Project title: European Federation for Cancer Images

Project acronym: EUCAIM

Grant Agreement: 101100633

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

D4.9 Central Core Infrastructure set-up

Author(s):		Ignacio Blanquer (UPV), Pablo Montoliu (UPV), Marcel Koek (Erasmus MC), Esther Bron (Erasmus MC, Health-RI), Pau Lozano (UPV), Andrei S. Alic (UPV)
Reviewers (WP)		Carina Soler (HULAFE), Ignacio-Gomez Rico (HULAFE), Celia Martin (QUIBIM)
WP2		
WP3		
WP4	Carina Soler (HULAFE), Ignacio Gómez-Rico (HULAFE)	
WP5		
Date of delivery:		30/06/2024
Version:		V0.1
Due date:		Month 18
Actual delivery date:		
Type:		DEM + R
Dissemination level:		Public

Table of contents

Abbreviations

Terms	**Definitions**
AAI	Authentication and Authorisation Infrastructure
AARC	Authentication and Authorisation for Research and Collaboration
AI4HI	AI4HI AI for Health Imaging Network
API	Application Programming Interface
Auth	Authentication
BBMRI-ERIC	European Infrastructure for Biobanking - European Research Infrastructure Consortium
CEPH	A free and open-source software-defined storage platform that provides object storage, block storage, and file storage
DBMS	Database Management Service
DCAT-AP	Data Catalogue vocabulary Application Profile
DCM4CHEE	A Java-based library and set of tools for working with DICOM files
DICOM	Digital Imaging and Communication In Medicine
DNS	Domain Name System
EduGain	Global interfederation service that interconnects multiple identity federations
EGI	European Grid Infrastructure
ELK	Elasticsearch, Logstash, Kibana
EOSC	European Open Science Cloud
EU	European Union
EUCAIM	European Federation for Cancer Images
FAIR	Findable, Accessible, Interoperable, Reusable
FDP	FAIR Data Point
GbE	Gigabit Ethernet
GDPR	General Data Protection Regulation
GPU	Graphics Processing Unit
Guacamole	A clientless remote desktop gateway
GUI	Graphical User Interface
IdP	Identity Provider
IM	Infrastructure Manager
K8S	Kubernetes
KubeApps	In-cluster web-based application for the management of Kubernetes applications
KubeAuthoriser	A federated cloud architecture for processing of cancer images on a distributed storage

Kyverno	A policy engine designed specifically for Kubernetes
LS-AAI	Life Sciences Authentication and Authorisation Infrastructure
LTS	Long Term Support
MOLGENIS	A modular web application for scientific data, initially focused on molecular genetics research (molecular genetics information system) but expanded to other disciplines.
Negotiator	BBMRI-ERIC service for structured negotiator for biomedical resources
NFS	Network Filesystem
OAuth2	Open Authorisation v. 2.0
OpenID	Open standard and decentralised authentication protocol
PACS	PACS Picture Archiving and Communication System
Postgres	Object-oriented relational Database management system
PV	Persistent Volume object
PVC	Persistent Volume Claim
QUIBIM	Spanish company on AI applied to Image Biomarkers
RIS	RIS Radiological Information System
TB	Terabyte
TOSCA	Topology and Orchestration Specification for Cloud Applications
UPS	Uninterrupted Power Service
UPV	Universitat Politècnica de València (Valencia University of Technology)
VAULT	Hasihcorp identity-based secrets and encryption management system
VM	Virtual Machine
VMI	Virtual Machine Image
VO	Virtual Organisation
WP	WP Work Package

Disclaimer

1. Introduction

EUCAIM leverages from the developments on other Research Infrastructures for health data, extending and customising services that are key for metadata cataloguing, federated search, access negotiation, data storage and computing-intensive processing.

Those services have been adapted and customised to deal with the specificities of EUCAIM and to enable their integration. These components have been prepared for being deployed on a Kubernetes infrastructure that manages the availability, network traffic and workloads efficiently. The document describes the backend infrastructure and the deployment manifests for each one of the core services of EUCAIM.

EUCAIM constitutes a federated infrastructure. The Reference (sometimes called central) storages are reference implementations of storage and processing nodes that can be replicated around the federation. The basic federation of EUCAIM described here involves two reference storages and two AI4HI providers.

EUCAIM also deploys two core service infrastructures, one for production and one for development, using the same components and under two different DNS. Both infrastructures have the same AAI registrations and use configuration files and environment variables for customising each deployment.

2. Backend infrastructure

The backend infrastructure comprises the hardware resources and the middleware layers used to run the services of EUCAIM. It has been divided into three blocks: Hardware resources, cloud management layer and container management layer.

2.1. Principles of the infrastructure

The architecture should support the following design principles:

- High availability. The hardware should be provided by an Uninterrupted Power Service (UPS), replicated storage and a reliable control plane.
- Efficiency. The hardware resources and the middleware software layers should provide enough resources and should enable high performance processing through GPU accelerators.
- Security. Exposure of services must be minimised, access to resources should be limited, users should be properly authenticated and authorised and all communications in external networks should be encrypted.
- Convenience. The infrastructure should rely on widely used components and protocols, reducing the migration burden from the providers.

2.2. Hardware fabric

The core services run on hardware resources shared with the reference storage and processing environment located at UPV, although the core services environment is isolated from the reference storage at the software level. The infrastructure at UPV, at the release of this deliverable, comprises 5 nodes Gigabyte R283-ZF0-AAL1 with 2 CPUs AMD EPYC 9474F 48-Core Processor each, summing up 480 cores, 3,84TB RAM and 15 A30 GPU accelerators with 24 GB of RAM each, as well as an additional storage server with 16 TB of NVMe link SSD disks connected through 25+25GbE to the nodes.

In the coming months, it is expected to expand the storage to 250 TB using similar technology and reach by Q1 2025 a total of 12 servers (1152 cores), 9TB of RAM and 36 GPU accelerators.

2.3. Cloud Backend

The hardware resources are offered in the format of a cloud on-premise offering, implemented through OpenStack Bobcat. OpenStack is the *de facto* standard in on-premise cloud management frameworks, and provides maintenance and frequent updates. The deployment exposes four public endpoints:

- Dashboard (Horizon): https://imaging.i3m.upv.es/horizon/
- Authentication (Keystone): https://imaging.i3m.upv.es:5000/v3/
- Virtual Machine Image Catalogue (glance): https://imaging.i3m.upv.es:19292
- Network manager (neutron); https://imaging.i3m.upv.es:19696

All public endpoints require authentication and are limited to the administration managers of the EUCAIM core services infrastructures. Access to the endpoints is restricted to the internal network of the UPV.

A specific project has been created within the platform to isolate the permissions of the users that manage the EUCAIM cores services from the users who manage the platform. The types of instances for the core system are:

- Instance types:
    - Large VM (12 cores, 62,5GB RAM)
    - Small (4 cores, 23,4GB RAM)
- VMIs:
    - Ubuntu 24.04 LTS (noble)
    - Ubuntu 22.04 LTS (jammy)

On top of those virtual resources, a Kubernetes infrastructure is deployed, which will be the place where the core services are deployed.

2.4. Kubernetes Backend

All the core services are embedded into software containers that are run on the platform through a Kubernetes container management platform.

Kubernetes is installed using Infrastructure Manager (https://www.grycap.upv.es/im) through the EGI IM Dashboard (https://im.egi.eu/im-dashboard/), a cloud orchestration solution which is widely used in e-Infrastructures, such as the EGI Federated cloud (https://www.egi.eu/egi-infrastructure/) and the Application Workflow Manager of the EOSC Core managed services of the EOSC EU Node.

The set up includes:

- One Small VM for the Kubernetes front-end.
- Three Large VMs for the worker nodes of Kubernetes.

Infrastructure manager facilitates the scalability of the platform by providing a convenient interface to expand or reduce the number of nodes. Reducing the nodes triggers the reallocation of the containers automatically. Expanding the nodes will open additional space for additional (or pending) workloads.

