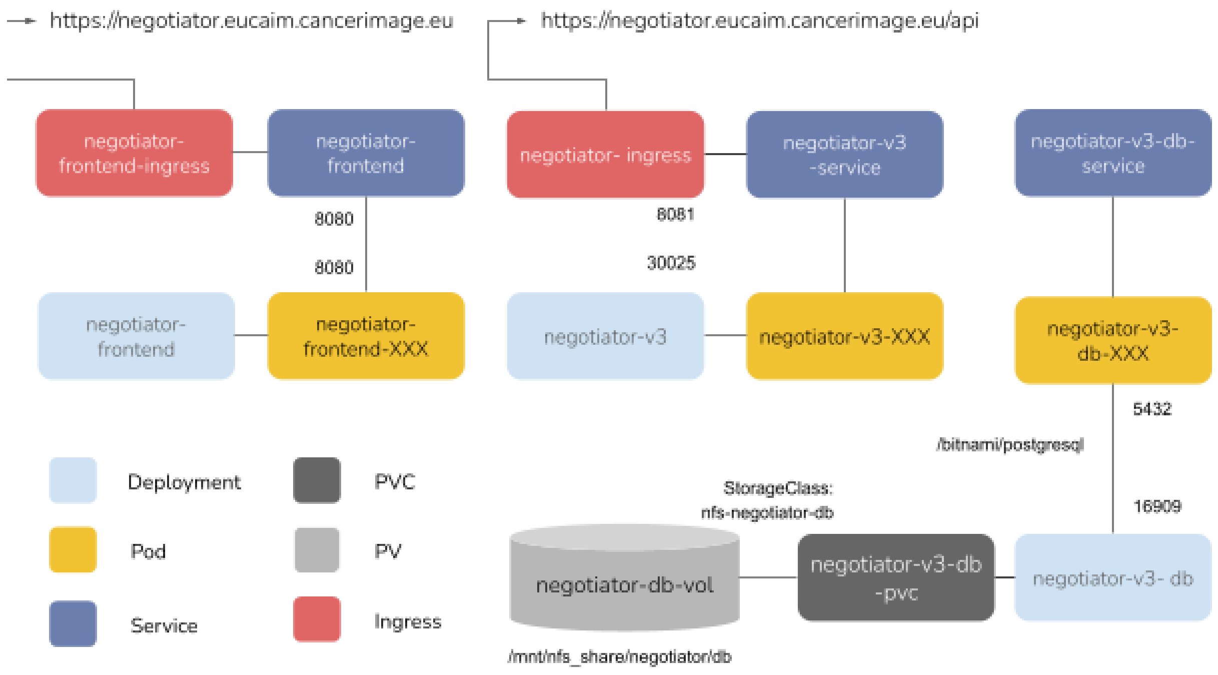


Figure 19. *Schema of Kubernetes manifests for the deployment of the Negotiator*

The deployment should first create the PV and the PVC, and then deploy the database, the backend and the frontend UI.

The customisation of the deployment of the negotiator implies the following steps:

- Update the IP of the NFS service and the location of the NFS folder.
- Update the LS-AAI details in the SPRING_* in the backend and the Molgenis catalogue URL.
- Update the LS-AAI configuration in the frontend (AUTH_URL, CLIENT_ID and REDIRECT_URI).

[53] https://github.com/EUCAIM/k8s-deployments/tree/main/negotiator_v3

- Update the hostname attributes in the Ingress objects.
- The SSL certificate is automatically created by Kubernetes if the Let's encrypt service is installed along with the provisioner. First time the service is deployed a secret with the certificate is created.

Important details to take into account:

- The SPRING_SECURITY_OAUTH2_RESOURCESERVER_JWT_ISSUERURI value should en in a backslash (e.g. "https://login.aai.lifescience-ri.eu/oidc/")
- The SPRING_SECURITY_OAUTH2_RESOURCESERVER_JWT_AUDIENCES value should contain the client ID of the UI rather than the URL (e.g. "a15a95d1-b251-4f12-b608-76ec02c68010").

4.7 Monitoring

The monitoring service provides an overview of the status of the different EUCAIM components by making requests to the associated web services at certain periods of time. In addition to this, the service is also capable of sending notifications to the person in charge of a EUCAIM component when one of the predefined rules is fulfilled.

The service's architecture is composed of 6 components of the technological stack Elasticsearch-Logstash-Kibana (ELK stack), which are deployed in a Kubernetes cluster using the operator pattern, that is responsible for defining Kubernetes software extensions that use custom resources to manage applications and their components.

1. **Elasticsearch:** Search engine that stores the information corresponding to different metrics in indexes and allows access to the data in a very fast way and with easy scaling. In this case, an index has been defined with the name *monitor-alerts*, where a document is written each time an alert occurs in one of the EUCAIM services.

2. **Kibana:** Software that allows the generation of different types of dashboards, making it easier to visualise and interpret data of different types and origins.

3. **Heartbeat:** Functionality that periodically checks the status of a set of predefined services and, based on the response received, is able to determine whether they are available or not.

4. **Logstash:** It is responsible for collecting the metrics of the Elasticsearch *monitor-alerts* index. Then, it filters the data collected and, depending on the service that sends the message, it sends a notification email to the person in charge of that service.

5. **Elastic Agent** and **kube-state-metrics:** These two components work together to collect Kubernetes cluster state metrics and store them in an Elasticsearch index for further analysis.

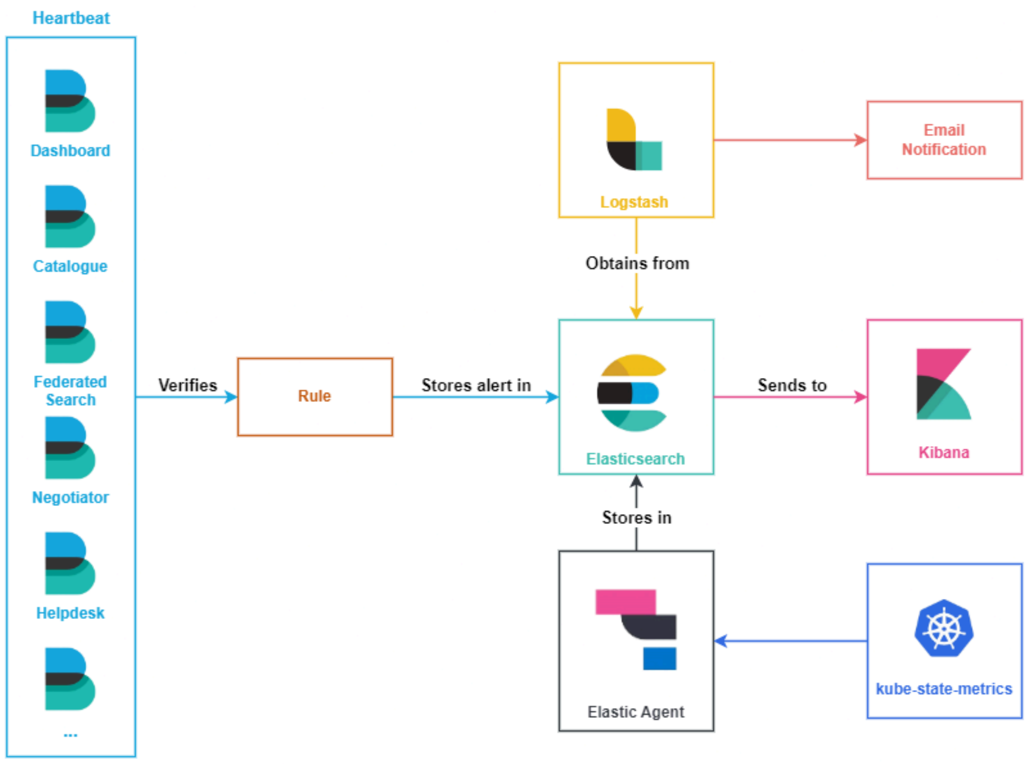


Figure 20. *Elasticsearch working schema*

5 Solution Development

5.1 Environment Configuration

To launch our services, it is necessary to pre-configure an environment that allows their proper functioning. Without going into the low-level configuration details of the environment (Kubernetes installation, OpenStack management, etc.), some of the fundamental configurations required to deploy our services are outlined below.

5.1.1 Vault

In this section, we explain how we configured our Vault service. This configuration may be replicated in other environments.

