Future Generation Computer Systems 139 (2023) 38–52

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A federated cloud architecture for processing of cancer images on a distributed storage

J. Damián Segrelles Quilis *, Sergio López-Huguet, Pau Lozano, Ignacio Blanquer

Instituto de Instrumentación para Imagen Molecular (I3M), Universitat Politècnica de València (UPV), Camino de Vera S/N, 46022 Valencia, Spain

ARTICLE INFO

Article history:
Received 15 June 2022
Received in revised form 7 September 2022
Accepted 19 September 2022
Available online 22 September 2022

MSC:
00-01
99-00

Keywords:
Medical imaging
Biomarkers
Storage and computing backends

ABSTRACT

The increased accuracy and exhaustivity of modern Artificial Intelligence techniques in supporting the analysis of complex data, such as medical images, have exponentially increased real-world data collection for research purposes. This fact has led to the development of international repositories and high-performance computing solutions to deal with the computational demand for training models. However, other stages in the development of medical imaging biomarkers do not require such intensive computing resources, which has led to the convenience of integrating different computing backends tailored for the processing demands of the various stages of processing workflows. We present in this article a distributed and federated repository architecture for the development and application of medical image biomarkers that combines multiple cloud storages with cloud and HPC processing backends. The architecture has been deployed to serve the PRIMAGE (H2020 826494) project, aiming to collect and manage data from paediatric cancer. The repository seamlessly integrates distributed storage backends, an elastic Kubernetes cluster on a cloud on-premises and a supercomputer. Processing jobs are handled through a single control platform, synchronising data on demand. The article shows the specification of the different types of applications and a validation through a use case that make use of most of the features of the platform.

1. Introduction and background

Increasingly, radiology is based on objective and quantifiable data extracted from *Quantitative Imaging Biomarkers* (QIBs). QIBs are quantitative indicators generated from structural, functional, physiological or biological characteristics of pathological lesions [1]. In the workflow development of QIBs, complex computational functions and models automatically extract attributes, namely radiomics features, from different types of radiological images to correlate them to the phenotype or genetic signatures of the lesions. These analyses aim to early detect and classify anomalies to predict prognostics, define follow-up results, or non-invasively assess the treatment response.

In the last years, developers have analysed the images by learning from retrospective data, enriching radiomics features with demographic, clinical, liquid biopsies and genomic data because they improve the clinical value of the biomarkers [2]. Thus, gathering data processes are crucial to developing useful *Clinical Decision support Systems* (CDSS) based on QIBs in clinical practice, requiring a massive storage and high-performance computing capacity [3] for managing data on image biobanks. Furthermore, the huge amount of data makes traditional statistical analyses impractical, leading to a transition to novel textitArtificial Intelligence (AI) solutions such as Deep Learning [4]. Running AI algorithms efficiently requires high-computing performance resources [5] connected to the data storage backends.

The *Quantitative Imaging Biomarker Alliance* (QIBA) from the *Radiological Society of North America* (RSNA) and the *European Imaging Biomarker Alliance* (EIBALL) of the *European Society of Radiology* (ESR) highly support the adoption of this new paradigm based on QIBs. In addition, the ESR has defined guidelines and best practices [1,6] to develop and validate QIBs, which are used as a baseline in this work to design a federated architecture that allows processing images and associated data in distributed repositories of cancer imaging, with the goal of supporting the implementation of the whole QIB development workflow.

In this context, PRIMAGE [7] and CHAIMELEON [8] projects are working to accelerate the development process and build novel QIBs based on AI solutions following the ESR guidelines. On the one hand, the PRIMAGE Project aims at providing an open cloud-based platform to support decision making in the clinical management of two rare paediatric cancers, *Neuroblastoma* (NB), the most frequent solid cancer of early childhood, and the *Diffuse Intrinsic Pontine Glioma* (DIPG), the leading cause of brain tumour-related death in children. The platform offers CDSS tools to assist on diagnosis, prediction, prognosis, therapy choice and

* Corresponding author.
E-mail address: dquilis@dsic.upv.es (J. Damián Segrelles Quilis).

https://doi.org/10.1016/j.future.2022.09.019

treatment follow-up. PRIMAGE CDSS tools are built on top of AI-based QIBs and translate this knowledge into predictors for the most relevant, disease-specific, *Clinical End Points* (CEPs). On the other hand, the CHAIMELEON project aims at setting up a repository of health imaging data, processing tools and methodologies, providing datasets as FAIR *(Findable, Accessible, Interoperable, and Reusable)* research objects, along with storage and computing resources for AI experimentation for cancer management. The repository aims to accelerate the development of CDSS tools based on AI-based QIB solutions providing datasets, computing and processing backends to data scientists and developers and offering clinical staff the developed tools through a marketplace. CHAIMELEON focuses on four types of cancer that currently have the highest prevalence worldwide: lung, breast, prostate and colorectal.

The work presented in this paper is mainly framed on the PRIMAGE project. The main objective is to design a federated architecture for processing data in a distributed repository of cancer images providing storage and processing resources (including both infrastructure and software tools) for covering all the requirements identified in the QIB ESR development guidelines.

The main contribution of the architecture is the integration of heterogeneous computing backends tailored for different workloads on a federated and synchronised distributed data storage. The article focuses on the deployment of the proposed architecture for the development of QIBs related to the two above mentioned paediatric cancers. The distributed storage of the architecture transfers and caches on-demand the data required by the Processing Backends. The three application abstractions (batch, HTC batch and interactive) provided by the architecture support the implementation of the QIB development cycle's life. The innovative approach of the architecture consists of the combination and integration of multiple cloud Storage Backends with Processing Backends for containers (Kubernetes cluster) and HPC batch jobs (supercomputers).

The paper has the following structure. First, Section 2 describes the ESR development guidelines for QIBs as a baseline to extract the requirements to design the proposed architecture. Then, in Section 3 describes the requirements identified. Next section presents the proposed architecture, outlining the components that address the requirements and the work carried out for their integration. Next, as result, the deployment and validation of the federated architecture to support the QIB Community of the PRIMAGE project are described. Besides, this section also presents some scientific results from PRIMAGE developers that were accomplished through the use of the deployed federated architecture. Finally, the last section exposes the conclusions.

2. QIB ESR guideline

The QIB ESR guideline comprises four main phases [1,6]: Hypothesis; Image Acquisition and preparation for Analysis; Image Analysis and Feature Extraction; and Biomarker Validation. Fig. 1 shows an overview of the stepwise of this methodology that the following subsections briefly describe.

2.1. Hypothesis

The work should start with the definition of the hypothesis. Expert clinicians on a given disease (NB or DIPG tumours in the case of PRIMAGE) define the target facts whose knowledge could help improving early diagnosis, or more adequate treatment. These facts should correlate with the biological and physiological changes, expressed as radiomics features, that need to be extracted from medical images.

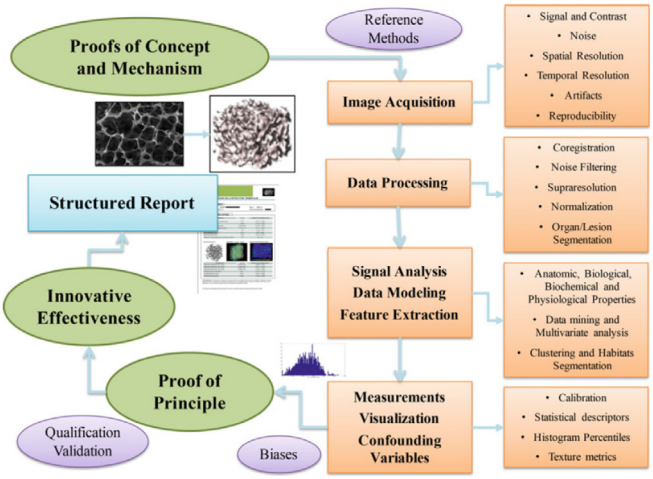

Fig. 1. Stepwise development of QIBs.
Source: Extracted from [1].

As the first step to formulate a hypothesis, a Clinician proposes a *Proof of Concept* (e.g. tumour aggressiveness is higher when the blood affluence to their tissues increases). Next, Theoretical Researchers and Engineers specialised in medical imaging (e.g. physicists, mathematicians, etc.) perform a *Proof of Mechanism* (e.g. the bloodstream on a tissue can be measured through Radiomics features extracted from b-value *Diffusion Weighted Imaging* (DWI) models applied on MRI [9]). The testing of the hypothesis requires choosing the *Reference Methods*, which are the widely recognised reference approaches or procedures that determine the actual state of the disease under evaluation (e.g. biopsy).

2.2. Image Acquisition and preparation for analysis

The appropriate acquisition of source images is essential for developing a QIB. Technicians carry out the *Image Acquisition* by collecting reproducible and standardised images. Acquired data must provide sufficient data coverage and maximise quality regarding noise ratios, spatial and temporal resolution, and artefacts. Technicians need *data processing* and *image preparation tools* (such as noise filtering and segmentation) to improve source image quality.

The acquisition should include additional data that reflect different aspects of pathological and physiological processes such as clinical [10] or genomic [11] data. These data complements and improves the clinical value of the biomarkers [2].

Technicians must anonymise the data collected, including both the images and the associated data, and code the information using standard terminologies (e.g. SNOMED-CT, RADLEX) before uploading them into the *Storage Backends*. Storage backends enable processing the data by specific computing infrastructures (*Processing Backends*) and are normally located beyond the hospital network borders. Anonymisation is a legal obligation to fulfil GDPR 679/2016 and other European-country regulations.

2.3. Image analysis and feature extraction

This stage covers the application of methods and procedures to extract radiomics features and signatures (e.g. anatomic, biological, biochemical, physiological). This stage is typically performed by Specialised Engineers and Researchers that apply those methods to either a *Region Of Interest* (ROI) (such as a lesion or an organ) or specific subvolumes. Next, they perform correlation

studies to generate multivariate analysis or data mining procedures among all gathered data (radiomics features, clinical and genomics data, reference methods, etc.).

The usage of AI methods based on Machine (Deep) Learning have become widely used to extract radiomics features and signatures [12] and to carry out correlation analysis [4]. The huge amount of data available have made conventional statistical analyses impractical. However, AI methods require high-performance computing resources that must be connected to the *Storage Backends* for efficiently processing the data.

After correlation analysis, the results are measured and quantified, assigning a number or rank to the extracted features. These results should be adequately visualised for direct and objective interpretation to assess for target-related confounders.

In addition, researchers must identify biases and consider them before the validation of a biomarker. Bias estimation quantify the systematic errors that affect and distort the measurement process, generating wrong values and misinterpreting results.