Along with the Kubernetes system, the following components are installed:

- A Flannel network plugin.
- The Kubernetes Dashboard.
- A Helm service to deploy Helm charts of applications.
- Let’s encrypt certificate manager.
- An NFS Persistent Volume provisioner.

The specification of the Kubernetes installation recipes follow the TOSCA standard (Topology and Orchestration Specification for Cloud Applications). The template of the recipe is available in https://github.com/grycap/tosca/blob/main/templates/kubernetes.yaml.

3. Kubernetes components

3.1. Design Principles

The core services deployment manifests have been implemented considering the following implementation principles:

- Every component is deployed on a different namespace.
- Reduced surface exposure by exposing only the essential endpoints.
- Endpoints accessible through https protocols on standard ports only.
- Services are instantiated through high-availability deployments.
- Storage is provided through an NFS Persistent Volume.

All the Kubernetes manifests are available in a GitHub Repository of the project (https://github.com/EUCAIM/k8s-deployments/)

3.2. Storage Backend

Containers are ephemeral and the local storage is volatile. Any file stored in the containers will disappear if the container is restarted. A persistent storage is attached via NFS persistent volumes to each one of the components. In some cases, components require more than one volume and they are provided as separate volumes.

The matching between the Persistent Volume Claim (PVC) and the Persistent Volume (PV) is performed through the definition of a specific StorageClass for each type of volume, easing traceability. Figure 1 shows a schema of the general deployment of each PV in the cluster. The manifests for each one of the PV types are included in the manifests of each component.

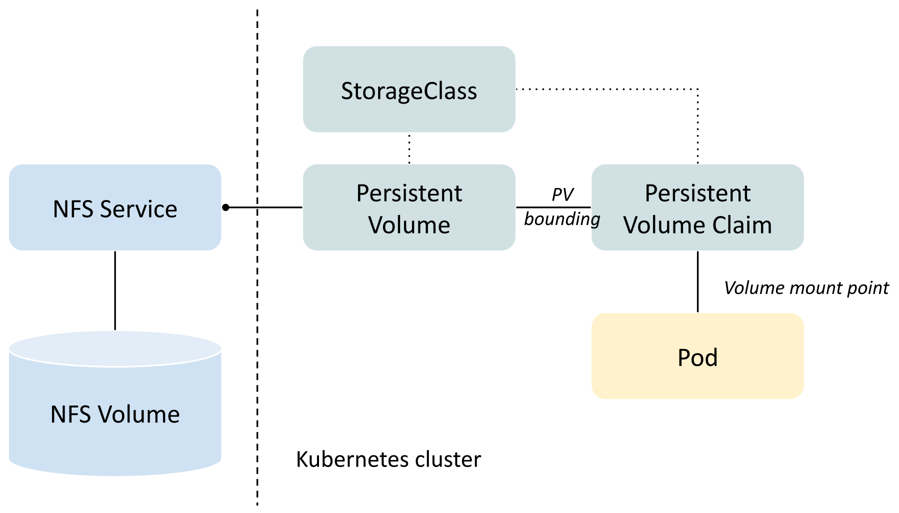


Figure 1. *Structure of Storage in the Kubernetes deployments.*

3.3. High Availability

High availability is implemented through three approaches:

- Use of scalable deployments with potentially multiple instances. This is implemented in the services with higher exposure which could suffer a higher workload.
- Use of automatic restart policies for the pods, ensuring that services are restarted in case of unexpected errors, by using the "Always" Restart Policy of the replica sets.
- Use of blue-green deployments (see figure 2) to roll-out new versions with the capability of immediately rolling back to previous deployments.

The blue-green deployments are implemented in the following cases:

- Dashboard, with separate persistent volumes for the web application to deal with updates in the application and the container images.
- Federated Search Explorer, to deal with updates at the container level.
- Negotiator, to deal with updates at the container level.

The catalogue uses a conventional deployment as the changes are made at the database level and rarely at the container level. In any case, a development deployment has been set up to facilitate testing new versions prior to launching them in the production environment.

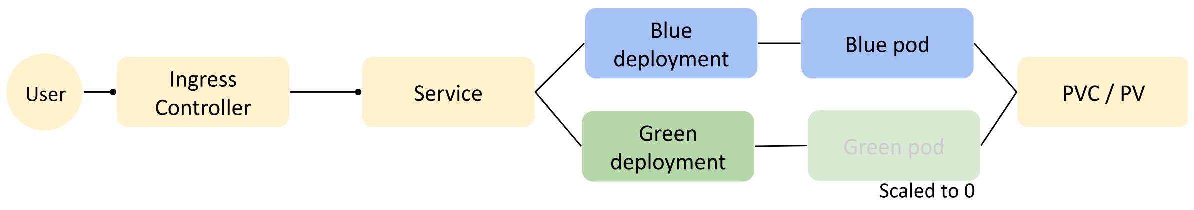


Figure 2. *Blue-green deployments. Two identical deployments with an ingress controller pointing out to the active one. The inactive deployment is scaled to 0 and is the one where changes are applied when rolling out a new application, changing the ingress controller to the green service. When the application is completely verified, the blue deployment is scaled to 0 and becomes the future testing environment.*

3.4. Components

This section briefly describes each one of the deployment artefacts for each one of the components that define the core services of EUCAIM.

The namespaces in each component deployed are self-explanatory (dashboard, catalogue, explorer, negotiator, monitoring).

It is important to outline that services are accessible only in the private overlay network of the Cilium plugin in Kubernetes. Only the services that require external access are exposed through an nginx proxy and using encrypted protocols.

Detailed descriptions of the components can be found in *Deliverable D4.5 First Federated Core Services*. Instructions for its usage can be found in *Deliverable D4.13 End-user guide to the system.*

3.4.1. Dashboard

The Dashboard is the main entry point to the EUCAIM environment. It is described in detail in Deliverable *D4.7 First EUCAIM Dashboard*. The manifests that describe the deployment are available in GitHub[1]. The schema of the components is shown in Figure 3. The Dashboard implies four main components:

- Two PV with the application and the database files.
- A MongoDB database running in a deployment, persisting the database files through a PVC and internally accessible through a service, which is used by the Dashboard application.
- A nodeJS server running in a deployment, mounting the dashboard application through a PVC and exposed through the `dashb-node-service`.
- An ingress controller that exposes the dashboard service in the root of the dashboard.* DNS.