StorageClass

The first step in configuring our Vault is to create an `nfs-vault` StorageClass that is linked to the automatic provisioner we mentioned earlier.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs-vault
provisioner: fuseim.pri/ifs
reclaimPolicy: Retain
volumeBindingMode: Immediate
```

Create necessary Persistent Volumes Claims

Next, we will proceed with creating the necessary PVCs (specified in the Helm chart mentioned below). By specifying the StorageClass we created in the previous step, these PVCs will create the persistent volumes needed to store our Vault's data.

Deploy HashiCorp Vault

The next step is to execute a Helm chart to deploy and configure Vault within our Kubernetes cluster. The Helm chart and configuration used for this deployment can be found in the EUCAIM GitHub repository.

The next three lines can be used for Vault installing and configuration:

```
helm repo add hashicorp https://helm.releases.hashicorp.com

helm search repo hashicorp/vault

helm install vault hashicorp/vault --values
vault-override-values.yml -n vault
```

This configuration allows us to inject Vault secrets into other pods. Additionally, readiness and liveness probes are performed for health checks, and the storage type to be used is configured.

Vault Configuration

Next, you need to unseal the Vault to manage our secrets. Once unsealed, it will return a key that we have to use to authenticate and authorise access to Vault's functionalities. It is important to note that this key is only generated once, so better store it in a safe place.

Once unsealed, using the script available in: https://github.com/EUCAIM/k8s-deployments/tree/main/federated-search/eucaim/conf-vault, proceed to create a Public Key Infrastructure (PKI), generate intermediate certificates and a root certificate, and create a new role to manage control and access to our Vault. (All these instructions must be executed within the Vault).

Create a certificate for Beam Broker

After this, we will obtain a certificate issued by Vault for our broker. This allows us to secure communications and ensure the authenticity and integrity of the data. With a certificate, broker clients can be assured that they are connecting to the legitimate broker and not a malicious replica.

Sign the Beam Proxy Certificate

Finally, once the certificate for the Beam Broker is created, we need to sign a certificate for our Beam Proxy. This signed CSR becomes a certificate that authenticates the identity of the Beam Proxy to the broker. Without a valid certificate, the broker cannot verify the authenticity of the Beam Proxy. Additionally, the certificate contains a public key used to establish an encrypted connection between the Beam Proxy and the broker. Without this certificate, communications would not be encrypted, exposing the data to potential interception and attacks. This certificate must be included in the Beam Proxy of each Data Host that wishes to be part of the federated infrastructure.

5.1.2 NFS (Network File System)

The storage scheme implemented in our solution can be seen in Figure 21. *NFS solution*. To operationalize this solution, it is necessary to carry out a series of steps on a pre-configured infrastructure to meet certain requirements.

The steps to configure NFS in our cluster are as follows:

1. NFS installation

```
sudo apt update

sudo apt install -y nfs-kernel-server
```

2. Create NFS directory

```
sudo mkdir -p /mnt/nfs_share #Create the directory for NFS share

sudo chown -R nobody:nogroup /mnt/nfs_share/ #Set ownership of the directory

sudo chmod 777 /mnt/nfs_share/
```

3. Edit /etc/exports file to define the directories to be shared via NFS and specify the permissions and access control for each shared directory.

```
/pv *.localdomain(rw,async,no_root_squash,no_subtree_check,insecure)

/mnt/nfs_share *.localdomain(rw,sync,no_subtree_check,no_root_squash)
```

4. Restart the NFS server

```
sudo systemctl restart nfs-kernel-server
```

5. If you want to automate the creation and management of Persistent Volumes (PVs) you can install an external provisioner:

```
$ helm repo add nfs-subdir-external-provisioner https://kubernetes-sigs.github.io/nfs-subdir-external-provisioner/

$ helm install nfs-subdir-external-provisioner nfs-subdir-external-provisioner/nfs-subdir-external-provisioner \

    --set nfs.server=x.x.x.x \

    --set nfs.path=/exported/path
```

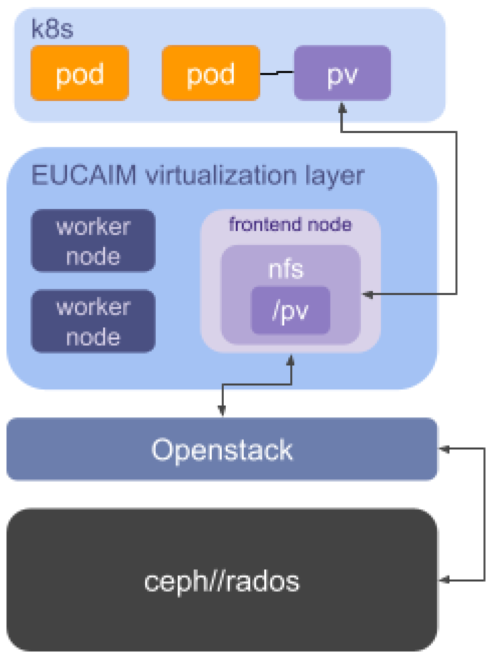


Figure 21. *NFS solution*

5.2 Kubernetes manifests

The Kubernetes manifests presented below are available in the EUCAIM project's GitHub repository[54]. The following sections provide a brief description of their content and highlight some key configuration aspects to consider for future deployments.

5.2.1 Molgenis

Within the Molgenis GitHub repository, you will find the YAML files for deploying EUCAIM Molgenis in Kubernetes. These deployment files are customised for UPV's deployment, and they are all necessary for the correct functioning of this version of Molgenis.

The manifests found within the Molgenis repository are as follows:

backend.yaml

[54] https://github.com/EUCAIM/k8s-deployments

Composed by five ConfigMaps, it configures an Nginx server as a reverse proxy and manage DNS resolutions (backend), define main settings for Nginx (nginxconf), set up web application parameters for Tomcat (tomcat-webxml), provide detailed server settings for Tomcat (tomcat-serverxml), and manage URL rewrites and redirects in Nginx to optimize content delivery (ltsconf).

deployment-molgenis.yaml

This Kubernetes manifest sets up a federated search architecture by deploying a total of eight deployments. The deployments include PostgreSQL, Elasticsearch, Kibana, Kibana Dashboard, OpenCPU, MinIO, Molgenis, and the Molgenis front-end. Each component needs to be configured with the proper environment variables, persistent storage, and exposed services. Additionally, it includes an Ingress configuration for front-end access through a specified domain, utilising Nginx and Let's Encrypt for TLS certificates.

deployment-nfspvc.yaml

In this YAML document, the necessary persistent volume claims for our deployment are declared. For more details, see 4.4 Catalogue.

deployment-nfsvols-template.yaml

In the `deployment-nfsvols-template.yaml` document, the persistent volumes to which the previously mentioned persistent volume claims are bound are declared. It is important to note that if you want to perform deployments using this template, you will need to configure both the NFS address and the mount path to adapt them to your specific environment.

5.2.2 Dashboard

dashb-mongo-deployment.yaml

This Kubernetes manifest deploys a MongoDB instance for the dashboard namespace. It includes a single replica running the eucaim.cancerimage.eu:10443/library/mongodb image, exposing port 27017. The deployment uses a persistent volume claim for data storage and mounts necessary Kubernetes API access secrets.

eucaim-dashboard-deployment.yaml

This Kubernetes manifest contains the deployment of the eucaim-dashboard application using a Node.js image. It configures the container to run a script (/data/runApp.sh), exposes port 8080, and mounts a persistent volume claim (node-vol-pvc) for data storage.

It's important to note that it is necessary to previously include all the data required by the dashboard application in the persistent volumes to allow this application to run.

dash_and_mongo_pvc_and_vols-template.yaml

This YAML defines the necessary persistent volume claims and the persistent volumes bound to them that are required for the previously mentioned deployments to function correctly.

dashboard-service.yaml

Necessary service to expose the pods functionalities.

dashboard-ingress.yaml

Ingress to allow access to the dashboard application via HTTP.

root-path-redirect-ingress.yaml

This ingress is in charge of redirecting all HTTP traffic from eucaim.cancerimage.eu to https://dashboard.eucaim.cancerimage.eu permanently.

5.2.3 Federated-Search

beam-proxy.yml

This manifest contains the deployment of the image: `samply/beam-proxy:develop` as well as the creation of the secret for storing the private key of the beam proxy, which is necessary for its connection to the broker and for the proper functioning of the explorer service.

Additionally, this manifest also includes a service for exposing the deployment services to other services. It is important to adapt the configuration of this YAML to the needs of each deployment, particularly the environment variables: `PROXY_ID and BROKER_URL`, which may vary depending on the configured environment.

broker.yaml

This file contains the ingress that makes the broker service accessible both externally and internally to manage requests from the data holders.

Additionally, it includes the deployment of the Beam Broker using the image: `samply/beam-broker:main`, a service (`broker-service`), and a secret (`pki-secret`).

It is important that the necessary secrets (`pki-secret`: Vault token; `root-crt-pem`: platform certificate generated in Vault; `broker-priv-pem`: broker's private key) are correctly generated to ensure proper connection.

common.yaml

This file contains the secrets that store the access token for Spot and the root certificate used to manage the other certificates.

lens.yaml

This manifest includes an ingress to allow users to connect to the Explorer (remember that the lens service is the frontend of the Explorer), a service, and a deployment of the Lens image: `docker.verbis.dkfz.de/eucaim/lens:eucaim`.

oauth2-proxy.yml

This manifest deploys an ingress to redirect authentication via OAuth, a service to access the deployment, and an OAuth proxy with the image: `quay.io/oauth2-proxy/oauth2-proxy:latest`.

For the configuration of the proxy, it is crucial to properly configure the environment variables and include the necessary secrets.

spot.yml

This manifest is responsible for deploying the ingress and the Spot deployment (`docker.verbis.dkfz.de/eucaim/spot:eucaim`), the backend of the Explorer.

vault-override-values.yml

This YAML file is essential for the correct installation of Vault.

(`helm install vault hashicorp/vault --values vault-override-values.yml -n vault`).

5.2.4 Negotiator

negotiator-db-deployment.yaml

Deployment of the SQL database (`bitnami/postgresql:14`) that will serve as storage for our Negotiator.

negotiator-deployment.yaml

Deployment of the Negotiator. It is crucial for the user to enter the corresponding URLs for their deployment and provide the necessary passwords and tokens. Please check that the image of the Negotiator is up to date.

negotiator-frontend.yaml

Manifest that sets up the user interface through which the user will access the Negotiator service. It is also important to enter a `CLIENT_ID` that corresponds to what the user has in their LSAAI account.

negotiator-ingress.yaml

Ingress through which the negotiator's services will be exposed. In our case, it is available at: https://negotiator.eucaim.cancerimage.eu/.

5.4 Automation in a CI/CD environment

CI/CD stands for Continuous Integration and Continuous Delivery or Deployment, aimed at improving and accelerating the software development lifecycle.

Continuous Integration (CI) is a practice of automatically and periodically incorporating code changes into a shared source code repository. Continuous Delivery and Continuous Deployment (CD) are a two-part process in which code changes are integrated, tested, and delivered. While Continuous Delivery does not automatically deploy changes to production, Continuous Deployment does automatically release updates to this environment.

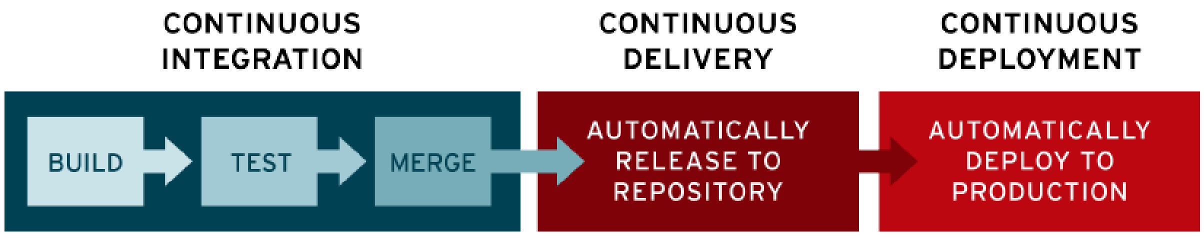


Figure 22. *CI/CD Flow*

CI/CD prevents errors and code failures in companies without disrupting the ongoing cycle of software development and updates. Its features can help reduce complexity, increase efficiency, and optimise workflows as applications grow.

In this project, we are using GitHub Actions. GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline. This has enabled us to test future deployments in our cluster in a safe environment and to prevent future coding mistakes or inconsistencies. By doing this, we ensure significant time savings in error corrections and repository updates, and we verify that we are using good programming practices in our manifests by checking them every time we push them into the repository. The architecture of our GitHub Actions integration would consist of a series of different services that will work together and allow us to execute our GitHub code on our own cluster. This architecture can be seen in Figure 23. [25][26]

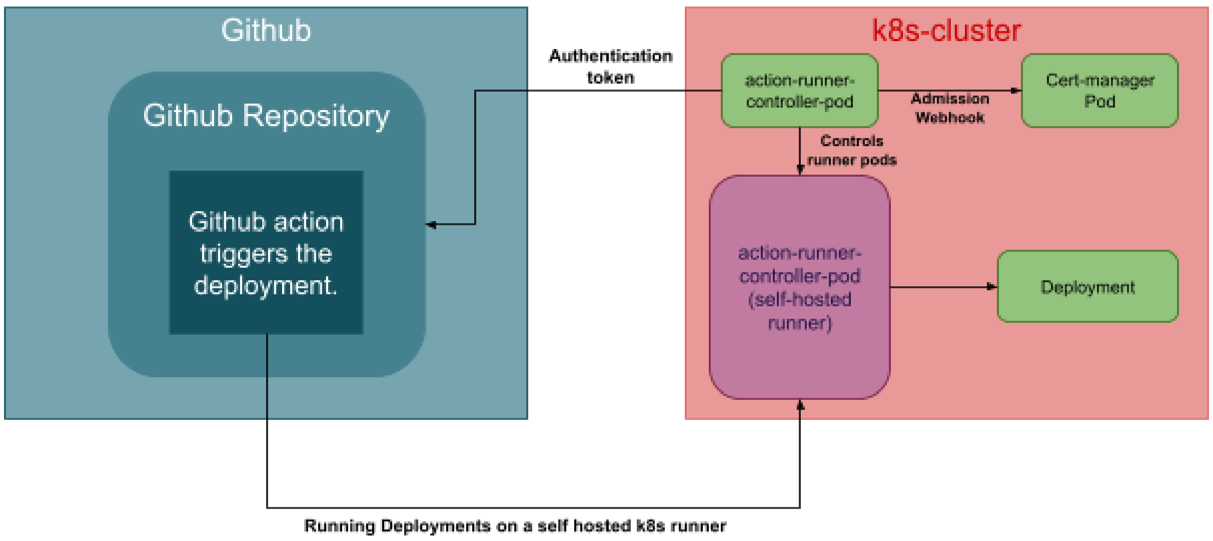

Github
Github Repository
Github action triggers the deployment.
Authentication token
k8s-cluster
action-runner-controller-pod
Admission Webhook
Cert-manager Pod
Controls runner pods
action-runner-controller-pod (self-hosted runner)
Deployment
Running Deployments on a self hosted k8s runner

Figure 23. *GitHub Actions Self Hosted Runner Schema*

5.4.1 Self-hosted GitHub Actions configuration

To set up a CI/CD environment that allows us to launch processes on our own cluster, the following configuration must be performed:

1. Install cert-manager

By default, actions-runner-controller uses cert-manager for certificate management of admission webhook, so we have to make sure cert-manager is installed on Kubernetes before we install actions-runner-controller.

You can install cert-manager with the next Helm CLI command:

```
helm install \
  cert-manager jetstack/cert-manager \
  --namespace cert-manager \
  --create-namespace \
  --version v1.15.0 \
  --set crds.enabled=true
```

2. Setting Up Authentication for Hosted Runners

There are two ways for actions-runner-controller to authenticate with the GitHub API (only 1 can be configured at a time, however):

- Using a GitHub App (not supported for enterprise-level runners due to lack of support from GitHub.)
- Using a PAT (personal access token)

To authenticate an action-runner-controller with the GitHub API, we will use a GitHub App with the action-runner-controller to register our self-hosted runner.

To do this, you can enter your Github and then go to Account > Settings > Developers settings > GitHub Apps > New GitHub App. Then you can configure your new GitHub App with the necessary parameters.

After creating the new GitHub App, you have to copy the generated ids to the bash and run the below commands to create a Kubernetes secret, this secret will be used by action-runner-controller deployment.

```
kubectl create ns actions-runner-system
kubectl create secret generic controller-manager -n actions \
 - from-literal=github_app_id=YOUR_GITHUB_APP_ID \
 - from-literal=github_app_installation_id=YOUR_INSTALLATION_ID \
 - from-literal=github_app_private_key=YOUR_PRIVATE_KEY
```

3. Install action runner controller (ARC) on the Kubernetes cluster

ARC is a Kubernetes controller that creates self-hosted runners on your Kubernetes cluster.
To instal the Action Runner Controller you can use the Helm following CLI command:

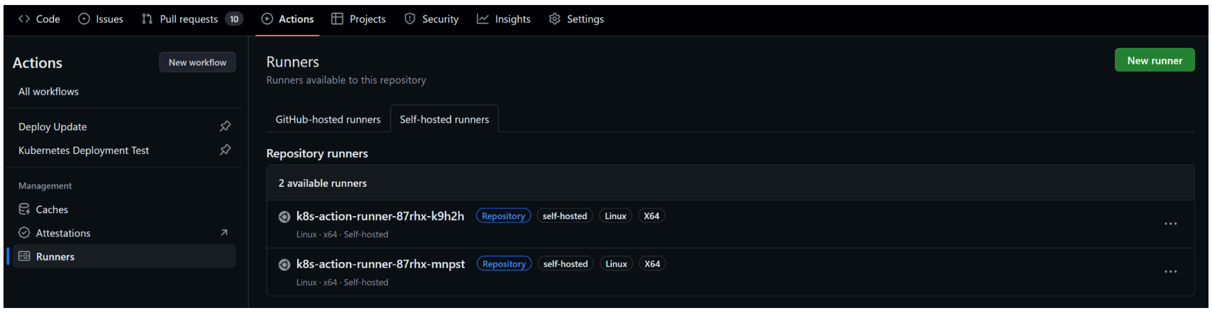
```
helm repo add actions-runner-controller
https://actions-runner-controller.github.io/actions-runner-controlle
r
helm repo update
helm upgrade --install --namespace actions-runner-system
--create-namespace --wait actions-runner-controller
actions-runner-controller/actions-runner-controller --set
syncPeriod=1m
```

4. Create a Repository Runner

This repository runner will be a RunnerDeployment Kubernetes object, which will create a self-hosted runner for the desired GitHub repository.
The RunnerDeployment manifest file would look like the next template.