2.4. Biomarker validation

Specialised Clinicians carry out the validation and qualification of biomarkers before incorporating them in the clinical practice. The validation implies measuring the effectiveness of the biomarker and the qualification assesses the acceptability in the clinical practice. As the first step, this process usually goes through a single-centre analysis (*proof of principle*) on a small sample of well-controlled and defined cases. It aims to verify that the developed biomarker can be used in clinical practice to evaluate the accuracy and potential confounding variables. Next, if the proof of principle is successful, clinicians carry out more extensive multicenter analysis to validate the biomarkers in clinical practice routine (*proof of effectiveness*). Finally, the results of a biomarker must be presented in an intuitively manner to fully qualify for clinical practice. The disease-driven structured reports are appropriate for transmitting these results [13].

3. Requirement analysis

This section includes a requirement identification process for a platform to carry out the previous stages efficiently and collaboratively among the different actors involved. The section ends up with the design of the proposed architecture, which will be fully described in the next Section 4.

The following subsections introduce five areas, each one presenting the list of functional and non-functional requirements identified for enabling the development stages of quantitative image biomarkers according to the ESR guidelines.

3.1. Federated authentication and authorisation

ESR life cycle involves different user profiles (Clinicians, Physicians, Engineers, Radiologists, AI Developers, etc.). In each stage, the users have to access the *Backends* (both processing and storage) to carry out processing actions through the *User Applications*. A *User Application* is a software application which runs on top of the *Backend* Resources and provides the users with interfaces (such as *Command Line Interfaces* — CLIs or user-friendly web interfaces) implementing different functionalities (e.g. ingestion of data, executing data quality processes, filtering noises, segmentation processes, training AI models, validation of AI models, biomarker tools).

Thus, the proposed architecture must enable a procedure to organise QIB communities as organisations (e.g. the *Virtual Organisations* - VO). A QIB community is a set of heterogeneous users (e.g. clinicians, physicists, Engineers, Radiologist, Metrologist, Mathematicians, AI Developers) from different knowledge areas that share their resources, experiences and knowledge for developing a QIB. The belonging to a VO entitle the user to access to the distributed *Backends* and the specific *User Applications*.

Table 1 shows the main functional requirements (from F.01 to F.03) for the federated users and their ESR guidelines related stages, and Table 2 shows the main non-functional requirements (NF.1 and NF.2).

3.2. Backend deployment

Users perform actions to collect, manage, and compute medical data during the whole ESR workflow through the *User Applications* running on the *Backend*. *Backends* should be deployed on-demand to deal with the requirements of the ESR stages. Deployment in the *Backend* will provide the hardware and software resources and will configure them to support the required *User Applications*.

The Hypothesis stage initiates the workflow, and users formulate a *Proof of Concept and Mechanism*. They must deploy a *Storage Backend* to store the data required (images and data associated, Reference Methods Data) to contrast the hypothesis in the next stages.

Next, data is prepared and ingested into the *Storage Backend* (Image Acquisition and preparation for Analysis stage). In this stage, users improve data quality (e.g. noise filtering, segmentation, registration, normalisation) through specific *Users Applications* that require specific hardware requirements (e.g. GPU, MPI Cluster) that are provided by *Processing Backends*. Therefore, *Processing Backends* are deployed first and *User Applications* are instantiated on top of them.

When the data collection and the ingestion into the *Storage Backends* is completed, users can start building Biomarker Tools (Image Analysis and Feature Extraction stage). First, users must deploy *User Applications* (e.g. AI frameworks) to extract radiomics features and signatures, and perform correlation studies to generate multivariate analysis or data mining procedures employing the data collected or generated (radiomics features, clinical and genomic data, reference methods, etc.).

Finally, in the Biomarker Validation stage, users deploy Biomarkers Tools as *User Applications* to run the *proof of principle and effectiveness* stage. This can be done using the *Processing Backends*, or deploy a new *Backends* if the Biomarker Tools have different software, hardware or execution environment configuration requirements.

A *Backend* can give support to several VOs and must be configured for allowing access to authorised users (users which belong to a certain VO) only. Thus, each VO has a member with the Deploy Manager user role who is capable of deploying and releasing *Backends* the services on the *Backend*.

Table 1 shows the main functional requirements (from F.O4 to F.07) for the *Backend* deployments and the ESR guidelines related stages, and Table 2 shows the main non-functional requirements (from NF.4 to NF.07).

3.3. Storage Backends

Storage Backends are a crucial asset in the ESR life cycle. When users carry out the Image Acquisition and Preparation for Analysis stage, they ingest images, associated data and Reference Methods data to the storage where authorised users will access it in further stages. In the ingestion process, users employ (*User Applications*) to ingest, curate, and anonymise the data and improve its quality before uploading them. Data is cast to an agreed e-form following a common data model. Thus, depending on the cancer,

Table 1
List of main functional requirements and related ESR guidelines stages (S1 — Hypothesis, S2 — Image Acquisition and preparation for Analysis, S3 — Image Analysis and Feature Extraction, S4 — Biomarker Validation.

COD	Description	Stages	Components
F.01	Users must manage their identity (create/delete identity, update data identity) to authentication processes.	All stages	IdPs supported by EGI-check-in
F.02	QIB Communities must organise users into VOs (enrol/leave).	All stages	EGI Community
F.03	QIB Community must manage user roles (e.g. clinician, physicians, engineers, radiologists, AI developers) in the VO where they belong (assign/revoke role). The role determines the actions allowed for the VO in the associated *Backends* (e.g. execute applications, ingest data).	All stages	EGI Community, Kubeauthoriser, EGI Data Hub
F.04	It (the platform) must deploy/release *Storage Backends* on resources on public or on-premise cloud providers	S1	Infrastructure Manager
F.05	It must deploy/release *Processing Backends* on resources on top of public or on-premise cloud providers.	S2, S3, S4	Infrastructure Manager
F.06	It must deploy/release *User Applications* on *Processing Backends*.	S2, S3	Kubernetes
F.07	It must deploy/release Biomarker Tools on *Processing Backends*.	S4	Kubernetes
F.08	*Storage Backends* must provide disease-specific e-forms to collect data (images and associated data).	S2	QUIBIM Precision (Front-End), DCM4CHEE
F.09	*Storage Backends* must offer tools for browsing data.	S2	QUIBIM Precision (Front-End), EGI Datahub
F.10	*Storage Backends* must offer tools for creating datasets.	S2	QUIBIM Precision (Front-End)
F.11	*Storage Backends* must provide spaces to share data among users.	S3	EGI Datahub
F.12	*Storage Backends* must provide private spaces for personal data users.	S3	EGI Datahub
F.13	*Processing Backends* must offer tools to launch *User Applications*.	S2, S3	Kubernetes (Dashboard)
F.14	*Processing Backends* must offer to access running *User Applications*.	S2, S3	Guacamole, Kubernetes (Dashboard)
F.15	*User Applications* must offer tools to access Images from *Storage Backends* efficiently.	S2, S3	EGI Datahub (OneClient)
F.16	*User Applications* must offer tools to manage Clinical Data from *Storage Backends* efficiently.	S2, S3	EGI Datahub (OneClient)

the *Backend* will provide a user-friendly interface to input the specific e-form variables, collecting all interesting associated data (e.g. clinical) and linking them to the related images, as well as storing them efficiently.

According to the QIB to be developed, developers may find a specific dataset relevant. Thus, *Storage Backends* must provide services for browsing and selecting data that has been previously ingested, creating the aforementioned datasets. In addition, *Processing Backends* must be connected to the storage with the selected data. Then, users can perform advanced computational processes through *User Applications* (e.g. to execute AI algorithms through specific platforms such as Tensorflow) for the image analysis and feature extraction. The *Backends* must provide a space to manage user private data (e.g. AI models to be trained) and another space to share data among other users (e.g. data results obtained in the image analysis and feature extraction). Table 5 lists the main functional requirements for the *Storage Backend*, the related stages and the existing components to fit them.

Table 1 shows the main functional requirements (from F.08 to F.12) for the *Storage Backends* and the ESR guidelines stages related, and Table 2 shows the main non-functional requirements (from NF.08 to NF.11).

3.4. Processing Backends

In the ESR life cycle (Image Acquisition and Preparation for Analysis, and Image analysis and feature extraction stages), there are processes that require *Processing Backends*. When users carry out these processes, they must launch and run specific *User Applications* (e.g. AI frameworks, Radiomic Python libraries, etc.) to compute data hosted at *Storage Backends*. Depending on the *User Application*, the *Processing Backend* will provide the required computational infrastructure (e.g. GPUs, MPI Cluster) and other capabilities to launch and access those applications. Also, it must provide an efficient access for to the *User Applications* launched to the data in the *Storage Backends*.

In addition, during the Biomarker Validation Stage Biomarker tools will be deployed to carry out the proof of principle and effectiveness evaluation stages.

Table 1 shows the main functional requirements (from F.13 and F.14) for the *Processing Backends* and the related ESR guidelines stages, and Table 2 shows the main non-functional requirements (from NF.12 to NF.14).

3.5. User Applications

As it has been described in previous subsections, actors involved in the QIB development need to run *User Applications* for preparing or carrying out the ingestion and data analysis in different stages (e.g. noise filtering, segmentation, registration, normalisation, AI methods based on Machine/Deep Learning algorithms, etc.). A *User Application* is an application and its associated software libraries and specific computing requirements (e.g. GPUS, MPI CLuster) needed to run. *User Applications* provide interfaces to the users for interacting with them.

Table 1 shows the main functional requirements (from F.15 and F.F17) for the *User Applications* and the related ESR guidelines stages, and Table 2 shows the main non-functional requirements (from NF.15 to NF.17).

Table 2
List of main non-functional requirements.