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

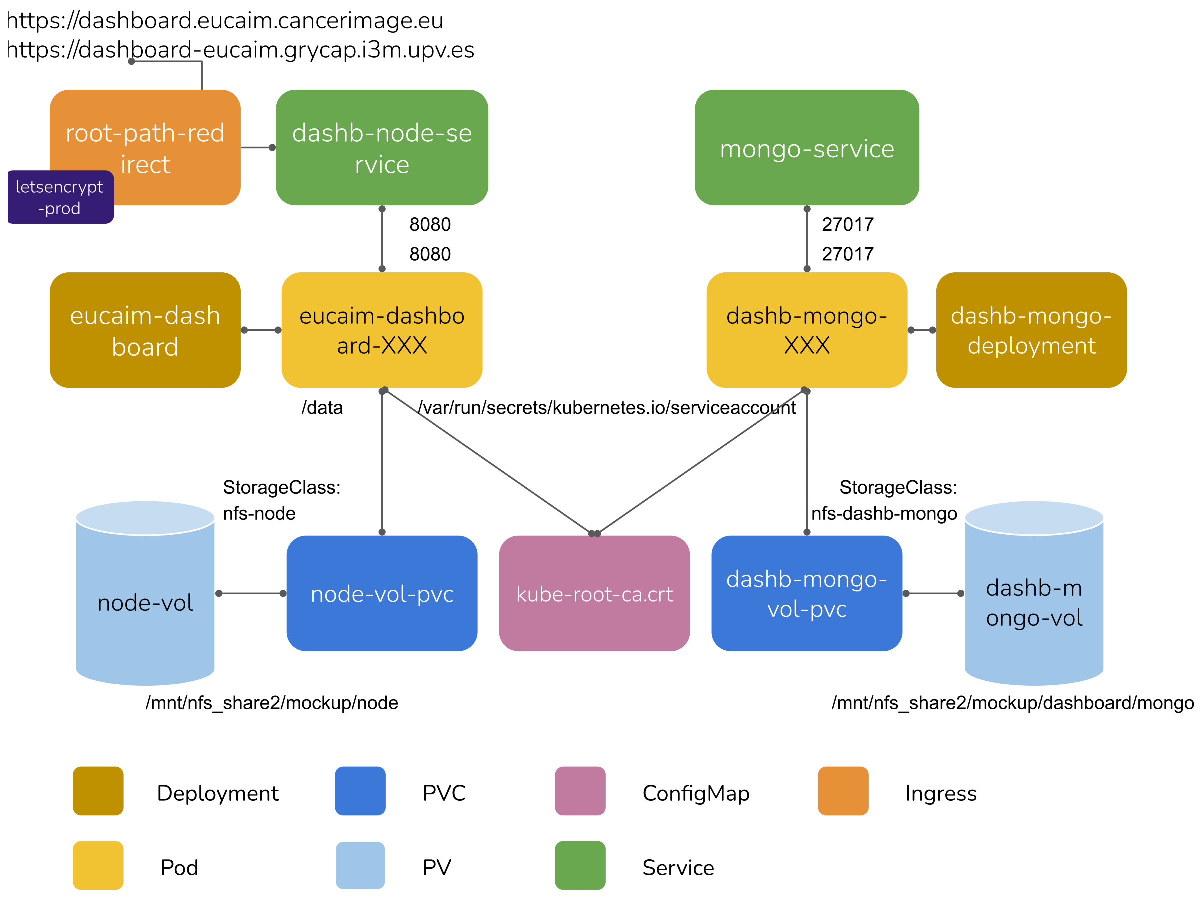


Figure 3: *Schema of Kubernetes manifests for the deployment of the Dashboard.*

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.

3.4.2. Public Catalogue

The public catalogue stores the metadata, offering the researchers descriptive information about the available datasets. The catalogue consists of the Molgenis 10.1 platform as a back-end with a custom Javascript front-end which is based on prior catalogues. The whole architecture of the Molgenis platform has been minimised to the components relevant for EUCAIM.

The manifests that are used for the deployment of the catalogue may be also found in GitHub[2]. The architecture of the solution is shown in Figure 4 and Figure 5. The deployment consists of four elements:

1. **Molgenis deployment:** This deployment is configured through two ConfigMaps that define the service parameters and two Persistent Volumes (PV) where the logs from the application (audit PV) and the backups and the necessary files for running the application backend (app-data PV) are stored.
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 mounted through a Persistent Volume Claim (PVC) mounted to the PV where the necessary files for running the Molgenis user interface are stored.
3. **Postgres database:** A database, where the catalogue metadata is stored. The Postgres database information is stored in the postgres-vol2 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 5, 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 optimize content delivery and application functionality.

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

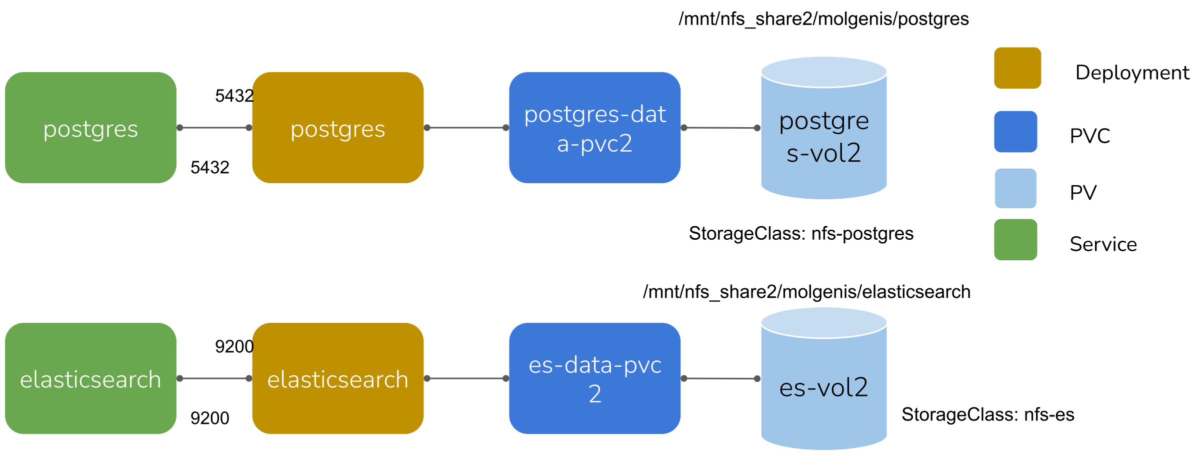


Figure 4. *Deployment schema of the postgres and elasticsearch deployments in the catalogue.*

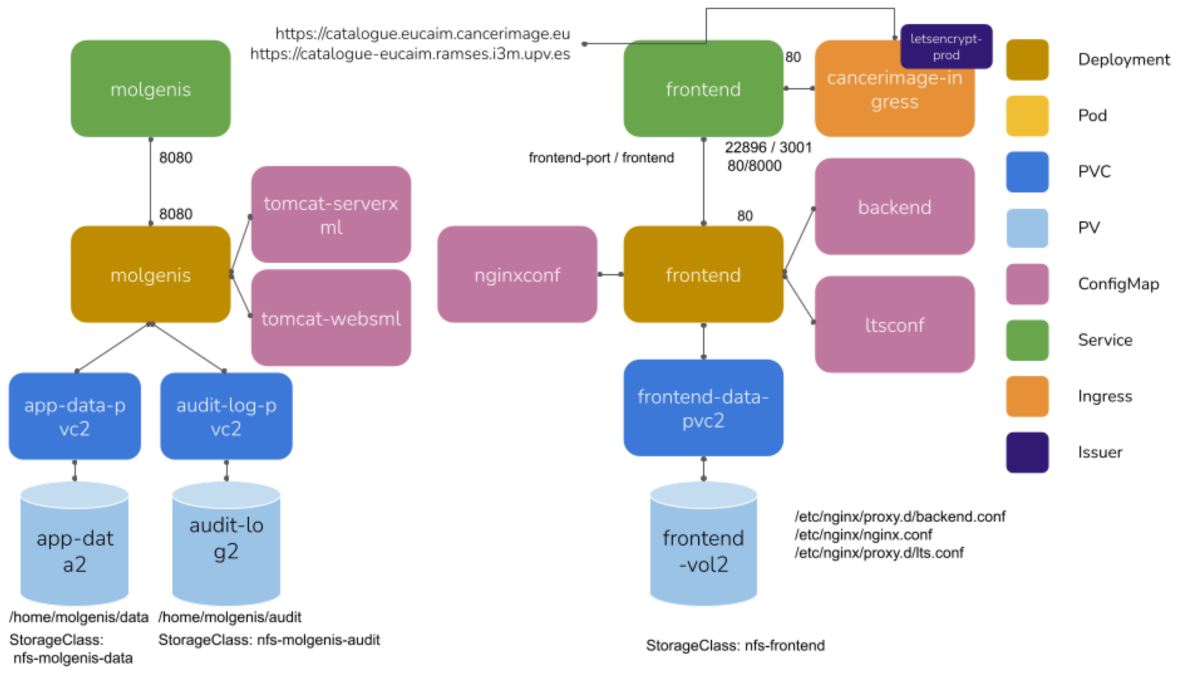


Figure 5. *Deployment schema of the Molgenis backend and frontend deployments.*

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 private repository https://gitlab.com/radiology/infrastructure/projects/catalogue/eucaim-molgenis-app

3.4.3. Federated Search

The 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 at 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 urls: https://explorer.eucaim.cancerimage.eu/ and https://explorer-eucaim.grycap.i3m.upv.es/

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.

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 6 shows the deployment at the central service and Figure 7 shows the deployment at the data holders, as well as the interaction among the components. As can be seen in Figure 7, 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, the pod objects associated with the deployments are not included.