```
apiVersion: actions.summerwind.dev/v1alpha1
kind: RunnerDeployment
metadata:
 name: k8s-action-runner
 namespace: actions-runner-system
spec:
 replicas: 2
 template:
   spec:
     repository: yourGitHub/repository
```

If everything has worked correctly, we should see the two action runners running in our cluster, and we should also be able to see them in the Actions section of the GitHub website.



Runners
Runners available to this repository

Repository runners

2 available runners

k8s-action-runner-87rhx-k9h2h Repository self-hosted Linux X64
Linux · x64 · Self-hosted

k8s-action-runner-87rhx-mnpst Repository self-hosted Linux X64
Linux · x64 · Self-hosted

Figure 24. *Two runners running*

Once this configuration is complete, our GitHub repository will be integrated with our cluster, allowing us to execute workflows from the GitHub website.

5. Create a workflow

To create a workflow in our GitHub repository, the first thing we need to do is create the `.github/workflows/` folder in the repository where we want to execute the workflow.

Once this folder is created, we can create our first workflow. In our case, this is what the workflow we used to automate the deployment and testing of Kubernetes manifests in our cluster looks like:

```
name: Object Update

on:
  push:
    branches: [main]
    paths:
      - 'k8s/**/*.yaml'
  pull_request:
    branches: [main]
    paths:
      - 'k8s/**/*.yaml'

jobs:
  deploy:
    name: Deploy to Kubernetes
    runs-on: self-hosted

    steps:
    - name: Checkout
      uses: actions/checkout@v2
    - name: Kubeval Validation
      uses: instrumenta/kubeval-action@master
      with:
        files: 'k8s/'

    - name: Count and Print Commits for 'k8s' Folder
      run: |
        COMMIT_COUNT=$(git log --oneline -- k8s/ | wc -l)
        echo "Number of commits affecting 'k8s/' folder: $COMMIT_COUNT"
      shell: bash

    - name: Install kubectl
      run: |
        curl -LO
"https://storage.googleapis.com/kubernetes-release/release/`curl -s
https://storage.googleapis.com/kubernetes-release/release/stable.txt`/bin/l
inux/amd64/kubectl"
        chmod +x ./kubectl
        sudo mv ./kubectl /usr/local/bin/kubectl


    - name: Kubectl Syntax Check
      run: |
        find k8s/ -type f -name '*.yaml' -exec kubectl apply
--dry-run=client -f {} \;


    - name: Apply Kubernetes Manifests
      if: ${{ github.event_name == 'push' && github.ref ==
'refs/heads/main' }}
      run: |
        echo "Applying Kubernetes manifests from pushed files..."
        find k8s/ -type f -name '*.yaml' -exec kubectl apply -f {} \;
```

If we break down each part of the file, we get the following sections:

Workflow Triggers:

- **on: push:** The workflow is triggered when a push is made to the **main** branch, but only if the changes include files within the path **k8s/**/*.yaml**. This means that any changes to YAML files within the **k8s** folder (and its subfolders) will activate this workflow.
- **on: pull_request:** Similarly, the workflow is triggered for pull requests to the **main** branch that affect the same paths.

Jobs

Our workflow consists of a single job that contains the following steps to execute:

Checkout: Uses the `actions/checkout@v2` action to obtain the repository code on the runner.

Kubeval Validation: Uses the `instrumenta/kubeval-action@master` action to validate the Kubernetes YAML files found in the `k8s/` folder. This helps ensure that the files are syntactically correct and valid according to Kubernetes specifications.

Count and Print Commits for 'k8s' Folder: Runs a script that counts and displays the number of commits affecting the `k8s/` folder. This is useful for tracking changes.

Install kubectl: Instals kubectl, a command-line tool for interacting with Kubernetes clusters, on the runner. This is necessary for executing Kubernetes commands later on.

Kubectl Syntax Check: Alternatively to using Kubeval, we can use `kubectl apply --dry-run=client` to perform a syntax check on the Kubernetes YAML files. This ensures that the files are applicable in Kubernetes without actually applying the changes.

Apply Kubernetes Manifests: This step runs **only if the event was a push to the main branch**. This ensures that changes are applied only when they are definitely accepted in the main branch via push, and not during pull requests or in other branches. It then executes `kubectl apply` to apply the changes specified in the YAML files to Kubernetes, thereby updating the cluster state as necessary.

Once our workflow is created, we just need to create a manifest in the indicated path and execute a pull request. This will trigger our workflow and return a result. If the workflow fails, we can check the logs from the GitHub Actions tab. If the workflow runs successfully, we will be able to see the deployment result in our Kubernetes cluster.

6. Testing

6.1 Load Testing of Services

To verify the proper functioning and load tolerance of our services under high load levels, we have conducted various load tests using the Apache JMeter tool[55].

JMeter is a free software tool that can be used to perform load and performance testing on web applications and services. Additionally, it allows for simulating multiple users and analysing response times and performance under different conditions.

In our case, we have configured various tests in JMeter to check the response times of our different endpoints:

- Dashboard: https://dashboard.eucaim.cancerimage.eu/

[55] https://jmeter.apache.org/index.html

- Explorer: https://explorer.eucaim.cancerimage.eu/
- Catalogue: https://catalogue.eucaim.cancerimage.eu/

For each of these endpoints, a test has been designed to simulate access by 5000 users spread over a period of 30 seconds. These tests were accompanied by monitoring the load handled by each of the pods serving those services.

In parallel to these tests, additional tests have been conducted using the load testing tool k6[56]. This tool has been particularly useful for load testing the Negotiator, as it was really challenging to bypass the authentication proxy from JMeter, so it was decided to perform the tests from within the cluster.

To achieve this, a Dockerfile was created and published on Docker Hub[57] containing the grafana/k6 image[58], a test script defining the load test, and the command that will be executed when the container starts.

```
FROM grafana/k6


COPY load-test.js /load-test.js


ENTRYPOINT ["k6", "run", "/load-test.js"]
```

Dockerfile

```
import http from 'k6/http';
import { sleep, check } from 'k6';


export let options = {
  stages: [
```

56 https://k6.io/

57 https://hub.docker.com/

58 https://hub.docker.com/r/grafana/k6

```
    { duration: '30s', target: 5000 }, // Ramp-up to 5000 users over 30
seconds
    { duration: '1m', target: 20 },  // Stay at 20 users for 1 minute
    { duration: '30s', target: 0 },  // Ramp-down to 0 users over 30
seconds
  ],
};