COD	Description	Components
NF.01	User Identity must be unique.	IdPs supported by EGI-check-In
NF.02	The credentials to access *Storage and Processing Backends* must be unique. The users always use the same credentials to perform the actions independently of the accessed Backend.	EGI-check-in, Kubeauthoriser, EGI Data Hub
NF.03	*Backend* deployment must be cloud-provider agnostic regarding.	Infrastructure Manager
NF.04	*Backend* deployment must be reproducible.	Infrastructure Manager
NF.05	*Backend* deployment must be scalable in terms of storage and processing capacity.	CLUES, Infrastructure Manager
NF.06	*Processing Backend* Deployment must offer specific computing (e.g. GPU) and storage infrastructure to execute *User Applications*.	Infrastructure Manager, Cloud provider
NF.07	A *Backend* Deployment must support a minimum of one VO.	Kubeauthoriser, EGI Data Hub
NF.08	*Storage Backends* must provide the capacity to manage all necessary data (images and associated data).	MongoDB, POSIX File System (e.g. Ceph, Azure File System)
NF.09	*Storage Backends* must provide efficient data access to *Processing Backends*.	EGI Datahub (OneProviders)
NF.10	*Storage Backends* must restrict access only to VO users with certain roles.	EGI Datahub
NF.11	*Storage Backends* must provide efficient access to *Processing Backends* belonging to the same VO.	EGI Datahub
NF.12	*Processing Backends* must provide the computing infrastructure to execute *User Applications* efficiently.	Infrastructure Manager, Cloud Providers, HPC-Connector
NF.13	*Processing Backends* must access data efficiently from *Storage Backends*.	EGI Datahub
NF.14	*Processing Backends* must provide access to Storage Backends belonging to the same VO.	EGI Datahub, Kubeauthoriser
NF.15	*User Applications* must embed specific software and libraries.	Docker Hub Registry, Docker Images
NF.16	*User Applications* must execute their processes in *Processing Backends* which provide specific computing infrastructure.	Kubernetes
NF.17	*User Applications* must employ Storage and Processing Backends belonging to the same user VO	Kube-authoriser, EGI Datahub (OneClient)

The most common *User Application* topologies in the ESR Development workflow are the following:

3.5.1. Batch Job

This type of *User Application* is mainly used for processing data hosted in a *Storage Backend*. It runs unattended and typically reads information from a set of files (e.g. DICOM images or clinical data) and produces files as output. Additionally, a Batch Job may receive other input parameters and even show some output in the console for monitoring. It can be, among others, a Machine learning training process, the co-registration of a set of files, or the inference of a pre-trained AI model on a new data file.

This type of *Applications* requires a computing infrastructure provided by *Processing Backends* which could have additional requirements (e.g. GPUs, MPI Cluster).

3.5.2. HTC Batch Application

As batch Jobs, High-Throughput Computing (HTC) jobs are mainly intended for processing data hosted in a *Storage Backend*. It executes a set of jobs (using the same executable) with different input arguments from a defined set. A basic example could be the application of a filter to all the files in a directory.

This type of application also requires specific computing resources.

3.5.3. Interactive

It exposes an access interface to enable users to interact with the application. This interaction determines the behaviour of the application. Therefore, Interactive Applications show three main differences with respect to Batch Applications: (a) it will run for an undetermined time and eventually longer than a batch job; (b) It should start shortly, although it may trigger long-lasting actions; (c) It must be accessible by the user. Therefore, we consider web-based applications as a canonical example of an interactive application (e.g. A Jupyter Notebook).

4. Architecture

The section presents the architecture of the platform, highlighting the main components and the integration work carried out to cover the requirements mentioned above. Fig. 2 depicts the federated and distributed architecture of the *Storage and Processing Backends* for cancer images. It includes the five main areas outlined in the following subsections. Tables 1 and 2 list functional and non-functional requirements for these areas and the components selected or implemented to match them.

4.1. Federated users (authentication and authorisation)

The architecture uses VOs to manage the authorisation of users in QIBs development Communities. The authentication is managed through a federation of *Identity Providers* (IdPs). A VO is a collection of user identities authenticated from a collection of trusted IdPs which is used to collectively manage access permissions to the resources (e.g. *Processing and Storage Backends* that bind to this VO. The VO model, as used in business [14] or other science disciplines [15] fits the QIB development. VOs allow us to federate IdPs for both authentication and user authorisation conveniently, so users can access the *Processing and Storage Backends* associated with the VO as a whole with minimal administration intervention. The architecture employs the EGI-Check-in

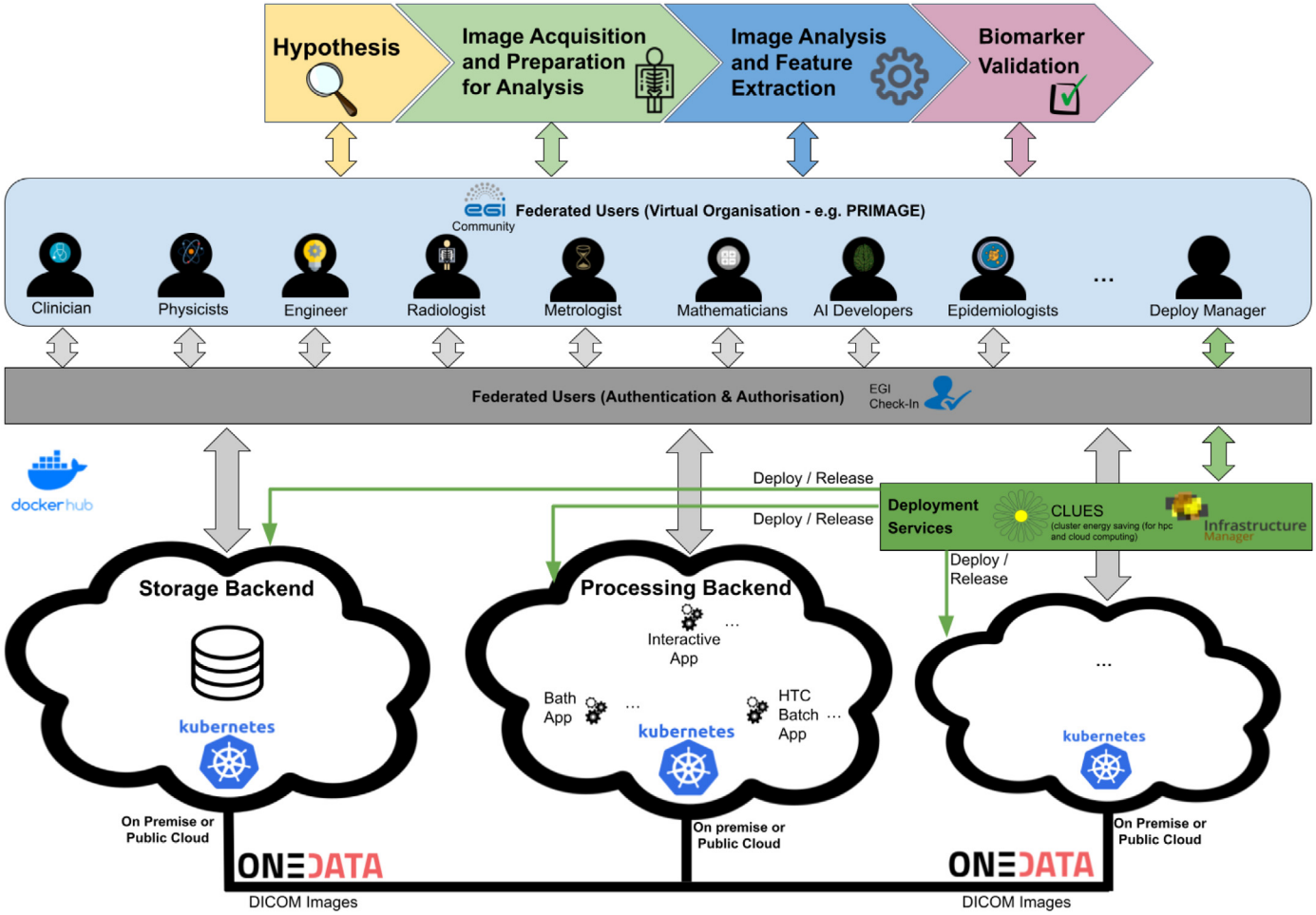

Fig. 2. Overview of the federated and distributed backends to implement ESR Guidelines.

Service[1] to implement the authentication processes and relies on the VOs for authorisation. EGI Check-in is a European Open Science Cloud (EOSC)[2] service provided by the EGI Community[3] for user authentication. It allows sign-up and sign-in users in the EGI community through the most common Identity Providers (IdPs) (e.g. Google, linked-in, EduGain, etc.) and manages VOs to organise the users for enabling access to associated resources (e.g. Storage and Processing Backends).

Storage Backends implement the authorisation through EGI Data Hub [16], QUIBIM Precision[4] and DCM4CHEE[5] (see Section 4.1.2) depending on the execution functionalities. The *Processing Backends* implement them through Kube-authorizer[6] (see Section 4.1.3). The component gets unique user identities and associated metadata (e.g. belonging to VOs, capabilities and/or roles) from supported IdPs and restricts access based on metadata gathered.

4.1.1. Backends deployment

The proposed architecture employs a Kubernetes (K8s)[7] system to: (a) run all *User Applications* required by both *Storage and Processing Backends* that offer functionalities for performing user actions; (b) execute *User Applications* in the *Processing Backends* for preparing or carrying out data analysis (e.g. noise filtering, segmentation, registration, normalisation, AI methods based on Machine/Deep Learning algorithms, etc.); (c) execute Biomarker Tools as *User Applications* in the *Processing Backends* for carrying out biomarker validations.

Infrastructure Manager (IM) is used to deploy and manage the cloud infrastructure. IM allows to deploy computing infrastructures in public and on-premises cloud providers. IM is platform-agnostic through deployment recipes written in *Resource and Application Description Language* (RADL).[8] IM relies in CLUES[9] service to automatically manage the horizontal elasticity of the K8s nodes. Therefore, we have built a set of RADL,[10] recipes to deploy on cloud infrastructures (private or public) required *Backends* (software, hardware and configuration) for running *User Applications* as Kubernetes processes. Also, depending on the *User Application* to run in the *Processing Backends* the recipes include the required computational resources (e.g. MPI clusters, GPUs, et.) and software (e.g. Tensorflow Framework, Pytorche, Python etc.). These recipes reflect the integration works carried out in the *Backends* because they define the required configuration of the hardware infrastructure and software employed for their implementations.