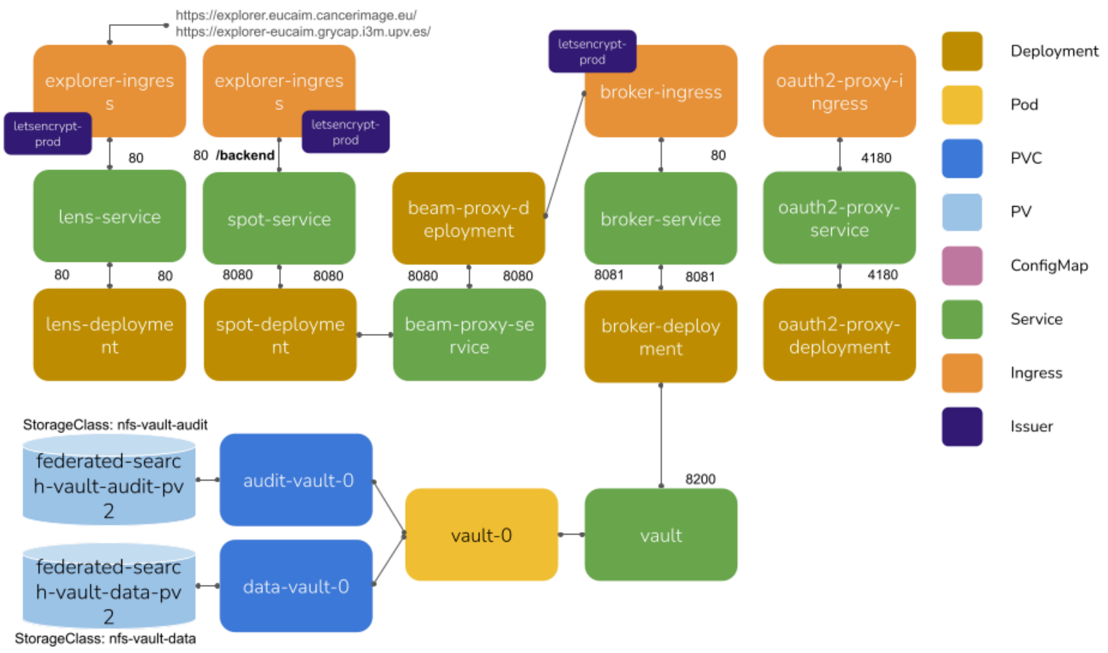


Figure 6. *Deployment schema of the federated search at the core services level.*

The manifests and some more information about the deployment of this service may be found at: https://github.com/EUCAIM/k8s-deployments/tree/main/federated-search/eucaim.

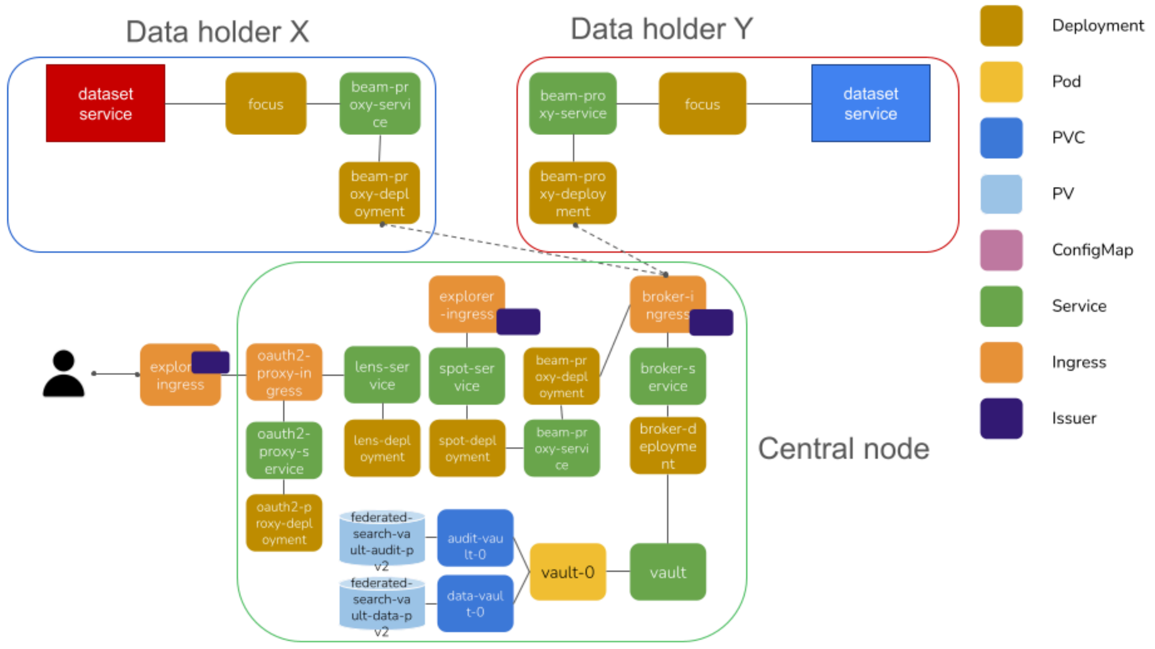


Figure 7*. Deployment schema of the federated search including the interactions with the data holders.*

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 persistent storage except for the case of the access credentials, which are 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[3]. 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.

3.4.4. Negotiator

The negotiator component deployment manifests are available in GitHub[4]. It comprises three main services that implement the UI, the backend and the database. Although the UI is the

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

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

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 8 describes the names and interactions of the components.

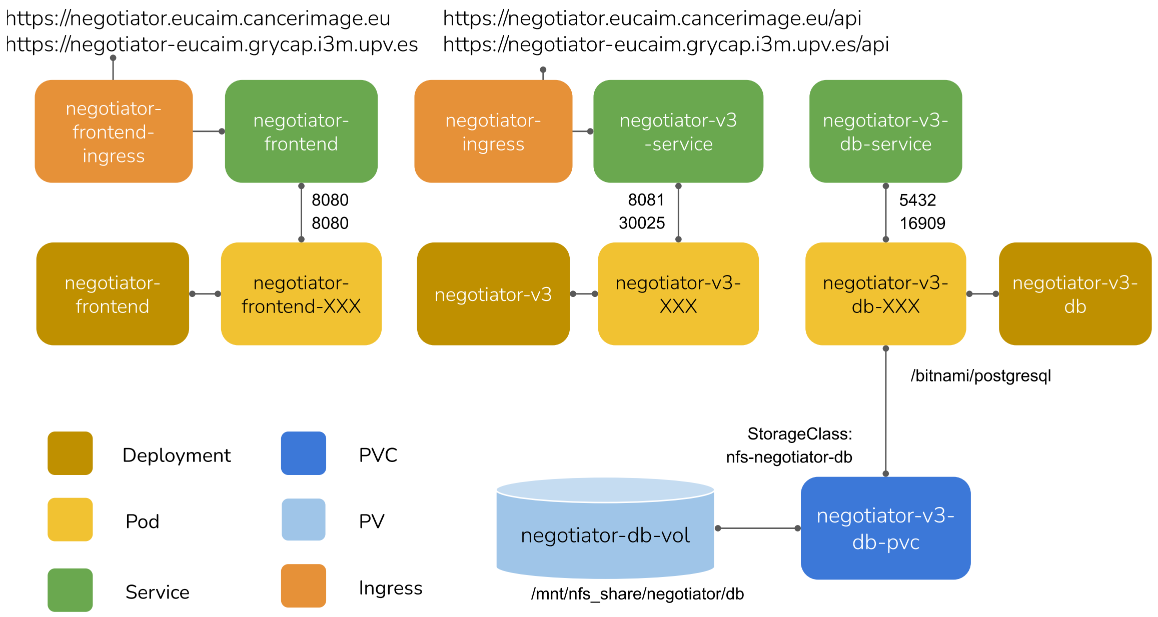


Figure 8*: Schema of Kubernetes manifests for the deployment of the Negotiator.*

The deployment first creates the PV and the PVC, and then instantiate 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. Especially take into account:
    - The Java Web Token issuer, which is indicated in the environment variable: SPRING_SECURITY_OAUTH2_RESOURCESERVER_JWT_ISSUERURI should be ended by a backslash (e.g. "https://login.aai.lifescience-ri.eu/oidc/").
    - The resource server audiences for the Java Web token (SPRING_SECURITY_OAUTH2_RESOURCESERVER_JWT_AUDIENCES) should contain the client ID of the UI rather than the URL (e.g. "a15a95d1-b251-4f12-b608-76ec02c68010").
- Update the LS-AAI configuration in the frontend (AUTH_URL, CLIENT_ID and REDIRECT_URI)
- 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.