export default function () {
  let res = http.get('http://<yout-service>:8080');
  check(res, { 'status was 200': (r) => r.status == 200 });
  sleep(1);
}
```

K6 Script

This k6 image that we uploaded to Docker Hub is used as the base in the Kubernetes job of our cluster to carry out a load test, from which we can also obtain graphical results thanks to the use of Prometheus and Grafana.

The results of these load tests are shown below:

Dashboard Load Test

For the dashboard load tests, we used both of the mentioned alternatives. First, we executed the Kubernetes job with k6 and 5000 simulated users making simple requests to observe the deployment's performance under such user load and requests. Secondly, we conducted a test for 7000 users over a 30-second period using JMeter, and the results were almost identical, as can be seen in the peaks of the graph.

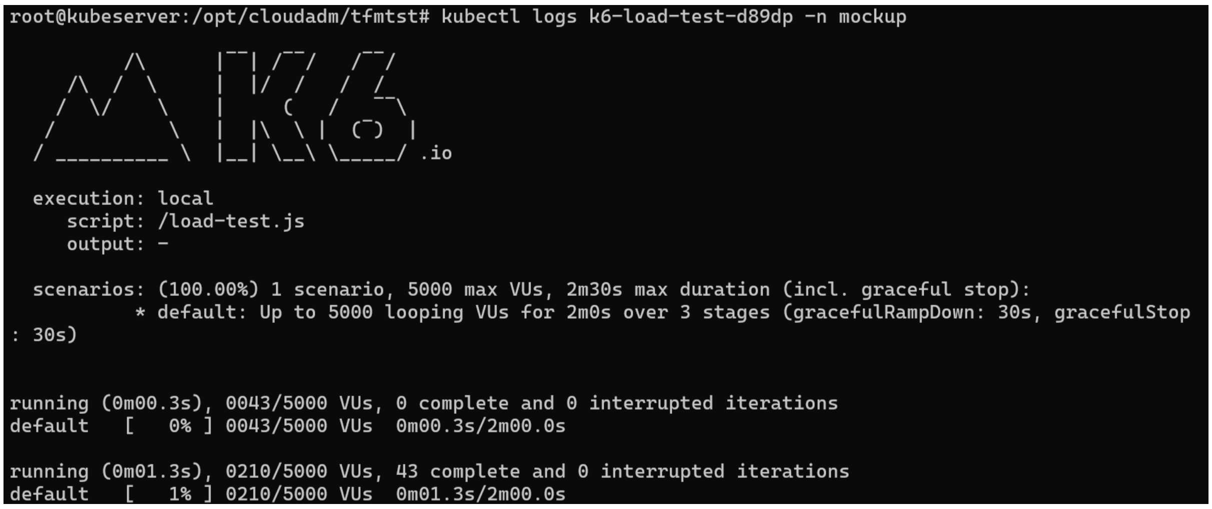
While these load tests do not guarantee the application's correct response in a real environment (with a wide variety of requests of different loads and involving other system elements), they do provide an indicative idea of the load capacity our application can handle.

For example, in the case of the Dashboard and in the other cases presented below, it can be seen that the system easily supports what would correspond to a fairly high load volume for an application of these characteristics.

The following images correspond to:

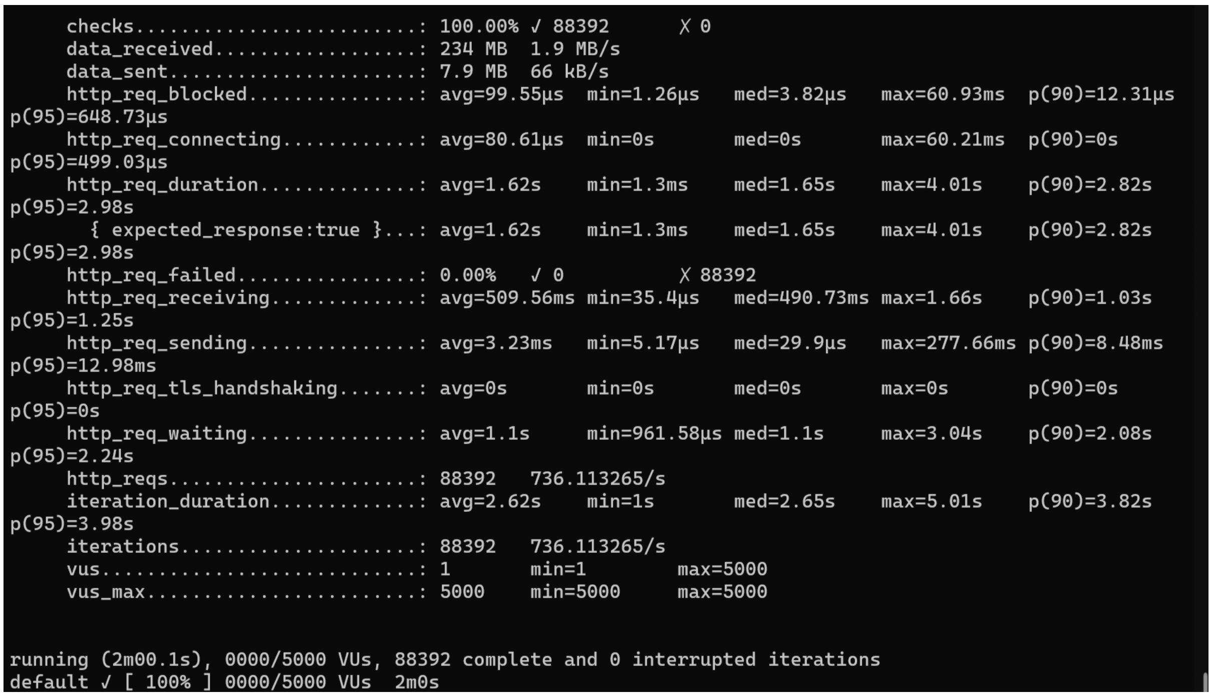
- Figures 25 and 26: K6 load test
- Figure 27: Load Peaks as Results of JMETER Tests (14:00) y K6 (~17:30)

- Figures 28 and 29: JMeter Load Test Results for Dashboard



```
root@kubeserver:/opt/cloudadm/tfmtst# kubectl logs k6-load-test-d89dp -n mockup

          /\      |‾‾| /‾‾/   /‾‾/
     /\  /  \     |  |/  /   /  /
    /  \/    \    |     (   /   ‾‾\
   /          \   |  |\  \ |  (‾)  |
  / __________ \  |__| \__\ \_____/ .io

  execution: local
     script: /load-test.js
     output: -

  scenarios: (100.00%) 1 scenario, 5000 max VUs, 2m30s max duration (incl. graceful stop):
           * default: Up to 5000 looping VUs for 2m0s over 3 stages (gracefulRampDown: 30s, gracefulStop
: 30s)


running (0m00.3s), 0043/5000 VUs, 0 complete and 0 interrupted iterations
default   [   0% ] 0043/5000 VUs  0m00.3s/2m00.0s

running (0m01.3s), 0210/5000 VUs, 43 complete and 0 interrupted iterations
default   [   1% ] 0210/5000 VUs  0m01.3s/2m00.0s
```

Figure 25. *K6 start*



```
     checks.........................: 100.00% ✓ 88392      ✗ 0
     data_received..................: 234 MB  1.9 MB/s
     data_sent......................: 7.9 MB  66 kB/s
     http_req_blocked...............: avg=99.55µs  min=1.26µs   med=3.82µs   max=60.93ms  p(90)=12.31µs
p(95)=648.73µs
     http_req_connecting............: avg=80.61µs  min=0s       med=0s       max=60.21ms  p(90)=0s
p(95)=499.03µs
     http_req_duration..............: avg=1.62s    min=1.3ms    med=1.65s    max=4.01s    p(90)=2.82s
p(95)=2.98s
       { expected_response:true }...: avg=1.62s    min=1.3ms    med=1.65s    max=4.01s    p(90)=2.82s
p(95)=2.98s
     http_req_failed................: 0.00%   ✓ 0          ✗ 88392
     http_req_receiving.............: avg=509.56ms min=35.4µs   med=490.73ms max=1.66s    p(90)=1.03s
p(95)=1.25s
     http_req_sending...............: avg=3.23ms   min=5.17µs   med=29.9µs   max=277.66ms p(90)=8.48ms
p(95)=12.98ms
     http_req_tls_handshaking.......: avg=0s       min=0s       med=0s       max=0s       p(90)=0s
p(95)=0s
     http_req_waiting...............: avg=1.1s     min=961.58µs med=1.1s     max=3.04s    p(90)=2.08s
p(95)=2.24s
     http_reqs......................: 88392   736.113265/s
     iteration_duration.............: avg=2.62s    min=1s       med=2.65s    max=5.01s    p(90)=3.82s
p(95)=3.98s
     iterations.....................: 88392   736.113265/s
     vus............................: 1       min=1        max=5000
     vus_max........................: 5000    min=5000     max=5000