1 EGI Check-in: https://www.egi.eu/services/check-in.
2 European Open Science Cloud:https://eosc-portal.eu.
3 EGI Community:https://www.egi.eu/tag/community.
4 QUIBIM Precision:https://quibim.com.
5 DICOM Archive:https://github.com/dcm4che.
6 https://gitlab.com/primageproject/kube-authorizer
7 Kubernetes:https://kubernetes.io.
8 RADL:https://imdocs.readthedocs.io/en/latest/radl.html.
9 Cluster Energy Saving:www.grycap.upv.es/clues.
10 https://gitlab.com/primageproject/deployment/-/tree/main/RADLs

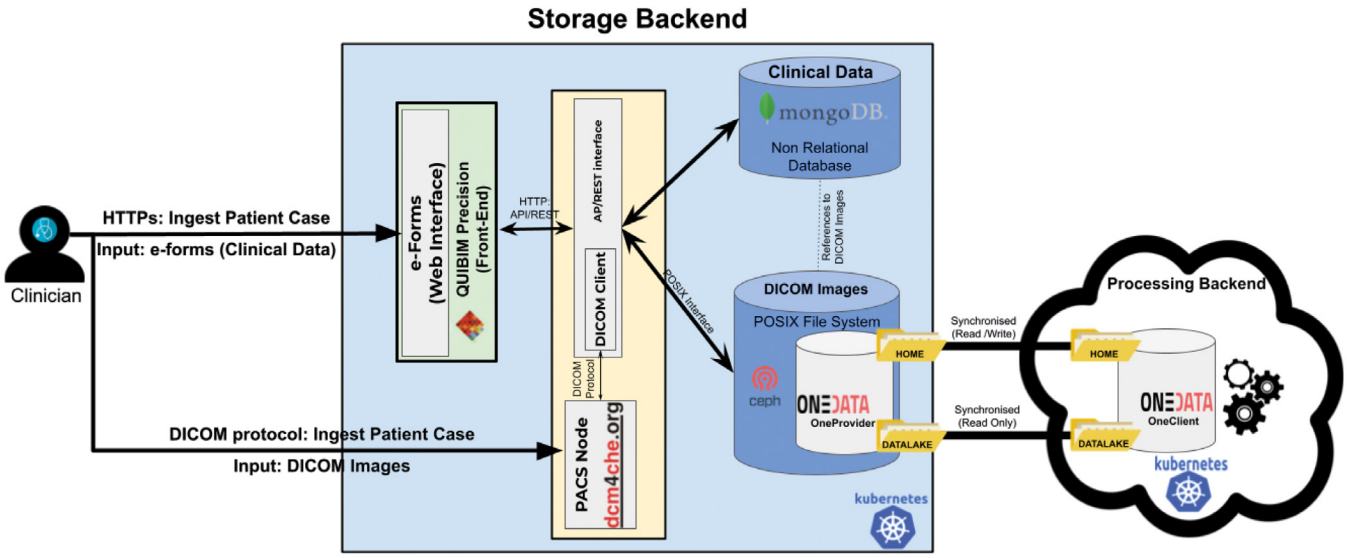

Fig. 3. Storage Backend Big picture.

4.1.2. Storage Backends

Fig. 3 shows the big picture and integrates all components in Storage Backends. The proposed architecture uses QUIBIM Precision[11] to implement the functionalities of ingesting, browsing, selecting and managing data. It offers a user-friendly web interface (front-end) to collect data via federated e-forms and uploads the associated medical images through a PACS node using the DICOM protocol. In the architecture, the PACS Node is implemented with DCM4CHEE, an open-source Image Archive. QUIBIM Precision organises the ingested data by patient cases and stores them in two different data containers. One is a POSIX file system that stores DICOM images. In on-premise deployments, the architecture employs a Ceph Distributed Storage System [17], an open-source distributed object storage for large amounts of data designed to deal with Big Data processing. In public cloud deployments, the architecture integrates the compatible POSIX file systems offered by the Cloud Providers (e.g. Azure File Systems, Google Filestore, etc.). The second database is MongoDB, which saves all clinical variables collected by the e-forms as JSON Files linking to corresponding DICOM images stored in the POSIX file system for each patient case.

In addition, QUIBIM Precision provides an interface to explore all stored patient cases and choose the most interesting cases (datasets) for developing purposes. The data selected (dataset) is defined as a JSON file that contains all the identifiers of the DICOM images and the selected e-forms (clinical variables and references to images).

The architecture uses different components to efficiently connect data among *Storage and Processing Backends* depending on the type of data (images or associated data collected by e-forms). Images are heavy to access or transfer efficiently, so the proposed architecture must host only the data to be processed where the process is carried out (*Processing Backends*). For that, images are replicated and synchronised among Backends through EGI Data Hub [16]. It is an EOSC service implemented through OneData[12] technology. This service allows the creation of isolated Data Volumes associated with a VO whose content is replicated and synchronised through OneProvider services. Users access Data volumes through Oneclient Services, which allows mounting the Data Volumes as local resources. Thus, each VO must create a volume named DATALAKE and install and configure a OneProvider Service in all sites where images will be replicated and synchronised (read-only). It is convenient to install the Oneprovider close to the *Processing Backends* to speed-up the access from *User Applications*, using the `direct--io` mode if possible. EGI Data Hub supports EGI-Check-in Service for the authentication and authorisation, restricting access to the Storage and Processing Backends belonging to a given VO.

Data collected in e-forms (clinical data) are transferred through JSON Files and do not require replication. They are exported as compressed text files that which only require a few Kbytes or Mbytes, which can transfer efficiently from the MongoDB database through a REST API.

Besides, the architecture creates another Volume named `HOME`, which provides the users with both a private and a folder shared among the users.

4.1.3. Processing Backends

Fig. 4 shows the overview and integrates all components in *Processing Backends*. The proposed architecture uses Kubernetes Dashboard[13] to implement the functionalities to launch *User Applications* defined through YAML files and Docker Containers (see Section 4.1.4).

Regarding accessing the data, *Processing Backends* use OneData to replicate and synchronise images from distributed *Storage Backends*. However, the data from e-forms is preserved on the MongoDB database and it is automatically downloaded when needed.

Interacting with Kubernetes through command line or REST API can be cumbersome for inexperienced users, so the platform includes a Kubernetes Dashboard to ease the deployment of *User Applications*. As Kubernetes Dashboard does not integrate OIDC, we implement it using EGI-Check-in. A proxy captures the requests and if the user belongs to the VO it redirects the traffic with the proper header to the Kubernetes Dashboard. To do this, it is necessary that: (1) Kubernetes is configured to allow OIDC (2) authorise the users through OIDC. We developed Kube-authoriser to authorise by creating an isolated execution environment for each user in K8s, called namespaces, and applying RBAC policies

11 QUIBIM Precision: https://quibim.com.

12 Global Data Access Solution for Science: https://onedata.org.

13 Kubernetes Dashboard: https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard.

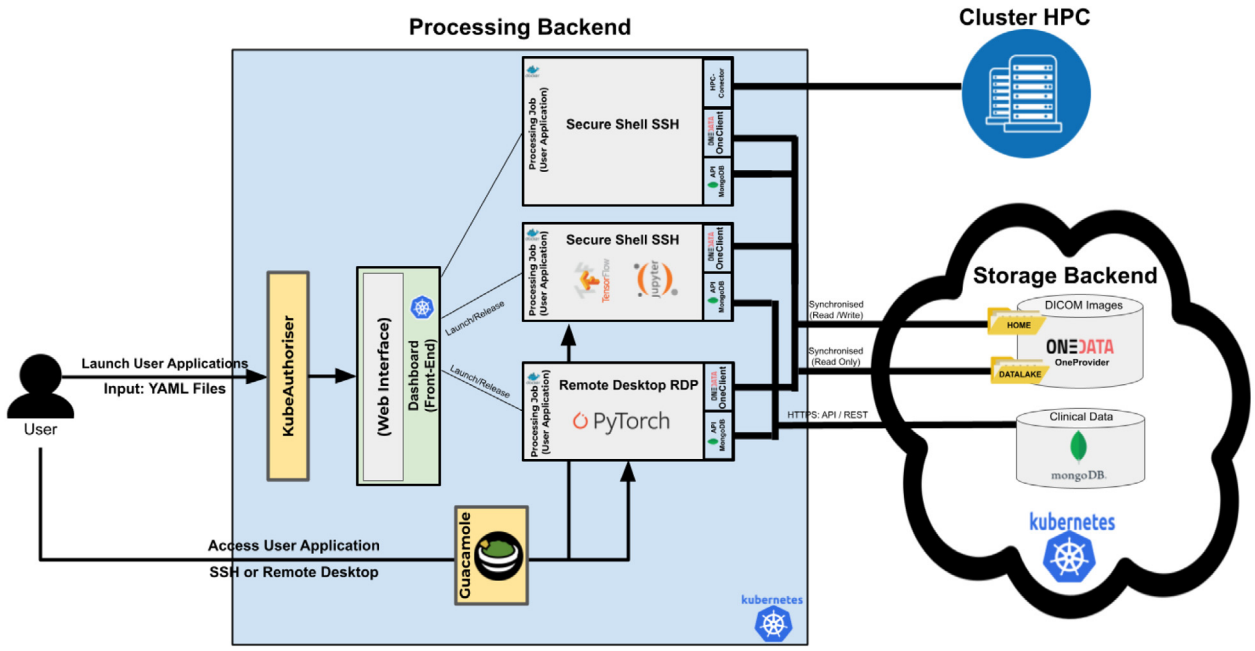

Fig. 4. Processing Backend Big picture.

and other actions such as the creation of the personal directory (in the HOME volume). These actions are automatically triggered at Kube-authoriser because the proxy of the Kubernetes Dashboard notifies Kube-authoriser every time a user is authenticated and belongs to the VO.

Users can access the *User Applications* deployed directly through Kubernetes Dashboard or Apache Guacamole.[14] It is a clientless remote desktop gateway and supports standard protocols like Remote Desktop (RDP) and SSH.

4.1.4. User applications

In the context of this work, a *User Application* is a program that is built following a specific design and implementation to run on the *Processing and Storage Backends* described above.