3.4.5. 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[5], 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.

For the deployment of these services, different YAML manifests have been used, which can be found in the EUCAIM project's GitHub[6]. Once all the aforementioned components are deployed, they will start interacting and the alert and monitoring service will be launched, being accessible at the URL https://elastic-eucaim.grycap.i3m.upv.es/. For more information on the interaction between the different components please refer to Figure 9, which shows the interaction at a high level of abstraction, or Figure 10, that depicts the interaction between the components at the Kubernetes object-level.

[5] Kubernetes operator pattern: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/
[6] GitHub repository EUCAIM/monitoring-services: https://github.com/EUCAIM/monitoring-services

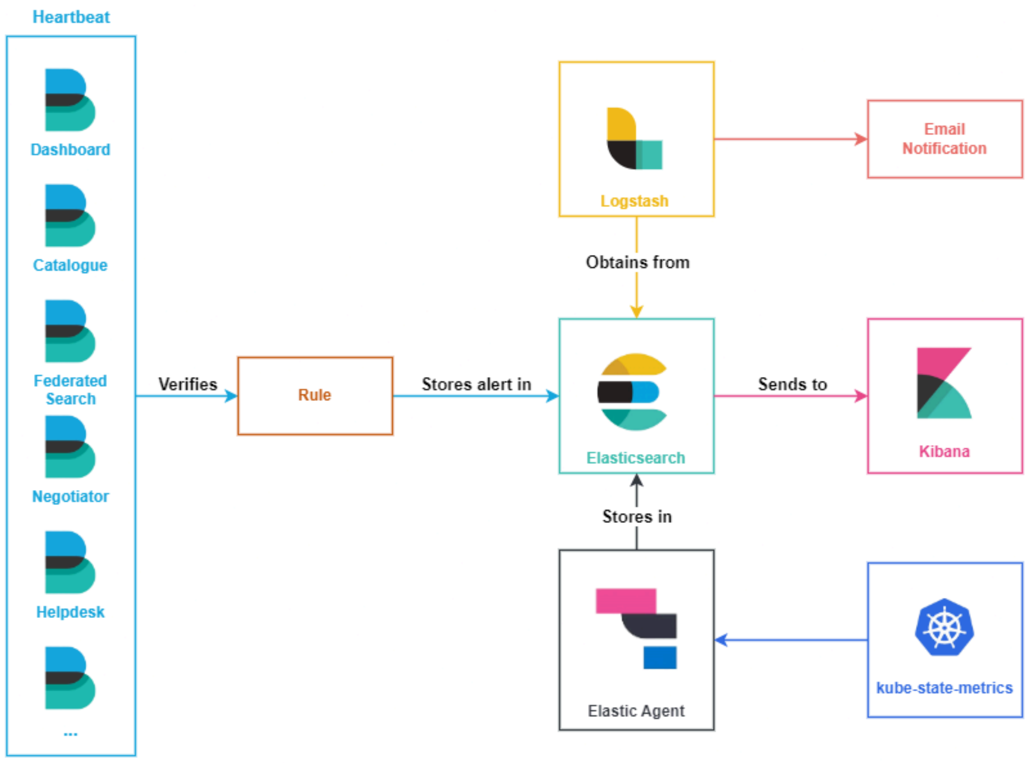


Figure 9. *Interaction diagram between monitoring service components.*

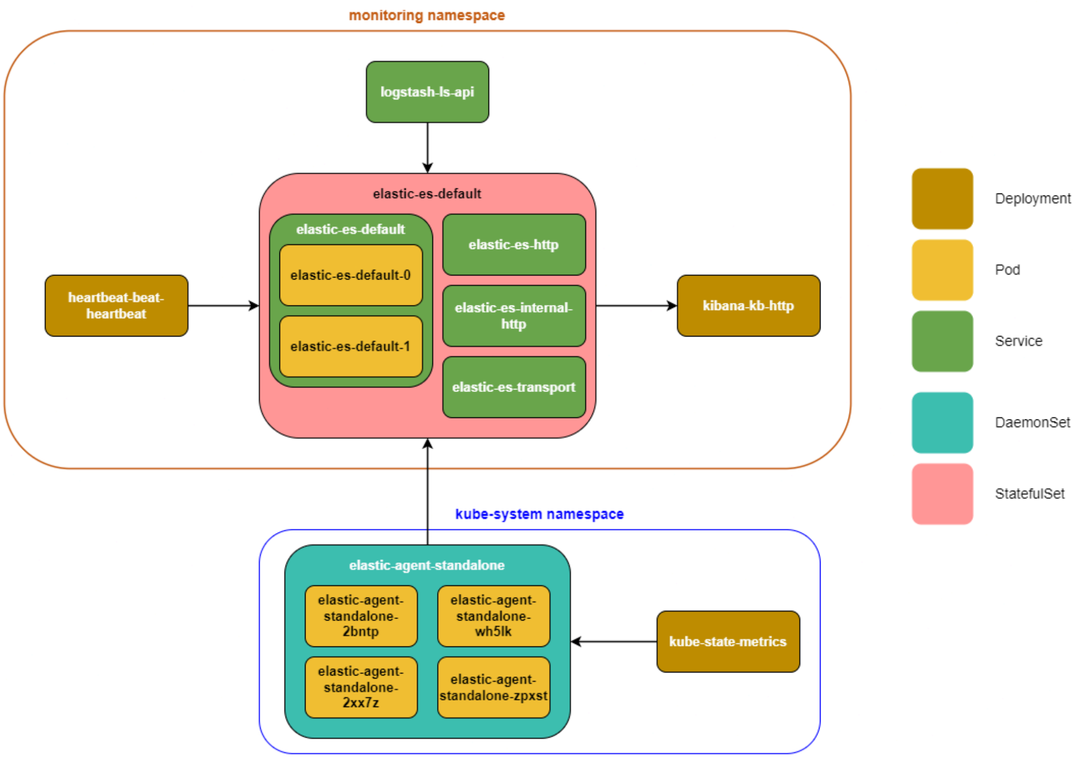


Figure 10. *Kubernetes object-level interaction diagram.*

3.5. External services

Along with the services deployed in the central node, EUCAIM uses external services for managing the Authentication and Authorisation and the Helpdesk ticketing.

3.5.1. LS-AAI

EUCAIM supports authentication based on OpenID. EUCAIM uses the Life Science Authentication and Authorisation Infrastructure (LS-AAI) in all the components. This facilitates the administration of the user's permission (performed through the membership to the EUCAIM VO Group. In addition to this, the LS-AAI enables third party applications, such as the federated nodes that already use the LS-AAI, to seamlessly integrate to the EUCAIM Federation.