running (2m00.1s), 0000/5000 VUs, 88392 complete and 0 interrupted iterations
default ✓ [ 100% ] 0000/5000 VUs  2m0s
```

Figure 26. *K6 Dashboard test results*

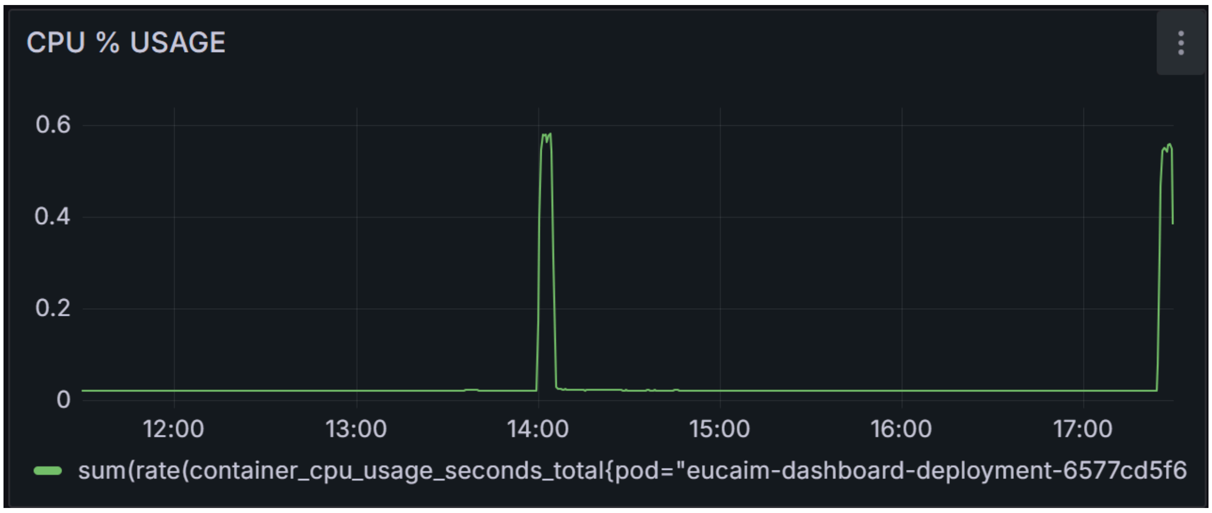
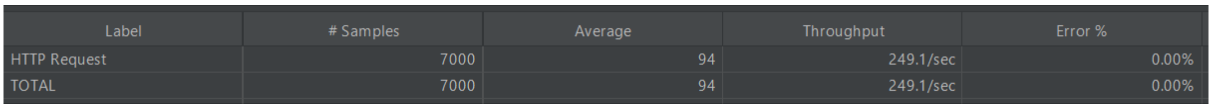
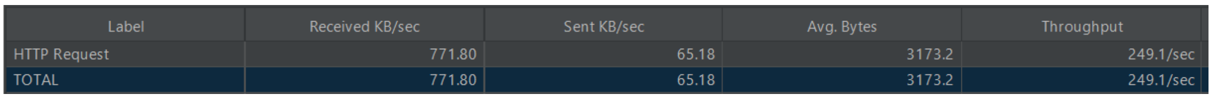


Figure 27. *Grafana dashboard graphics*



Label	# Samples	Average	Throughput	Error %
HTTP Request	7000	94	249.1/sec	0.00%
TOTAL	7000	94	249.1/sec	0.00%

Figure 28. *Jmeter summary I Dashboard*

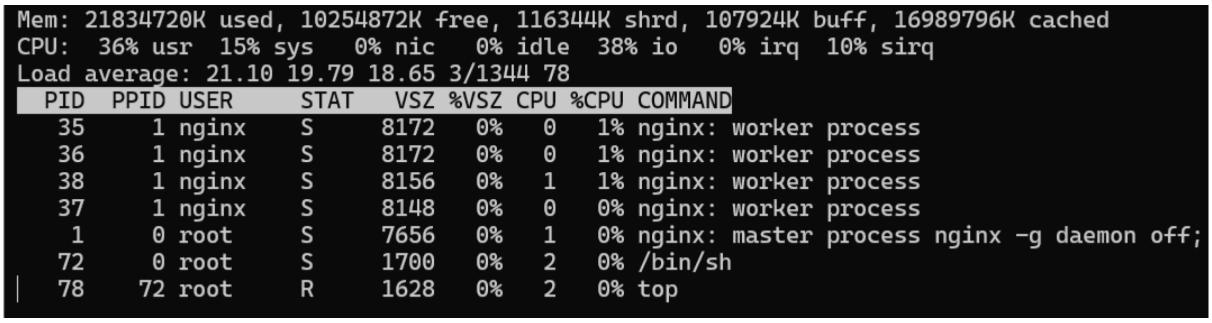


Label	Received KB/sec	Sent KB/sec	Avg. Bytes	Throughput
HTTP Request	771.80	65.18	3173.2	249.1/sec
TOTAL	771.80	65.18	3173.2	249.1/sec

Figure 29. *Jmeter summary II Dashboard*

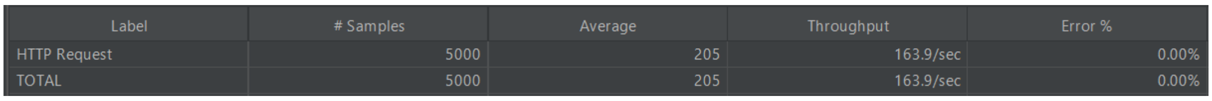
Explorer

For the Explorer case, we made requests from JMeter simulating 5000 users over a period of 20 seconds, and as can be seen (by observing the load from within the pod itself), the load received by the system barely reached 1% of the total capacity.



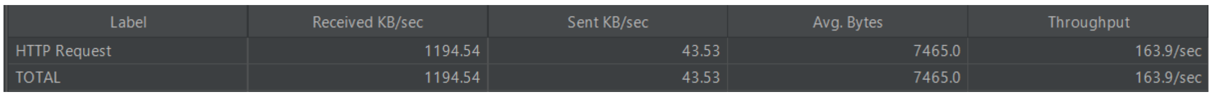
```
Mem: 21834720K used, 10254872K free, 116344K shrd, 107924K buff, 16989796K cached
CPU:  36% usr  15% sys   0% nic   0% idle  38% io   0% irq  10% sirq
Load average: 21.10 19.79 18.65 3/1344 78
  PID  PPID USER     STAT   VSZ %VSZ CPU %CPU COMMAND
   35     1 nginx    S     8172   0%   0   1% nginx: worker process
   36     1 nginx    S     8172   0%   0   1% nginx: worker process
   38     1 nginx    S     8156   0%   1   1% nginx: worker process
   37     1 nginx    S     8148   0%   0   0% nginx: worker process
    1     0 root     S     7656   0%   1   0% nginx: master process nginx -g daemon off;
   72     0 root     S     1700   0%   2   0% /bin/sh
   78    72 root     R     1628   0%   2   0% top
```

Figure 30. *Top Command from Explorer pod*



Label	# Samples	Average	Throughput	Error %
HTTP Request	5000	205	163.9/sec	0.00%
TOTAL	5000	205	163.9/sec	0.00%

Figure 31. *Jmeter Summary I Explorer*



Label	Received KB/sec	Sent KB/sec	Avg. Bytes	Throughput
HTTP Request	1194.54	43.53	7465.0	163.9/sec
TOTAL	1194.54	43.53	7465.0	163.9/sec

Figure 32. *Jmeter Summary II Explorer*

Catalogue

For the Catalogue, we repeated the same tests as for the Explorer (simulating requests from 5000 users over a period of 20 seconds), and the results are practically identical, as can be seen in the load graph in Figure 33 and in the results of the tables generated by JMeter.

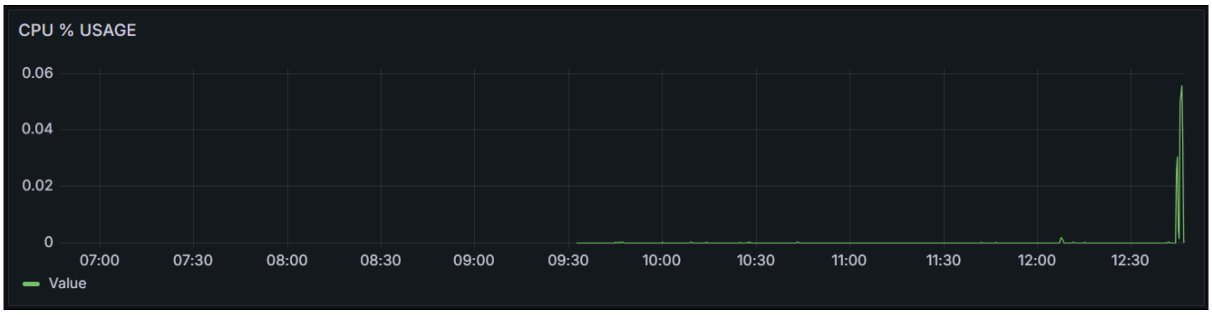
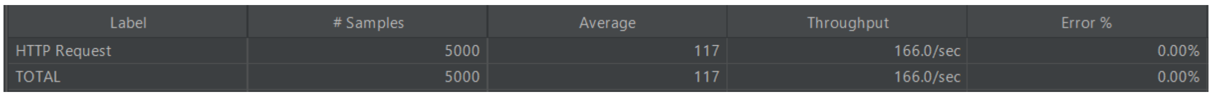
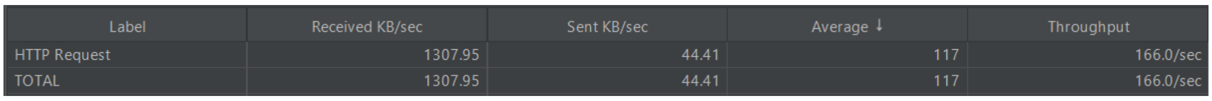


Figure 33. *Grafana Catalogue Graphics*



Label	# Samples	Average	Throughput	Error %
HTTP Request	5000	117	166.0/sec	0.00%
TOTAL	5000	117	166.0/sec	0.00%

Figure 34. *Jmeter Summary I Catalogue*

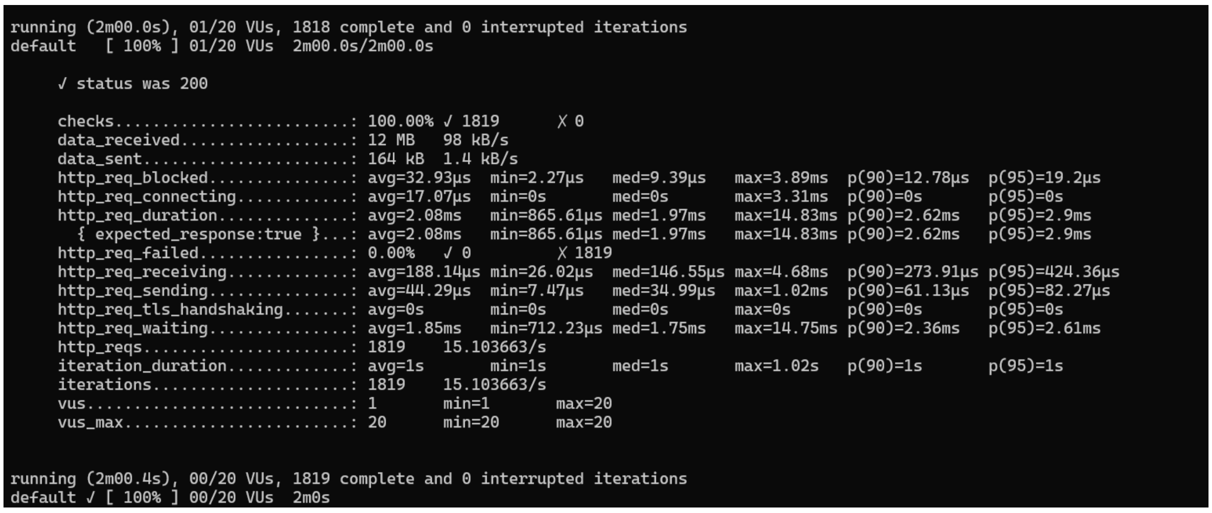


Label	Received KB/sec	Sent KB/sec	Average ↓	Throughput
HTTP Request	1307.95	44.41	117	166.0/sec
TOTAL	1307.95	44.41	117	166.0/sec

Figure 35. *Jmeter Summary II Catalogue*

Negotiator

Since the Negotiator application is behind a proxy, and it was really complex to bypass that proxy from JMeter, we opted to perform load tests using K6 only. As can be seen in both the K6 test results and the Grafana graph, the generated load volume does not even consume 1% of the computing capacity of our application, and all requests are responded to satisfactorily.