A *User Application* is composed of the following elements:

- **Application's Executable.** It will consist of an executable (or bundle of them) implemented by user developers or an existent component, it can be used through Command Line Interface (CLI) or Graphical User Interface (this way it is necessary to connect using Guacamole).
- **Docker Container.** Commonly the Application's Executable requires some dependencies, configuration files and tools that define the execution environment of the *User Application*.
  The proposed architecture uses Docker[15] and Docker Hub[16] to implement and manage these execution environments. Docker container images are a lightweight, standalone, executable package of software that includes everything needed to run an Application's Executable. Docker Hub is a registry of Docker images and allows organisations their management.
  To enable accessing the data all docker containers employed to build *User Applications* have to embed OneClient and MongoDB client libraries to allow efficient access to *Storage Backend* that contains both the images and the associated clinical data. OneClient allows mounting the `DATALAKE` and `HOME` OneData volumes as a data local volume. Both volumes were created through EGI data hub Service. By mounting the `DATALAKE` and the `HOME` volumes, Application's Executable has direct access to the images directory and the user's shared directory, through a POSIX interface. Furthermore, REST/API MongoDB client allows downloading the e-form data from the MongoDB database in the *Storage Backends*. Besides, the containers for Application's Executables requiring HPC resources include an instance of HPC-Connector [18]. This component is an open-source tool that manages the full life cycle of jobs in HPC infrastructures, seamlessly from the cloud job scheduler interacting with the workload manager of the HPC system.
- **Application Template.** In order to run a *User Application* in the *Processing and Storage Backends*, developers need a Kubernetes application template. Such templates can be described using JSON or YAML. An Application Template includes several Kubernetes objects such as jobs, pods, services, statefulset, configmaps, to implement the application topology. We distinguish three topologies (Batch jobs, HTC Batch Applications and Interactive Applications). As a result of this work, we provide a set of application templates (YAML files[17]) to build and run applications in the Backends based on these topologies. In the specific case of HTC topology and despite Kubernetes provides a "job" object, it does not fulfil the requirements of a batch queue jobs. Indicating different arguments for different Kubernetes job instances is not trivial and scheduling is performed within the multiple instances of a specific job. We can easily limit the number of concurrent instances of a job, but limit the allocation of resources for several competing jobs in a multi-tenant environment is not straightforward. However, this feature is demanded and an incubator project has recently been started.[18] Therefore, we have developed an object to support HTC batch job execution.

14 Apache Guacamole. https://guacamole.apache.org.
15 Docker: https://www.docker.com/.
16 Dockerhub: https://hub.docker.com/.
17 YAML templates: https://gitlab.com/primageproject/applications.
18 https://github.com/kubernetes-sigs/kueue

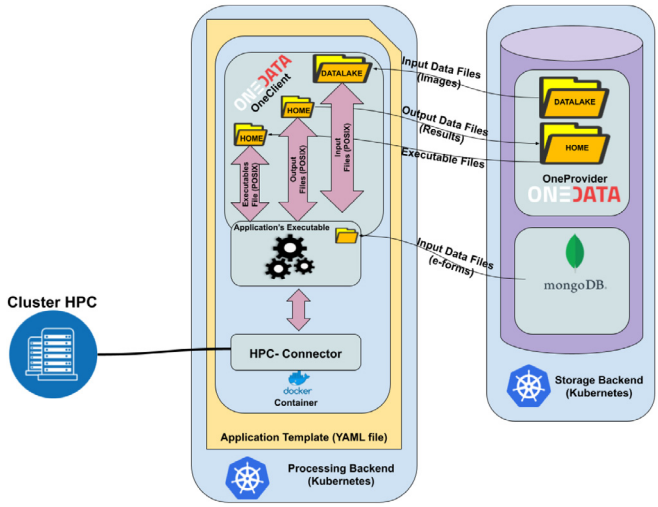

Fig. 5. Big picture of the Bath Job Architecture.

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: #batch_job_name#
  namespace: #namespace_name#
  labels:
    name: primage-batch-job
spec:
  template:
    metadata:
      name: primage-batch-job
    spec:
      containers:
      - name: primage-batch-job
        image:  #docker_image#
        imagePullPolicy: Always
        args: ["#command_line_arguments#"]
        securityContext:
          privileged: true
        env:
        - name: ONECLIENT_ACCESS_TOKEN
          valueFrom:
            secretKeyRef:
              name: access-primage-storage
              key: one-client-access-token
        - name: ONECLIENT_PROVIDER_HOST
          valueFrom:
            configMapKeyRef:
              name: primage-configmap
              key: oneprovider
        - name: MOUNT_POINT
          valueFrom:
            configMapKeyRef:
              name: primage-configmap
              key: primage.storage

        <<YOUR ENVIRONMENT VARIABLES>>

      restartPolicy: Never
  backoffLimit: 2
```

Fig. 6. YAML Template to build Batch Jobs.

4.1.5. Batch Jobs

Fig. 5 shows the architecture of the Batch jobs which comprises the three core components described above.

As a result of this work, we have created a YAML template and a procedure to create customised Batch jobs from the template.

In the first step, a developer builds the Application's Executable. Then, in the second step, the developer chooses a Docker container image to run the Application's Executable. As a result of this work we have created a set of Dockerfiles files[19] with OneClient, MongoDB libraries and HPC-Connector embedded to be used as a basis for customised Docker images. All new images generated from these Dockerfiles have to be registered in a Docker registry to be accessible by the nodes.

Next, in the third step, the YAML template is updated with the job-specific information by providing the information for the next parameters (see left Fig. 6):

- **#batch_job_name#.** This is the name of the Batch Job instance in Kubernetes.
- **#namespace_name#.** This is the name of the Kubernetes namespace of the Processing Backend where the jobs will run. The platform automatically creates a user-specific namespace in the first access to the platform. The Batch job will only be accessible from the owner of this namespace.
- **#docker_image#.** This is the name of the Docker image from the Docker Registry.

19 https://gitlab.com/primageproject/containers/-/tree/master/ubuntu18

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: backend-#NAME#
  namespace: serlohu-at-upv-dot-es
data:
  name: #NAME#
  configuration: |
    {
      "ENDPOINT": #ENDPOINT#
      "PROXY": #CREDENTIALS#
    }
```

Fig. 7. YAML file to configure HPC-Connector.

- **#command_line_arguments#.** These are the command line arguments needed to run the Application's Executable.

The YAML template includes a set of fixed and predefined environment variables (see the right side of Fig. 6) that have been automatically created in the user registration process. These are the following:

- **ONECLIENT_ACCESS_TOKEN.** This is the token to grant access to the *Storage Backend*.
- **ONECLIENT_PROVIDER_HOST.** The Batch Job will connect to the *Storage Backend* through this host using the credentials provided by the token. It is interesting to connect to the geographically nearest Oneprovider to get the best performance.
- **MOUNT_POINT.** This is the root folder of the *Storage Backend* where the `DATALAKE` and `HOME` volumes are accessible.

In the case that the job requires an HPC infrastructure, an additional YAML file will contain the specific configuration for the HPC-Connector. The parameters to be configured (see Fig. 7) are the following:

- **#name#**. The name of the HPC resource (e.g. Prometheus).
- **#namespace_name#.** This is the name of the Kubernetes namespace in the *Processing Backend*. This configuration will be accessible only to the owner of this namespace.
- **#END_POINT#.** The endpoint od the HPC resource. (e.g. prometheus.cyfronet.pl)
- **#CREDENTIALS#.** These are the credentials needed to access the HPC resource. (e.g. In the case of Prometheus a proxy file is required).

After configuring HPC-Connector, the HPC job is created thought another YAML file (see Fig. 8. The parameters to be configured are the following:

- **#name#**. This is the name of the job.
- **#namespace_name#.** The name of the Kubernetes namespace in the *Processing Backend*.
- **#Script#.** A script that implements the HPC job with the additional configuration required by the backend. (e.g. In the case of Prometheus a `SCRATCH` file is required).