All the application endpoints are registered in the LS-AAI catalogue:

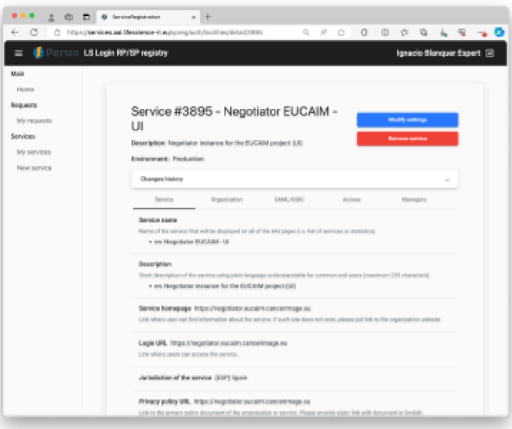
- Dashboard (dashboard.eucaim.cancerimage.eu)
- Federated Search (explorer.eucaim.cancerimage.eu)
- Negotiator UI (negotiator.eucaim.cancerimage.eu)
- Negotiator backend (negotiator.eucaim.cancerimage.eu/api)
- Helpdesk (helpdesk.eucaim.cancerimage.eu)
- Reference Node at UPV (eucaim-node.i3m.upv.es)

Each service has a client ID and a secret that are used for the authentication at the level of the service. The services receive the following entitlements from the LS-AAI service.

- urn:geant:lifescience-ri.eu:group:lifescience:communities_and_projects:EUCAIM#aai.lifescience-ri.eu

The basic configuration of a service in EUCAIM is shown in Figure 11

Alternatively, a Keycloak instance could be deployed to manage the authorisation of all the services and users.



Service #3895 - Negotiator EUCAIM - UI

Description: Negotiator instance for the EUCAIM project (UI)

Environment: Production

Service name
- en: Negotiator EUCAIM - UI

Description
- en: Negotiator instance for the EUCAIM project (UI)

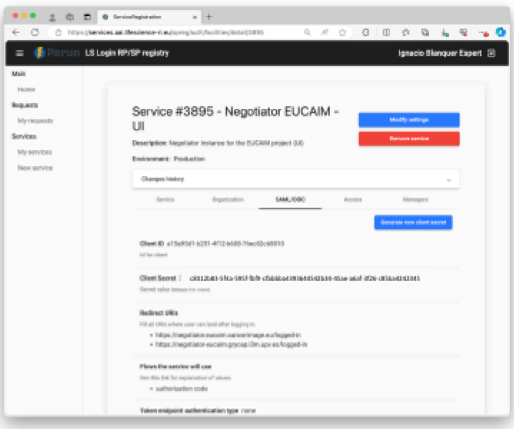
Service homepage https://negotiator.eucaim.cancerimage.eu

Login URL https://negotiator.eucaim.cancerimage.eu

Jurisdiction of the service (ESP) Spain

Privacy policy URL https://negotiator.eucaim.cancerimage.eu

a)



Service #3895 - Negotiator EUCAIM - UI

Description: Negotiator instance for the EUCAIM project (UI)

Environment: Production

Client ID [illegible]

Client Secret [illegible]

Redirect URIs
- https://negotiator.eucaim.cancerimage.eu/logged-in
- https://negotiator-eucaim.grycap.i3m.upv.es/logged-in

Flows the service will use
- authorization code

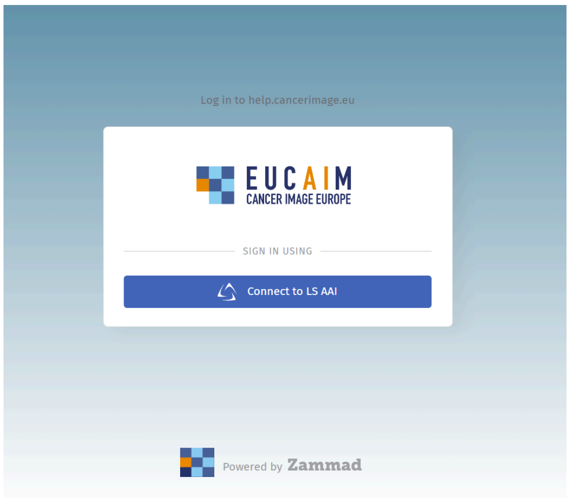
Token endpoint authentication type none

b)

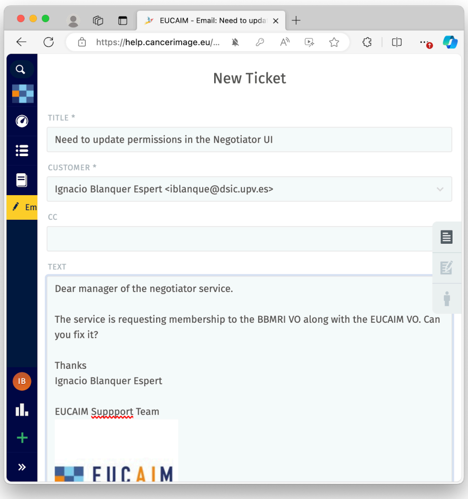
Figure 11: Service configuration of one of the EUCAIM core services. Service description (a) and OIDC configuration (b).

3.5.2. Helpdesk

The helpdesk instance is hosted by the Karlsruhe Institute of Technology (KIT) and it has been configured to accept LS-AAI accounts. Therefore, it is not deployed along with the rest of the core services in the central node, and no additional details are provided in this document. Figure 12 shows the entry page[7] of the Helpdesk and the creation of a ticket.



a)



EUCAIM - Email: Need to upda

https://help.cancerimage.eu/...

New Ticket

TITLE *

Need to update permissions in the Negotiator UI

CUSTOMER *

Ignacio Blanquer Espert <iblanque@dsic.upv.es>

CC

TEXT

Dear manager of the negotiator service.

The service is requesting membership to the BBMRI VO along with the EUCAIM VO. Can you fix it?

Thanks
Ignacio Blanquer Espert

EUCAIM Suppport Team

b)

Figure 12. EUCAIM Helpdesk. a) Entry page and b) creation of a ticket.

4. Reference Nodes

Data holders that cannot set up a federated node should transfer the data to a reference node in the infrastructure. Both UPV and Erasmus MC provide reference nodes for the community, each one with a different protocol and requirements, facilitating the uptake of data holders in terms of geographical coverage and requirements.

The core services run the services required for the federation. Along with those services, the UPV and Erasmus MC nodes constitute two complementary examples of the computing and storage nodes that can be linked to the federation.

4.1. UPV Reference Node

The architecture of the UPV reference node is described in Figure 13. It comprises the following components:

- User-level components
    - A Dataset explorer, which comprises the catalogue and the dashboard for accessing the data.
    - The Case Explorer, which provides visualisation of DICOM images and the capacity of creating datasets.
    - The application registry, based on KubeApps.
    - A clientless remote desktop gateway, based on Apache Guacamole.

[7] Helpdesk: https://help.cancerimage.eu/#login

- Platform-level components
    - A DICOM PACS Based on DCM4CHEE that acts as a DICOM Server.
    - The Keycloack AAI, which manages authentication and the KubeAuth AAI Proxy, which enacts the authorisation pipelines.
    - The Dataset Service, which manages the catalogue.
    - The Tracer, which registers the accesses into a BlockChain and a database.
    - A Policy Manager, based on Kyverno, that stores the policies that describe permissions and can be associated with users or groups.
    - The Jobman service for running jobs on the platform.
- Infrastructure level components
    - A Storage resource, based on a CEPH server with high-availability and replication that manages the storage volumes used by the VMs and the processing platform.
    - Processing Resources, based on Kubernetes which are used to deploy the interactive and the batch applications.