```
running (2m00.0s), 01/20 VUs, 1818 complete and 0 interrupted iterations
default   [ 100% ] 01/20 VUs  2m00.0s/2m00.0s

     ✓ status was 200

     checks.........................: 100.00% ✓ 1819      ✗ 0
     data_received..................: 12 MB   98 kB/s
     data_sent......................: 164 kB  1.4 kB/s
     http_req_blocked...............: avg=32.93µs  min=2.27µs   med=9.39µs   max=3.89ms  p(90)=12.78µs  p(95)=19.2µs
     http_req_connecting............: avg=17.07µs  min=0s       med=0s       max=3.31ms  p(90)=0s       p(95)=0s
     http_req_duration..............: avg=2.08ms   min=865.61µs med=1.97ms   max=14.83ms p(90)=2.62ms   p(95)=2.9ms
       { expected_response:true }...: avg=2.08ms   min=865.61µs med=1.97ms   max=14.83ms p(90)=2.62ms   p(95)=2.9ms
     http_req_failed................: 0.00%   ✓ 0         ✗ 1819
     http_req_receiving.............: avg=188.14µs min=26.02µs  med=146.55µs max=4.68ms  p(90)=273.91µs p(95)=424.36µs
     http_req_sending...............: avg=44.29µs  min=7.47µs   med=34.99µs  max=1.02ms  p(90)=61.13µs  p(95)=82.27µs
     http_req_tls_handshaking.......: avg=0s       min=0s       med=0s       max=0s      p(90)=0s       p(95)=0s
     http_req_waiting...............: avg=1.85ms   min=712.23µs med=1.75ms   max=14.75ms p(90)=2.36ms   p(95)=2.61ms
     http_reqs......................: 1819    15.103663/s
     iteration_duration.............: avg=1s       min=1s       med=1s       max=1.02s   p(90)=1s       p(95)=1s
     iterations.....................: 1819    15.103663/s
     vus............................: 1       min=1       max=20
     vus_max........................: 20      min=20      max=20