Then, the developer can run the customised Batch job on the Processing and Storage Backends, creating a new job with the YAML file through the Kubernetes Dashboard. Finally, in step 4, the developer can monitor the Batch Job running process through the Kubernetes Dashboard.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: #NAME_JOB#
  namespace: #NAMESPACE#
data:
  monitoring_period: "15"
  job: |
    {
      "host": #END_POINT#",
      "script": #SCRIPT#"
    }
```

Fig. 8. YAML file for configuring HPC Job.

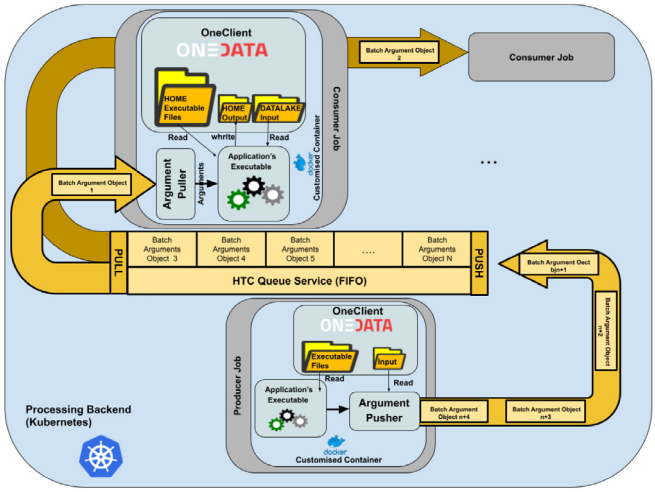

Fig. 9. Big picture of the HTC Batch Application Architecture.

4.1.6. HTC Batch Applications

Fig. 9 shows the three main elements of an HTC Batch Application. Its purpose is to execute an Application's Executable repeatedly, each execution with a different input argument from a defined set. The elements are one HTC Queue Service (Queue Service), one HTC Batch Producer (Producer Jobs) and a set of HTC Batch Consumer (Consumer Jobs).

The arguments of each Application's Executable are generated and pushed to the Queue Service by the Producer Job. Typically, it reads information from a set of files, although it may receive other types of input parameters, and generate the arguments. Then, it pushes the argument through the Argument Pusher. The Consumer Jobs retrieves iteratively a set of arguments from the Queue Service through an Argument Puller and execute repeatedly an Application's Executable (one execution per retrieved argument), which typically produces some kind of files or standard output. Both Producers and Consumers run as unattended Jobs on the Processing Backend. The mechanism to communicate them is through a FIFO (First In, First Out) queue provided by the Queue Service.

A basic example of a Producer Job could be a programme that reads all the files in a directory and puts their paths on the Queue Service. A Consumer Job iteratively pulls a filename from the Queue Service and executes repeatedly an Application's Executable as a Batch job that processes that file until the queue becomes empty.

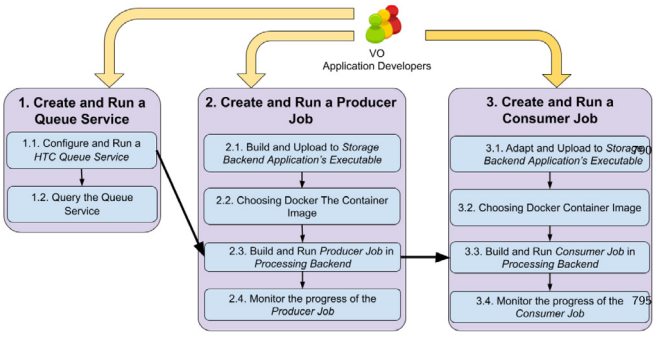

Fig. 10. Procedure to create HTC Batch applications.

As a result of this work, we have created a set of YAML templates and a procedure for the purpose to complete them and creating customised HTC Batch Applications by developers. Fig. 10 shows three branches in the procedure.

Each HTC Batch Application has to deploy and run a separate Service Queue (left branch in Fig. 10 - Step 1). This guarantees that each application has the same Quality of Service (e.g. the upper limit for the number of concurrent jobs running on the Processing Backend). In step 1.1, developers have to customise a YAML template indicating the name of the Queue Service (parameter #service name#) and then run the Queue Service on the Processing Backend, creating a new service employing the customised YAML file through the Kubernetes web interface. Then, in step 1.2., developers can consult the status of the queue (e.g. number of arguments in the queue). Then, developers create and run a customised Producer Job (centre branch — Step 2, in Fig. 10). Basically, the Producer Job is a Batch Job (see Section 4.1.5) with the specific purpose of generating parameters and pushing them to the Queue Service. For that, as a result of this work, we have prepared a specific YAML template for developers which is equal to the Batch Job Template (see Fig. 6), adding the name of the Queue Service (parameter #Name of HTC Batch Queue service#). Also, as a result of this work, we have created a Docker image with the Argument Pusher Component embedded to build new Docker images for Producer Jobs from it.

Next, developers create and run customised Consumer Jobs (right branch — Step 3, in Fig. 10). A Consumer Job is a Batch Job with the specific purpose of processing parameters pushed from the Queue Service. For that, as a result of this work, we have prepared a specific YAML template for developers which is equal to the Batch Job Template (see Fig. 6), adding the name of the Queue Service (parameter #Name of HTC Batch Queue service#) and the number of Consumer Job that will be running concurrently (parameter #parallelism#) which can be enlarged or shrunk during the execution of the HTC Batch Application. Also, as a result of this work, to build new docker images for Consumer Jobs we have created a docker image with the Argument Puller Component embedded to inherit in the docker containers employed by developers to build the consumers.

When all the jobs (consumer and producers) are completed, the whole HTC Batch Application can be terminated releasing the jobs (Queue Service, Producer Job and Consumer Jobs).

4.1.7. Interactive Application

Basically, the architecture of the Interactive Application is equal to the Batch application (see Section 4.1.5) but adds an interactive interface which allows ending users to interact with the application.

As a result of this work, we have created a YAML template and a procedure for the purpose to complete them and creating

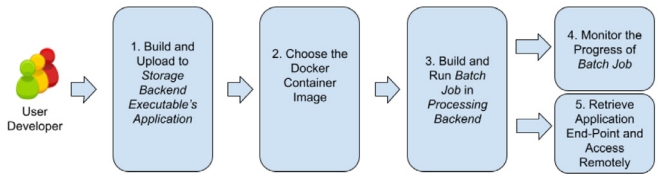

Fig. 11. Procedure to create Interactive applications.

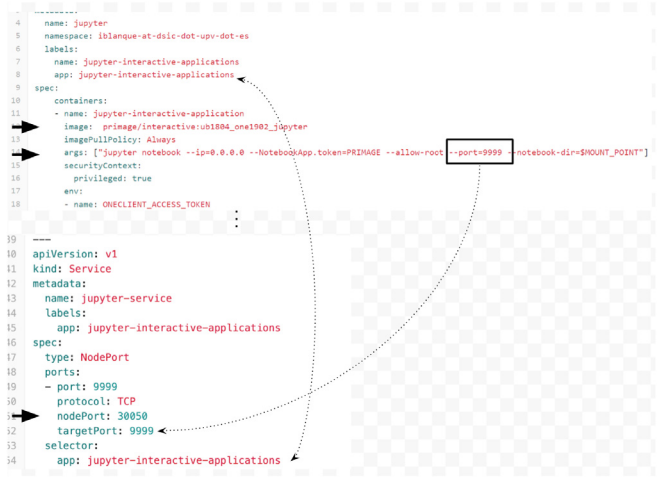

Fig. 12. Example YAML Template to build Interactive Applications.

customised Interactive Applications by developers. Fig. 11 shows the phases of the procedure.

Steps 1 to 3 of Fig. 11 are the same as those described in batch jobs.

In the second step, the developer chooses a Docker container image to run the Application's Executable. As a result of this work, we have created two base containers to build new Docker Images as execution environments: a Jupyter Notebook server and an apache NGINX web server. In the case of the Apache web server, developers can upload the full web application to the Storage Backend and reuse the Apache webserver container. Alternatively, developers can create a custom container with the application installed on it, but this will reduce the flexibility of the Docker Image. the two base containers have to embed Oneclient and MongoDB libraries.

Next, in the third step, the YAML template is customised. In this specification, there are three important details to consider regarding Batch applications (see Fig. 12):

- **#Targetport#**. The value of the targetPort should match the value of the port that the interactive application will use.
- **#nodePort key#**. The value of the port where the application will be accessible from outside.

5. Results and discussion

This section describes the deployment in the PRIMAGE project, where a QIB community uses the federated architecture for developing QIBs related to two paediatric cancers.

5.1. PRIMAGE Project deployment

As a result of this work, we present the deployment of the architecture (see Fig. 13) described above for giving support to the QIB community involved in the PRIMAGE Project [7].

PRIMAGE Project has created a VO in the EGI Community named `vo.primage.eu`. Furthermore, a `Service Level Agreement` (SLA) has been is made between EGI Foundation (the Service Provider) and vo.primage.eu (the Customer represented by the Universitat Politècnica de València - UPV) to define the provision and support of the services through the EGI federation. This Agreement is valid to 31/06/2025 and guarantees the operation (twenty-four hours a day and seven days a week) of the main services used, which are the EGI Check-In and the EGI Datahub.

The PRIMAGE VO involves more than 30 researchers from more than 10 European research institutions belonging to the PRIMAGE project Consortium.

Regarding the backend infrastructure, three storage backends have been deployed. These three storage backends manage the medical images and clinical data related to NB and DIPG tumours: An Azure File Service, which is the data source running a two file systems in UPV in Spain and Cyfronet in Poland). Medical images are automatically replicated on-demand from three sites to bring data closer to the computing resources. For that, a OneProvider Service has been installed and configured in each site giving support to a Volume of 3.6 TB for storing the medical images and another limited volume of 50 GB for the personal data. The volumes can be extended on demand up to 10 TB.

For the data ingestion process, QUIBIM Precision provides two specific e-forms to gather the clinical data. Currently, the total number of patient cases is 841 (783 of NB and 58 of DIPG) which are stored in the storage volume (1.4 TB) and the MongoDB for the clinical data.

In addition, a Processing Backend has been deployed on the cloud resources provided by the UPV, featuring 32 CPUs, 2 GPUs, 512 RAM, and 512 GB SSD Hard Disk. It hosts an elastic Kubernetes cluster which manages the resources on demand.

The DATALAKE volume is replicated in this Processing Backend through a OneProvider component. It is also replicated in Cyfronet, which hosts the supercomputer Prometheus. Prometheus is the 1st HPC system in Poland (475th on Top 500 in June 2022, 38th in June 2015). Furthermore, all Clinical Data hosted at Storage Backend are accessible from the Processing Backend through a REST API".

5.2. User Application validation

To validate the *User Application* typologies, a set of *User applications* has been implemented and executed employing the PRIMAGE *Backends*. The validation consists in the implementation of the whole workflow to train, validate and test a *Convolutional Neural Network* (CNN) for to predicting breast cancer in breast histology images employing the Keras deep learning library[20] .

The *User Applications* implemented are a *HTC Application* and a *Interactive Application* that use cloud resources provided by UPV, and a *Batch job* that use GPUs of Prometheus Cluster provided by Cyfronet. All these *user Applications* could be a typical applications required to implement the ESR Cycle's life to develop a QIB.

5.2.1. Dataset