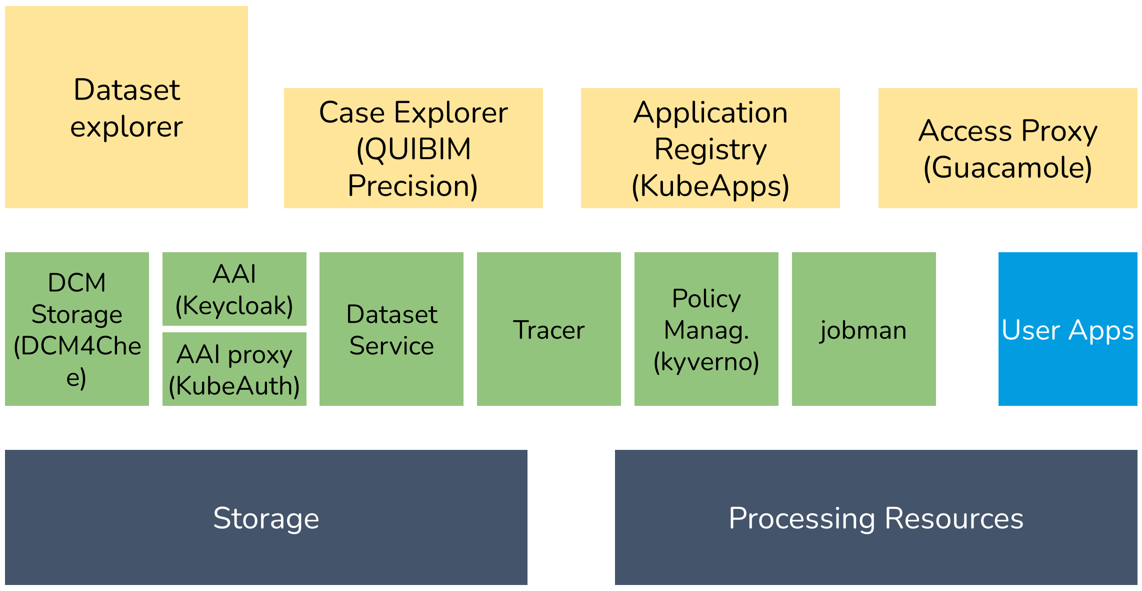


Figure 13. *Deployment schema of the components in the UPV Reference Node.*

The code of the reference node is available in https://github.com/chaimeleon-eu. The specific deployment of the UPV reference node has been performed following the Kubernetes manifests in https://github.com/EUCAIM/k8s-deploy-node/tree/master.

The deployment of UPV reference node requires a CEPH storage and a Kubernentes cluster. UPV provides a TOSCA cloud orchestrator to deploy those virtual infrastructures on top of an on-premise cloud platform (see section 2.4 of this document).

The deployment will use 5 dedicated physical nodes, each one 96 cores (AMD EPYC 9474F processor), 756 GB of RAM, 3 NVIDIA A30 GPUs with 24GB RAM each and a local SSD disk with 1 TB. The total figures are: 480 (real, double if hyperthreaded) cores, 3780 GB of RAM, 15 A30 GPUs and 5 TB of local disk space (permanent storage is on a different server with 75 TB of storage in NVE Link high-speed disks, connected through a double 25 Gb link to the switch of the processing services.

This will enable the platform to deal with a concurrent processing capacity of:

- 40 Interactive desktops with 1.2-4 cores, 16 GB RAM and 5 GB disk each. Interactive desktops are the main interface of the VRE for the users.
- A queue for small jobs with 8 slots, each one for jobs up to 5.5 cores, 32 GB RAM a 0.25 of an A30 GPU (6 GB) and 40 GB of storage.
- A queue for medium jobs with 4 slots, each one capable of running jobs up to 11 cores, 64 GB RAM, 0.5 A30 GPU (12 GB), and 50 GB of storage.
- A queue for large jobs with 11 slots, each one capable of running jobs up to 12 cores, 70 GB RAM and a full A30 GPU with 24 GB, plus 50 GB of local storage.
- A queue for non-gpu jobs, with 4 slots for jobs requesting up to 12 cores, 70 GB RAM and 50 GB of storage.

The services running in the UPV reference node are the following:

- Security services
  - Authentication: Keycloak is deployed using plain YAMLs from https://github.com/EUCAIM/k8s-deploy-node/tree/master/keycloak and exposed in the URL: https://node-eucaim.i3m.upv.es/auth/.
  - Authorization: Kube-authorizer is deployed using plain YAMLs from https://gitlab.com/primageproject/kube-authorizer and only accessible internally.
  - Security Policy Management System: Kyverno is deployed using the official helm chart as explained in https://github.com/EUCAIM/k8s-deploy-node/tree/master/kyverno using the “baseline” Pod Security Standard Policy and other policies fromhttps://github.com/EUCAIM/k8s-deploy-node/tree/master/kyverno/policies.
  - Authentication proxy: OAuth2-proxy is deployed using the official helm chart as explained in https://github.com/EUCAIM/k8s-deploy-node/tree/master/oauth2p

Container image and Helm Chart repository:

- Harbor: deployed using the official helm chart as explained in https://github.com/chaimeleon-eu/k8s-deployments/tree/master/harbor and exposed in https://harbor.node-eucaim.i3m.upv.es

Interact with Kubernetes (deploy resources):

- Kubeapps (for normal users): deployed using plain YAMLs from https://github.com/EUCAIM/k8s-deploy-node/tree/master/kubeapps and exposed in https://node-eucaim.i3m.upv.es/apps
- Kubernetes Dashboard (only for administrators): deployed using the Kubernetes Ansible role. URL: https://node-eucaim.i3m.upv.es/dashboard

Interact with deployed resources:

- Guacamole: deployed using the helm chart as explained in https://github.com/EUCAIM/k8s-deploy-node/tree/master/guacamole. And exposed in: https://chaimeleon-eu.i3m.upv.es/guacamole
- Kubernetes Dashboard: Only available for administrators.

Ingestion services:

- We currently have deployed Quibim Precision and DC4CHEE, following the helm chart as explained in https://gitlab.com/primageproject/k8s_quibimprecision and the

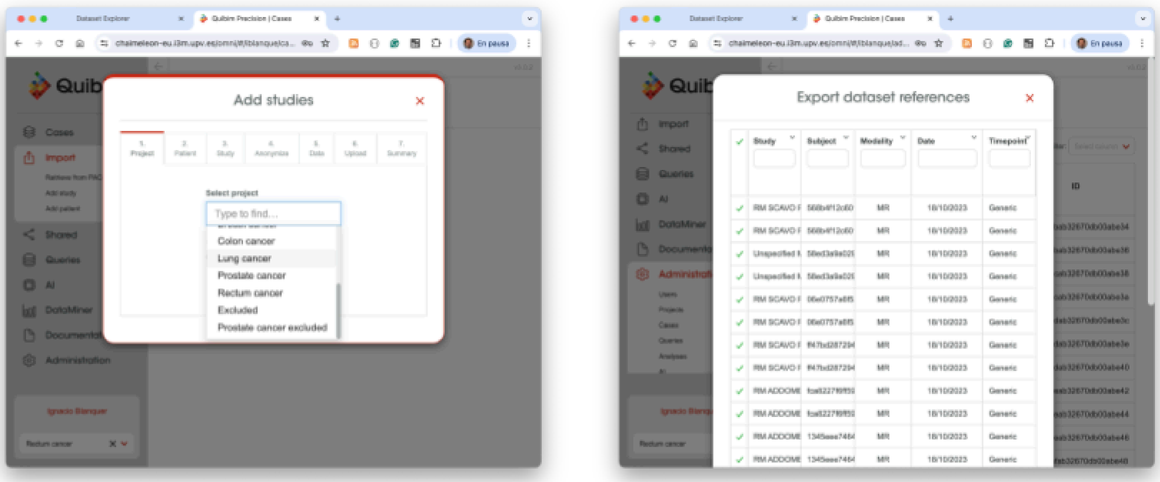
DCM4CHEE PACS using the plain YAMLs from https://gitlab.com/primageproject/k8s_quibimprecision/-/tree/master/without_chart/pacs. However, both services will be replaced by QP-Insights in the short time, as soon as the integration tests are completed.

Dataset administration and Traceability System:

- Dataset service: all the details of deployment and usage in https://github.com/chaimeleon-eu/dataset-service
  REST API exposed in: https://node-eucaim.i3m.upv.es/dataset-service/api
  WEB UI (Dataset explorer) exposed in:
  https://node-eucaim.i3m.upv.es/dataset-service
- Chaimeleon K8s Operator: deployed using the helm chart as explained in https://github.com/chaimeleon-eu/k8s-chaimeleon-operator.
- Tracer service: all the details of deployment and usage in https://github.com/chaimeleon-eu/tracer
  REST API exposed in: https://node-eucaim.i3m.upv.es/tracer-service/tracer
  WEB UI included in the Dataset Service end-point (Dataset explorer).