running (2m00.4s), 00/20 VUs, 1819 complete and 0 interrupted iterations
default ✓ [ 100% ] 00/20 VUs  2m0s
```

Figure 36. *K6 Negotiator test results*

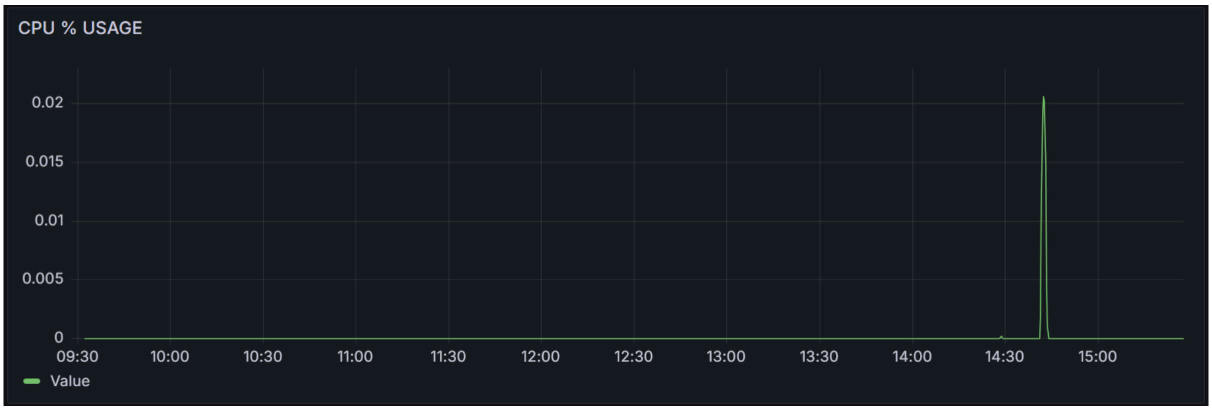


Figure 37. *Grafana Negotiator results*

Due to the difficulty of logging into this service from JMeter and being able to perform load tests, in this work we have opted to create jobs within our own infrastructure that make direct calls to the Negotiator service and observe its response under load. In previous tests, we have already verified that the network can handle a high load of requests, so we believe it is not necessary to simulate a large number of users in this test as we did in the previous ones to determine if the system will respond correctly. From the previous tests, we can conclude that it will.

7. Conclusions and Future Plan

7.1 Future Plan

During the final drafting of this work, various ideas or proposals not previously considered have emerged. We believe that these ideas could be added in the future to improve the documentation and depth of this work. Therefore, we leave the reader with some of these improvement proposals:

Scalability of CI/CD

By configuring the number of Runner Deployments to automatically adjust based on demand, we could automatically scale up or down the number of runners running on a given machine. This would facilitate future deployments, ensuring that developers do not need to worry about whether they have enough Runner nodes to execute the tasks of the Actions, as well as saving resources when those pods are underutilised

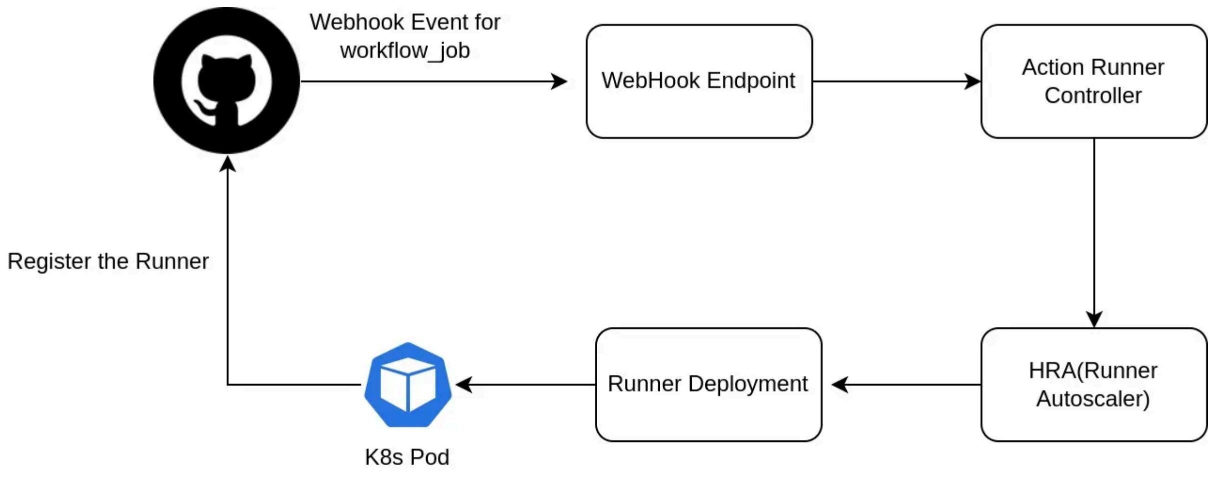


Figure 36. *GitHub Actions WebHook Flow*

This improvement can be easily implemented by adding a Webhook endpoint to our model. Webhooks allow you to subscribe to events occurring within a software system and automatically receive data delivery to your server whenever those events happen. [26][27]

Unlike polling an API, where you intermittently check for new data, webhooks push data to your server in real-time as events occur. This means you only need to register your interest in an event once, when you create the webhook, and you'll receive updates automatically.

Azure Kubernetes Service (AKS) Implementation

This improvement would be truly interesting from the perspectives of scalability, availability, and compatibility (three non-functional requirements mentioned in section 3.2.2 Non-functional Requirements). By exploring the possibility of deploying our recipes on Azure, we could achieve increased scalability, as we could scale quickly and easily in cases of sudden demand spikes. This would also protect the continuity of our service by extending beyond our physical infrastructure, thus ensuring high availability. Finally, we could ensure the compatibility of our deployment beyond the environments in which it has been tested (OpenStack[59] and OpenNebula[60]).

Replacing NFS with CephFS

As discussed in section 5.1.2 NFS (Network File System), our deployment uses NFS as the file management system. This improvement proposes replacing NFS with CephFS, thereby

[59] https://www.openstack.org/
[60] https://opennebula.io/

avoiding several intermediate connections between the pod's persistent volume and the physical storage managed by Ceph. [29]

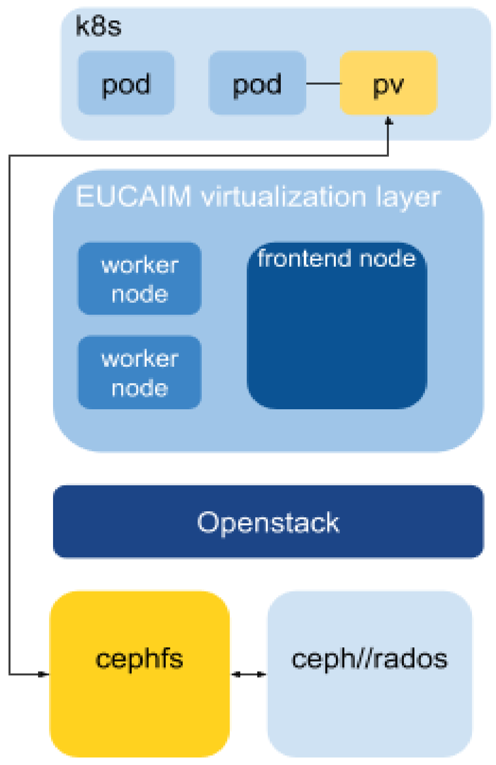


Figure 37. *NFS Proposal*

Improving the Reliability of Load Testing

It would be interesting to conduct more exhaustive load tests that encompass the overall functionality of the application. It would be beneficial to perform load tests that involve access requests to some of the different databases to see how our infrastructure behaves under a high volume of requests to the database systems.

7.2 Contributions and Conclusion

Once this work is completed, along with the Master's Degree in Cloud and High-Performance Computing, I would like to briefly review the contributions of this work as well as its relationship with everything learned during the master's program.

This work, which is currently in production at dashboard.eucaim.cancerimage.eu, has improved the deployments of this platform in the following aspects:

1. Firstly, as mentioned in the objectives, the automation of this environment has been significantly improved by integrating the CI/CD (Continuous Integration/Continuous Deployment) pipeline. This has allowed for the seamless deployment of updates, reducing manual intervention and ensuring that the platform remains up-to-date with the latest features and security patches. The CI/CD pipeline has automated the building and deployment processes, and it has also automated part of the testing, ensuring that the YAML files we were using are fully correct. This has significantly decreased the time required for these tasks and minimised the risk of human error.
2. Secondly, this work has implemented a blue-green deployment architecture, improving the deployment process in several aspects:
    a. **Zero Downtime:** Blue-green deployment allows for seamless updates without any downtime. The new version of the application is deployed to a separate environment (the "green" environment) while the current version continues to run in the "blue" environment. Once the new version is verified, traffic is switched to the green environment, ensuring continuous availability of the application.
    b. **Safe Rollback:** If any issues are detected in the new version, traffic can be quickly switched back to the blue environment, allowing for an immediate rollback. This ensures a higher level of reliability and reduces the risk associated with deployments.
    c. **Improved Testing**: The green environment can be thoroughly tested with production-like traffic before going live. This helps in identifying and resolving issues that might not be apparent in a staging environment, leading to a more stable and reliable application.
3. After applying best practices for creating recipes, we can say now that our application is more secure and less exposed to human errors.
4. Finally, after transferring our deployment to another machine and getting it back into production, we can say that our understanding of the deployment needs and relevant configuration aspects has greatly increased. This improved understanding, along with thorough documentation of these configurations, will facilitate future deployments on other machines.

As can be observed, these improvements, as well as all the work presented in this document, are closely related to everything I learned during my time in the Master's Degree in Cloud and High-Performance Computing.

Practically everything discussed and developed in this work, from the concept of containers, the use of NFS, the creation of Kubernetes recipes, the management of the cluster using kubectl, the use of Docker, etc., has been explained and developed in some of the courses of this master’s program. Special mention goes to the course "Container Management

Platforms" (PGC) taught by Professor Ignacio Blanquer Espert, who is also the advisor of this thesis.

8. Bibliography

[1] https://health.ec.europa.eu/system/files/2022-02/eu_cancer-plan_en_0.pdf

[2]
https://www.cuatrecasas.com/es/spain/propiedad-intelectual/art/consejo-ue-aprueba-data-ac

[3] https://www.consilium.europa.eu/en/policies/data-protection/data-protection-regulation/

[4] https://digital-strategy.ec.europa.eu/en/policies/data-spaces

[5] https://www.go-fair.org/fair-principles/

[6]
https://www.bde.es/wbe/es/noticias-eventos/actualidad-banco-espana/el-ine--la-aeat--la-seguridad-social--el-banco-de-espana-y-el-sepe-firman-un-acuerdo-para-permitir-el-acceso-conjunto-a-sus-bases-de-datos-para-trabajos-cientificos-de-investigacion-de-interes-publico.html#

[7] https://kubernetes.io/es/docs/concepts/workloads/pods/pod/

[8] https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

[9] https://kubernetes.io/docs/concepts/storage/persistent-volumes/

[10] https://kubernetes.io/es/docs/concepts/workloads/controllers/replicaset/

[11] https://kubernetes.io/es/docs/concepts/workloads/controllers/statefulset/

[12] https://kubernetes.io/es/docs/concepts/workloads/controllers/daemonset/

[13] https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

[14] https://kubernetes.io/es/docs/concepts/services-networking/service/

[15] https://kubernetes.io/docs/concepts/workloads/controllers/job/

[16] https://kubernetes.io/es/docs/concepts/configuration/configmap/

[17] https://kubernetes.io/docs/concepts/configuration/secret/

[18] https://www.ibm.com/docs/es/i/7.5?topic=security-network-file-system-nfs

[19] https://debian-handbook.info/browse/es-ES/stable/sect.nfs-file-server.html

[20] https://kubernetes.io/blog/2019/01/15/container-storage-interface-ga/

[21] https://github.com/kubernetes-sigs/nfs-subdir-external-provisioner

[22] https://kubernetes.io/docs/reference/access-authn-authz/rbac/

[23]
https://www.redhat.com/es/topics/devops/what-is-ci-cd#:~:text=CI%2FCD%20es%20la%20sigla,vida%20de%20desarrollo%20del%20software.

[24] https://www.redhat.com/en/topics/devops/what-is-blue-green-deployment

[25] https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions

[26] https://docs.github.com/en/actions/learn-github-actions/finding-and-customizing-actions

[27] https://www.redhat.com/en/topics/automation/what-is-a-webhook

[28] https://docs.github.com/en/webhooks/about-webhooks

[29] https://docs.ceph.com/en/latest/cephfs/

[30] https://docs.ceph.com/en/reef/architecture/

[31] https://docs.ceph.com/en/latest/

8.1 Figures Index

8.2 Tools Documentation

Apache JMeter. (n.d.). https://jmeter.apache.org/

GitHub Actions. (n.d.). https://github.com/features/actions

Kubernetes Documentation. (n.d.). https://kubernetes.io/docs/home/

Prometheus. (n.d.). https://prometheus.io/

Grafana. (n.d.). https://grafana.com/

Docker Hub. (n.d.). https://hub.docker.com/

k6 Load Testing Tool. (n.d.). https://k6.io/

CephFS. (n.d.). https://docs.ceph.com/en/latest/cephfs/

OpenStack. (n.d.). https://www.openstack.org/

OpenNebula. (n.d.). https://opennebula.io/

Appendix 1. Relationship with the Sustainable Development Goals (SDGs)

On September 25, 2015, world leaders adopted 17 Sustainable Development Goals (SDGs) to protect the planet, combat poverty, and strive to eradicate it, with the aim of building a more prosperous, just, and sustainable world for future generations. These goals were set within the 2030 Agenda for Sustainable Development.

The 17 SDGs aim to involve governments, businesses, civil society, and individuals. Each goal includes various targets, each with its own indicators to determine whether the objective is being met. Among the set of SDGs are goals focused on the advancement of clean energy, decent work and economic growth, responsible consumption and production, climate action, industry, innovation, and infrastructure.

Degree of Relation of the Work with the Sustainable Development Goals (SDGs)

Sustainable Development Goals	**High**	**Medium**	**Low**	**Not Applicable**
SDG 1. **No Poverty**				**x**
SDG 2. **Zero Hunger**				**x**
SDG 3. **Good Health and Well-Being**	**x**			
SDG 4. **Quality Education**				**x**
SDG 5. **Gender Equality**				**x**
SDG 6. **Clean Water and Sanitation**				**x**
SDG 7. **Affordable and clean energy**				**x**

SDG 8. **Decent work and economic growth**				x
SDG 9. **Industry, Innovation and Infrastructure**			x	
SDG 10. **Reduced inequality**				x
SDG 11. **Sustainable cities and communities**				x
SDG 12. **Responsible consumption and production**				x
SDG 13. **Climate action**				x
SDG 14. **Life below water**				x
SDG 15. **Life on land**				x
SDG 16. **Peace, justice and strong institution**				x
SDG 17. **Strengthen the means of implementation and revitalise the global partnership for sustainable development goals**				x

In this section, it is mentioned which of the previously mentioned SDGs are related to this project.

3 GOOD HEALTH AND WELL-BEING

Objective 3: Ensure healthy lives and promote well-being for all at all ages

Being a project with direct application in the health field, this project aims to directly improve aspects related to achieving a healthier life for everyone at all ages. The facilities offered in data processing as well as in their accessibility can be of great help both for professionals in this field and for patients.