The used dataset is related to *Invasive Ductal Carcinoma* (IDC), the most common of all breast cancer. The dataset was originally curated [19,20] and is available in public domain.[21]

It has a total of 277.524 patches of 50 × 50 pixels from images (see Fig. 14), including 198.738 negative examples (i.e., no breast cancer) and 78.786 positive examples (i.e., breast cancer found in the patch).

20 https://pyimagesearch.com/2019/02/18/breast-cancer-classification-with-keras-and-deep-learning/

21 https://www.kaggle.com/datasets/paultimothymooney/breast-histopathology-images

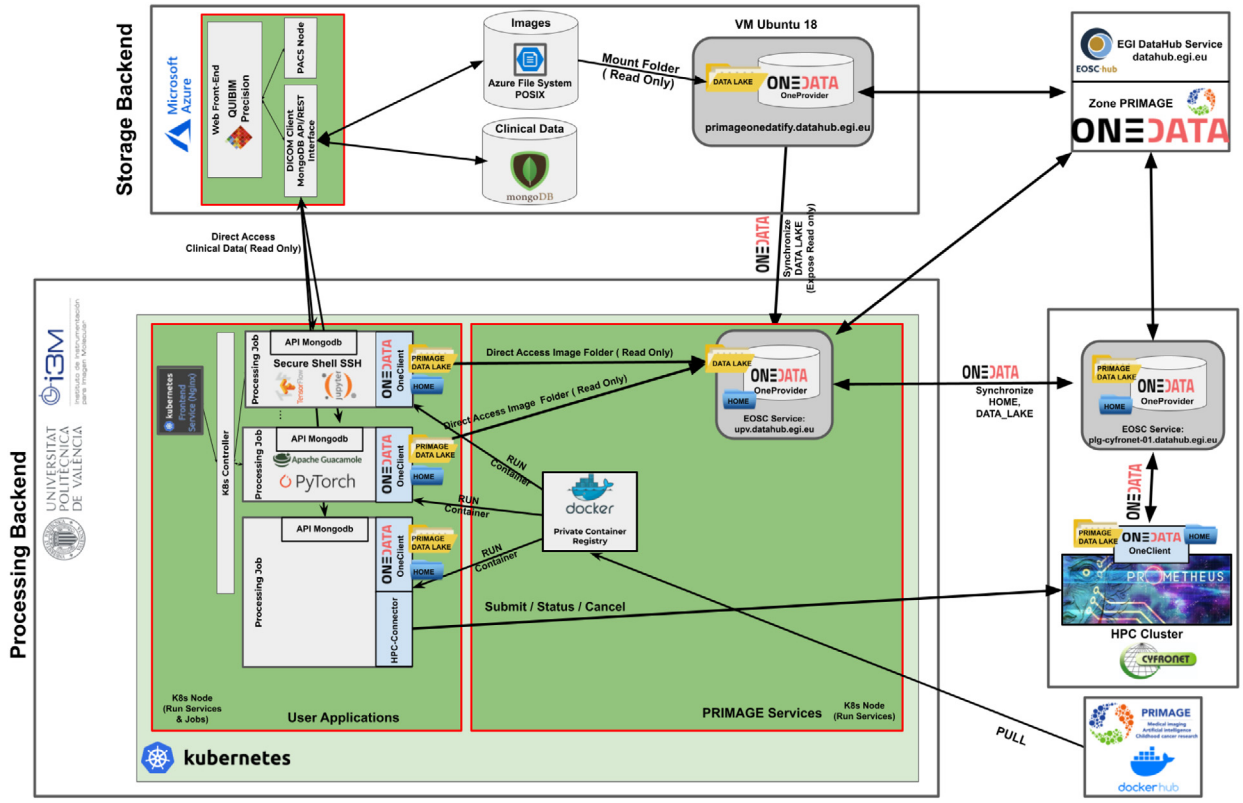

Fig. 13. PRIMAGE Project Deployment for the QIB community.

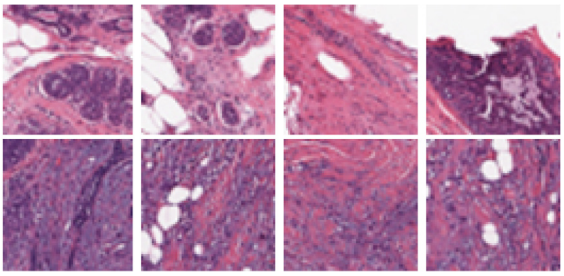

Fig. 14. Eight patches of 50 × 50 pixels extracted from Breast Histopathology. The four images at the top of the figure are negative examples, and the four at the bottom are positive.

All these images were available from `DATALAKE` Volume in the deployed *Storage Backend*. Images were synchronised at the UPV and Cyfronet by the Oneproviders so they are accessible to all *User Applications* running in the cloud resources provided by the UPV and the HPC resources provided by Cyfronet.

5.2.2. Training and validation

A *User Application* is required to execute the training and validation of the AI Model. This is implemented thought a Keras Deep Learning classification architecture employing Tensorflow 2.0 framework and a set of specific libraries such as matplotlib (a scientific plotting package that is the de-facto standard for Python), Keras or Numpy (a Python library for numerical processing) to name a few. Thus, a new *Batch Job*[22] has been built but in this case we decide to execute the training process in Prometheus, the HPC resource provided by Cyfronet.

To execute the training, the GPUs provided by Prometheus cluster are used in this *Batch job*. It uses HPC-Connector (YAML file[23]) to launch and monitoring the processes in the Prometheus GPUs.

Fig. 15 shows the evolution by iteration of the loss and the accuracy during the 5 epochs of the training. It shows a quick convergence to the final values, which justifies the reduced number of epochs used in the experiment.

5.2.3. Testing

When the training and validation is finished, a new *User Application*[24] is used to carry out the testing process of the trained AI-Model. This is a *HTC Batch Application* with a Producer Job and some Consumer Jobs. The Producer code (*htc-producer.py*) creates partitions over the whole testing images, writes down that partitions in CSV files and puts the path of that files in the Queue Service. Each Consumer (*htc-consumer.py*) gets a partition file path from the Queue Service and load the file as a *pandas dataframe* which can be used as input source for the prediction flow over the previously trained model, writing finally the array of predicted indexes in a file.

5.2.4. Visualisation of results

In order to combine and visualise the results of the previous testing process, finally another *User Application*[25] was deployed, this time an *Interactive Application* which is a Jupyter Notebook

[22] Batch job used to run the validation and the logs: https://gitlab.com/primageproject/deployment/-/tree/main/Validation/1-batch-job--training.

[23] https://gitlab.com/primageproject/deployment/-/blob/main/Validation/1-batch-job--training/hpc-connector-job-breasttumour.yaml

[24] HTC batch application used to do the testing process and the logs: https://gitlab.com/primageproject/deployment/-/tree/main/Validation/2-htc-application--testing.

[25] Interactive application used to show the results: https://gitlab.com/primageproject/deployment/-/tree/main/Validation/3-interactive-application--visualization-of-results

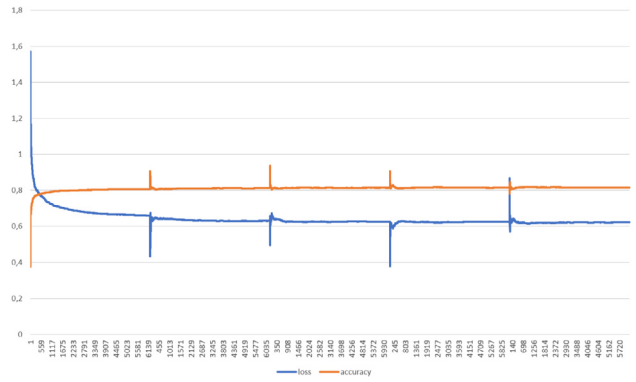

Fig. 15. Accuracy and Loss evolution by Iteration.

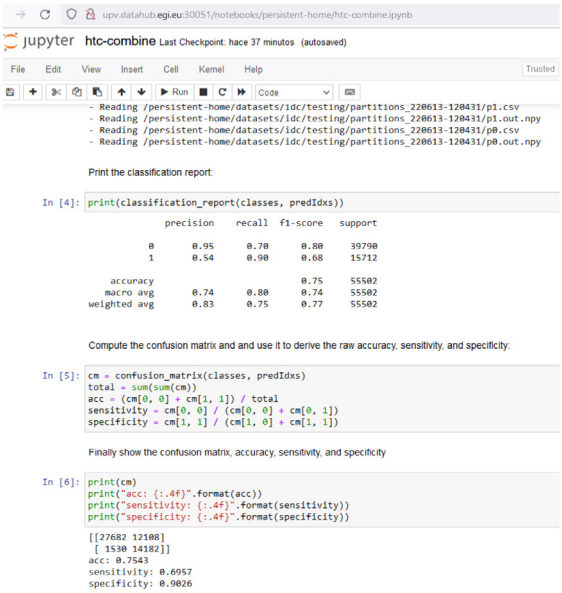

Fig. 16. Visualisation of results in a Jupyter Notebook.

web service. Accessing to the URL of that service the user can interactively execute some python code in a web page to read the output files of the previous HTC process, combine the predicted index arrays and present results as a plot or table (see Fig. 16).

5.2.5. Performance evaluation with respect to local environments

This section compares in terms of performance the execution of the test case in a local premises to the performance using the PRIMAGE Platform. The PRIMAGE platform eases the access to the shared data and the management of a workload with several jobs. This comes with a cost, as data should be distributed on demand and job orchestration will require managing the jobs. The main goal of the three User applications implemented is to validate the usability of the application models running on the PRIMAGE backend, so the execution time is short. Therefore, the overhead may seem quite high in comparison with the execution time. However, as it does not depend on the job execution time, the results are acceptable for most of the cases.

We instrumented the HTC Batch application to measure the overhead of the distributed storage for dealing with the input, output and executable files. We compared the execution of all the jobs in a local environment, the execution of all the jobs as a single Batch job and the execution of an HTC Batch Application splitting the jobs individually and distributing them across four consumers.

In the architecture, Onedata connects the storage and processing backends and provides on-demand synchronisation of the files. Onedata mounts remote volumes in the applications (Batch and HTC) file systems as if they were local data. In the first access to any remote file Onedata introduces an overhead due to the data caching to the processing backend and it is much more efficient to send one big file than many small files. Thus, all the jobs (Batch and HTC Batch Application) include an initial transference and extraction of the image package from the Storage Backend to the application local folder through Oneclient, avoiding transferring the files individually.

Table 3 shows the execution time for the different stages of the three experiments. The execution time obtained gives us an idea of the overhead of the creation of the container creation and transfer of files. Local execution (first column of Table 3) corresponds to a Kubernetes Pod deployed in the PRIMAGE processing backend with all medical images and binary executables hosted locally in a folder. In this case, the overhead introduced is only due to the creation of the container in kubernetes (3 s) and all data (images, executable files and results) are managed locally. This overhead is the same in all the experiments as it uniquely depends on Kubernetes.

Batch job time corresponds with the implementation of a user application following Section 4.1.5. In this case, data are managed by the PRIMAGE storage backend and synchronised by Onedata, introducing mainly an overhead due to the data transferred between backends (7 s). This overhead is produced because the Oneclient transfers images and executables files from the storage backend, and also copies from the processing backend to store the results. The execution time has been the same in the local case. Both cases use the same computational resources. HTC Bath time corresponds to one producer, one queue service and four consumers. We observe that the overhead introduced by file transfer is higher than the Batch Job (16 s instead of 7 secs). In this case there is an overhead introduced by the producer accessing the image and executable files to create the input arguments and put them into the queue. Also, consumers introduce overhead accessing images and executable files, and copying back the results. In this case, the execution time is lower than Batch Job because jobs are executed in parallel. It is foreseeable that longer jobs will show a better relation between the overhead and the execution time.

As a summary, the execution time shows a reasonable overhead with respect to the execution in a local environment, and the advantages of using the platform (on-site processing, no need to download data, synchronised heterogeneous computing resources) constitute an important benefit for the analysis of the data. Additionally, the use of virtual cloud backends facilitates scaling up and down the resources to fit the workload requirements. This is implemented through the automatic scalability offered by CLUES.

5.3. PRIMAGE User Applications

Supported by different PRIMAGE project partners, users have developed using the approaches proposed in this work, a set of *User Applications* to implement the ESR workflow stages for biomarkers in NB and DIPG cancers. All these applications demonstrate the feasibility of the method proposed in the article.

5.3.1. Interactive applications

University of Konstanz (UKON) has developed interactive applications [21–23] based on the large-scale multivariate database hosted in the deployed Storage Backend and running on Processing Backend. The applications provide an interactive, visual data exploration interface that facilities experts to get an overview

Table 3
Execution time (in seconds).

	Local	Batch job	HTC Batch job
Container Creation	3	3	3 (each consumer in parallel)
File Transferences	–	7	16
Queue Service Creation	–	–	5
Execution Time	140	140	43
Total	143	150	64

of the data. The proposed interface is designed to display large amounts of data to enable clinical experts to make sense of the displayed data at an overview level. This allows the medical expert to integrate their domain-knowledge into a search query, therefore achieving more accurate results. For example, a pattern consists of a combination of biomarkers which reflects generic disease profiles, the similarity search facilitates confirming or denying hypothesis by testing suitable patterns. In addition, related patients are arranged using re-ordering and clustering algorithms.

5.3.2. Batch jobs and HTC Batch Applications