4.2.1 Data ingestion

Figure 14 shows the uploading of data and the creation of a dataset in the case of the UPV reference node as an example.



✓	Study	Subject	Modality	Date	Timepoint
✓	RM SCAVO F	566b4f12c60	MR	18/10/2023	Generic
✓	RM SCAVO F	566b4f12c60	MR	18/10/2023	Generic
✓	Unspecified I.	58ed3a9a02	MR	18/10/2023	Generic
✓	Unspecified I.	58ed3a9a02	MR	18/10/2023	Generic
✓	RM SCAVO F	06e0757a8f5	MR	18/10/2023	Generic
✓	RM SCAVO F	06e0757a8f5	MR	18/10/2023	Generic
✓	RM SCAVO F	f47bd287294	MR	18/10/2023	Generic
✓	RM SCAVO F	f47bd287294	MR	18/10/2023	Generic
✓	RM ADDOME	fca822789f5	MR	18/10/2023	Generic
✓	RM ADDOME	fca822789f5	MR	18/10/2023	Generic
✓	RM ADDOME	1345aaa7464	MR	18/10/2023	Generic
✓	RM ADDOME	1345aaa7464	MR	18/10/2023	Generic

a) b)

Figure 14, adding a study to the reference storage a), and creating a collection from the data uploaded b).

4.2. Euro-BioImaging Reference Node

The Euro-BioImaging Medical Imaging Repository (a reference node in EUCAIM) focussed on storing data for Data Holders. The underlying platform for storing and managing imaging data is XNAT. XNAT is an extensible open-source imaging platform that simplifies common tasks in imaging data management. The Euro-BioImaging Medical Imaging Repository service is an XNAT instance operated by Health-RI. Erasmus MC is providing 2nd line support and is responsible for managing the service roadmap. XNAT allows its user to manage large imaging datasets, adjust users permissions, preview and review data, as well as run processing pipelines. In EUCAIM the Euro-BioImaging XNAT will be used to store imaging data and will be integrated with other core services.

4.2.1. High level Architecture

The high-level architecture of the Euro-BioImaging EUCAIM reference node is depicted in Figure 15. The components described are:

- *Q API*: Query API. Refer to the Federated Core Services deliverable (D4.5) for more information
- *Dashboard*: EUCAIM Dashboard. Refer to the Dashboard deliverable (D4.7) for more information
- *Catalogue*: Place where to store metadata about the collections contained in the Repository. Refer to the Federated Core Services deliverable (D4.5) for more information
- *AAI*: Authentication & Authorization Infrastructure
- *FDP*: Fair Data Point (means of exposing the metadata of a repository: https://www.fairdatapoint.org/). We refer to the Federated Core Services deliverable (D4.5) for more information.
- *Web UI*: User interface for data management, application management, data browsing and inspection, use of a case explorer and data annotation, etc. We refer to the end-user guide deliverable (D4.13) and the future training materials
- *A/IO API*: Access and Data I/O API, this is a REST API
- *Annotation and analysis*: data resulting from annotation and analysis of raw data.
- *Data Holder*: Center or project that holds data
- *Ingestion & Curation*: Data ingestion from the Data Holder into the Repository

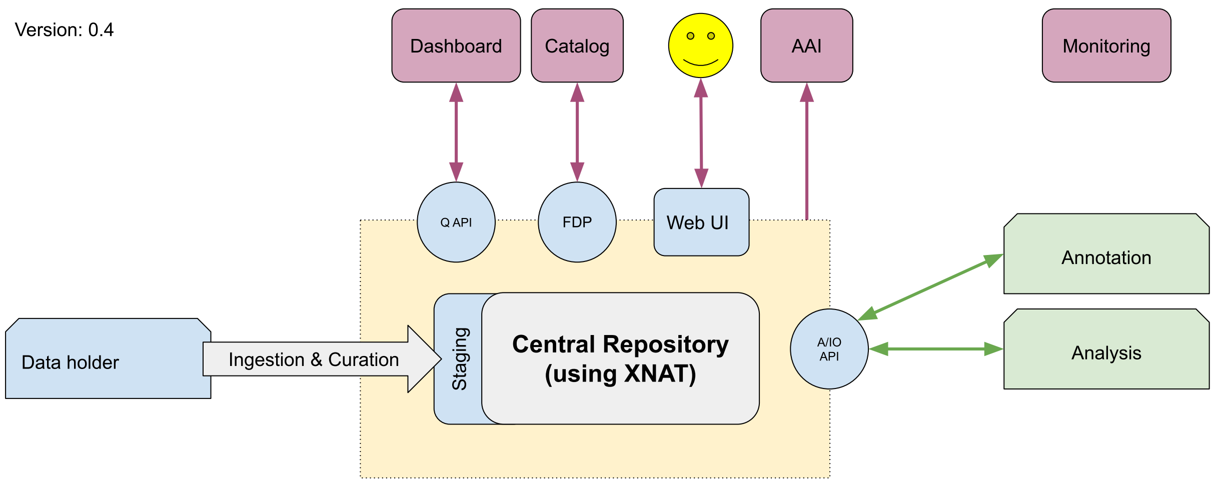


Figure 15. *Deployment schema of the components in the Euro-BioImaging Reference Node.*

4.2.2. Technical implementation

The XNAT server is a standard XNAT installation (currently version 1.7.5.6). From a high level view, XNAT consists of three big parts: a relational database backend, middleware components, implemented in Java, and user web interface. First, data types stored by XNAT are described using XML schema definition (XSD). Based on these definitions, XNAT then generates corresponding tables in the database as well as necessary UI components for the interface. Modern versions of XNAT use PostgreSQL database to store and look up the file locations of the imaging data corresponding to a certain subject, scan session and so on. The imaging data itself is not stored in this PostgreSQL database. XNAT uses the file locations in this database to find specific images stored separately on the machine where XNAT is deployed. Furthermore, XNAT offers a REST API, facilitating easy accessibility and interoperability. The REST API caters for both data interaction and configuration and administrative tasks. Since the Euro-BioImaging XNAT is an unmodified instance, we refer to the official documentation for a more detailed overview of its functionality and architecture.[8]

4.2.3. Data ingestion

For ingesting data, multiple options are available, depending on the specific capabilities of the Data Holder and the data format, the XNAT DICOM Receiver or the A/IO API can be used to upload data to the XNAT repository.

- Clinical Trail Processor: https://mircwiki.rsna.org/index.php?title=MIRC_CTP)
- XNATpy: https://xnat.readthedocs.io/)

The Clinical Trail Processor (CTP), is an advanced DICOM processing tool. This is the preferred tool to upload DICOM data to the Euro-BioImaging XNAT.

In the Euro-BioImaging Medical Imaging Repository, a CTP is placed between the XNAT built-in DICOM receiver and the public internet. Because the communication protocols of DICOM are not designed for public facing networks, this has to be protected. Two CTP's can act as a bridge between a Data Holderand the Euro-BioImaging XNAT using an encrypted connection. CTP is able to process DICOM headers and can be used for anonymizing the data. It is fully compliant with the DICOM standard.

XNATpy is a python client, developed by Erasmus MC, which interacts with XNAT through the XNAT API. It offers programmatic access as well as Command Line Interface (CLI) interaction. XNATpy adapts itself to the XNAT instance it communicates with through the XNAT data model XSD definitions. XNATpy offers query and search functionality. It is recommended by the international XNAT community as the tool to use for programmatic access to XNAT.

5. Conclusions

This deliverable describes the backend infrastructure and the deployment of the core services that manage the federation of EUCAIM. In a nutshell, EUCAIM core services are based on YAML manifests that are deployed on top of a Kubernetes cluster that runs on a virtual infrastructure managed by an OpenStack. The deployment of the components facilitate isolation from the actual data, which is stored on a separate deployment. Most of the components are released under open source licences and use well known APIs, so replacement of components is feasible.

This document complements the description of the architecture of the core services in D4.5 and the end-user manual described in D4.13.

[8] https://wiki.xnat.org/documentation/xnat-administration/understanding-the-components-of-xnat