HULAFE has developed a set of Batch Jobs and HTC Batch Applications. All these applications have run on the Processing Backend deployed. As a relevant result is novel registration pipelines for NB tumours [24]. The study was carried out with a dataset (patient cases) hosted at Storage Backend. Applications were developed to optimise and select the best generic registration parameters. Furthermore, the results of this study were the baseline to develop a new registration pipeline for DIPG. To improve the accuracy of these registration pipelines, AI-based solutions are being explored by using the VoxelMorph library and new *User Applications* in the Processing Backend will be created. Furthermore, HULAFE used the Processing platform to evaluate the performance of 5 denoising filters [25] anisotropic diffusion filter (ADF), curvature flow filter (CFF), Gaussian filter (GF), non-local means filter (NLMF), and unbiased non-local means (UNLMF) in T2-weighted MR images of paediatric patients with NB.

In addition, Batch Jobs and HTC Batch Applications have been developed by HULAFE, QUIBIM and UPV partners to explore a novel methodology (Fit-Cluster-Fit) [26] based on confidence habitats and voxel uncertainty to improve the power of the apparent diffusion coefficient to discriminate between benign and malignant neuroblastic tumour profiles in children. This method improves the classification performance of imaging biomarkers for paediatric solid tumour cancers and it can be adapted to evaluate any dynamic tumour signal.

6. Conclusions

This work describes the design, implementation and validation of a software architecture to support the development and application of Quantitative Image Biomarkers. It implement a federated model to synchronise data among the different storage backends linked to different processing environments, including both Cloud and HPC resources. The architecture provides a federated Authentication and Authorisation Infrastructure based on Virtual Organisations that provide coherent and scalable authorisation management across the different providers. The processing backend is supported by a Kubernetes container management platform that runs the platform services and customised applications. The architecture is the outcome of a requirement elicitation process and uses mainstream, widely available components.

The architecture uses the abstraction of the batch, High-Throughput-Compute, High-Performance computing and Interactive jobs to provide a simplified framework to develop applications with POSIX access to the distributed storage backends. HPC Jobs are managed through a mirror Kubernetes job that interacts with HPC batch queues to provide a seamless and coherent environment.

The architecture has been instantiated for the case of the PRIMAGE project, providing support for the storage and processing of paediatric cancer data in Neuroblastoma and DPIG, facilitating the development of applications on top of the platform. The article includes a validation of the HPC and HTC cases to show the customisation and instantiation of the jobs. The components developed are mainly released under open-source licences.

CRediT authorship contribution statement

J. Damián Segrelles Quilis: Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Project administration. **Sergio López-Huguet:** Conceptualization, Investigation, Software, Resources, Writing – review & editing. **Pau Lozano:** Conceptualization, Investigation, Validation, Software. **Ignacio Blanquer:** Conceptualization, Writing – review & editing, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data used is public and is referenced in the article.

Acknowledgements

The work presented in this paper has been partially funded by the project PRIMAGE (PRedictive In-silico Multiscale Analytics to support cancer persona-lised diaGnosis and prognosis, empowered by imaging biomarkers) BusinessPlace is a Horizon 2020, Research and Innovation Actions (RIA) (Topic SC1-DTH-07-2018) with grant agreement no: 826494. This research was supported in part by PLGrid Infrastructure.

References

[1] L. Martí-Bonmatí, A. Alberich-Bayarri, Imaging Biomarkers: Development and Clinical Integration, Springer, 2016, http://dx.doi.org/10.1007/978-3-319-43504-6.
[2] R.J. Gillies, P.E. Kinahan, H. Hricak, Radiomics: Images are more than pictures, they are data, Radiology 278 (2) (2016) 563–577.
[3] A. Alberich-Bayarri, E. Neri, L. Martí-Bonmatí, Imaging biomarkers and imaging biobanks, in: Artificial Intelligence in Medical Imaging, Springer, 2019, pp. 119–126, http://dx.doi.org/10.1007/978-3-319-94878-2.
[4] R. Forghani, P. Savadjiev, A. Chatterjee, N. Muthukrishnan, C. Reinhold, B. Forghani, Radiomics and artificial intelligence for biomarker and prediction model development in oncology, Comp. Struct. Biotech. J. 17 (2019) 995.
[5] M.A. Talib, S. Majzoub, Q. Nasir, D. Jamal, A systematic literature review on hardware implementation of artificial intelligence algorithms, J. Supercomput. 77 (2) (2021) 1897–1938.

[6] European Society of Radiology (ESR) communications@ myESR org, ESR statement on the stepwise development of imaging biomarkers, Insights Into Imaging 4 (2013) 147–152.
[7] L. Martí-Bonmatí, Á. Alberich-Bayarri, R. Ladenstein, I. Blanquer, J.D. Segrelles, L. Cerdá-Alberich, P. Gkontra, B. Hero, J. García-Aznar, D. Keim, et al., PRIMAGE project: Predictive in silico multiscale analytics to support childhood cancer personalised evaluation empowered by imaging biomarkers, Eur. Radiol. Exp. 4 (1) (2020) 1–11.
[8] L.M. Bonmatí, A. Miguel, A. Suárez, M. Aznar, J.P. Beregi, L. Fournier, E. Neri, A. Laghi, M. França, et al., CHAIMELEON project: Creation of a pan-European repository of health imaging data for the development of AI-powered cancer management tools, Front. Oncol. 12 (742701) (2022) 515.
[9] J. Zhang, Q. Qiu, J. Duan, G. Gong, Q. Jiang, G. Sun, Y. Yin, Variability of radiomic features extracted from multi-b-value diffusion-weighted images in hepatocellular carcinoma, Transl. Cancer Res. 8 (1) (2019) 130.
[10] H. Zhu, Y. Ai, J. Zhang, J. Zhang, J. Jin, C. Xie, X. Jin, Preoperative nomogram for differentiation of histological subtypes in ovarian cancer based on computed tomography radiomics, Front. Oncol. 11 (2021) 852.
[11] F. Valdora, N. Houssami, F. Rossi, M. Calabrese, A.S. Tagliafico, Rapid review: Radiomics and breast cancer, Breast Cancer Res. Treat. 169 (2) (2018) 217–229.
[12] P. Papadimitroulas, L. Brocki, N.C. Chung, W. Marchadour, F. Vermet, L. Gaubert, et al., Artificial intelligence: Deep learning in oncological radiomics and challenges of interpretability and data harmonization, Phys. Med. 83 (1) (2021) 108–121.
[13] A. Pomar-Nadal, C. Pérez-Castillo, A. Alberich-Bayarri, G. García-Martí, L. Martí-Bonmatí, et al., Integrating information about imaging biomarkers into structured radiology reports, Radiologia 55 (3) (2013) 188–194.
[14] D. Dziembek, P. Bajdor, Concept of information strategy of virtual organization with using the cloud computing solutions, in: IT Systems and Decisions in Business and Industry Practice, in: Applied Mechanics and Materials, vol. 795, Trans Tech Publications Ltd, 2015, pp. 61–68, http://dx.doi.org/10.4028/www.scientific.net/AMM.795.61.
[15] A. Wiggins, K. Crowston, Developing a conceptual model of virtual organisations for citizen science, Intl. J. Org. Des. Eng. 1 (1–2) (2010) 148–162.
[16] M. Viljoen, Ł. Dutka, B. Kryza, Y. Chen, Towards European open science commons: The EGI open data platform and the EGI datahub, Procedia Comput. Sci. 97 (2016) 148–152.
[17] A. Aghayev, S. Weil, M. Kuchnik, M. Nelson, G.R. Ganger, G. Amvrosiadis, File systems unfit as distributed storage backends: Lessons from 10 years of Ceph evolution, in: Proc. of the 27th ACM Symp. on Operating Systems Principles, 2019, pp. 353–369, http://dx.doi.org/10.1145/3341301.3359656.
[18] S. López-Huguet, J.D. Segrelles, M. Kasztelnik, M. Bubak, I. Blanquer, Seamlessly managing HPC workloads through kubernetes, in: International Conference on High Performance Computing, Springer, 2020, pp. 310–320, http://dx.doi.org/10.1007/978-3-030-59851-8.
[19] A. Janowczyk, A. Madabhushi, Deep learning for digital pathology image analysis: A comprehensive tutorial with selected use cases, J. Pathol. Inform. 7 (2016) http://dx.doi.org/10.4103/2153-3539.186902.
[20] A. Cruz-Roa, A. Basavanhally, F. González, H. Gilmore, M. Feldman, S. Ganesan, N. Shih, J. Tomaszewski, A. Madabhushi, Automatic detection of invasive ductal carcinoma in whole slide images with convolutional neural networks, in: Medical Imaging 2014: Digital Pathology, vol. 9041, SPIE, 2014, 904103, http://dx.doi.org/10.1117/12.2043872.
[21] T. Spinner, U. Schlegel, H. Schäfer, M. El-Assady, ExplAIner: A visual analytics framework for interactive and explainable machine learning, IEEE Trans. Vis. Comput. Graph. 26 (1) (2019) 1064–1074.
[22] J.F. Buchmüller, U. Schlegel, E. Cakmak, et al., SpatialRugs: A compact visualization of space and time for analyzing collective movement data, Comput. Graphics 101 (2021) 23–34, http://dx.doi.org/10.1016/j.cag.2021.08.003.
[23] U. Schlegel, D.A. Keim, Time series model attribution visualizations as explanations, in: 2021 IEEE Workshop on TRust and EXpertise in Visual Analytics, TREX, IEEE, 2021, pp. 27–31, http://dx.doi.org/10.1109/TREX53765.2021.00010.
[24] L. Cerdá Alberich, V. Canuto, Fully automated segmentation of neuroblastic tumours on multisequence MRI using convolutional neural networks, in: EMJ, Vol. 2, no. 1, 2021, pp. 24–26.
[25] M. Fernández Patón, L. Cerdá Alberich, C. Sangüesa Nebot, B. Martinez de Las Heras, D. Veiga Canuto, A. Cañete Nieto, L. Martí-Bonmatí, MR denoising increases radiomic biomarker precision and reproducibility in oncologic imaging, J. Digit. Imaging 34 (5) (2021) 1134–1145.
[26] L. Cerdá Alberich, C. Sangüesa Nebot, A. Alberich-Bayarri, J.M. Carot Sierra, B. Martínez de las Heras, D. Veiga Canuto, A. Cañete, L. Martí-Bonmatí, A confidence habitats methodology in MR quantitative diffusion for the classification of neuroblastic tumors, Cancers 12 (12) (2020) 3858.

J. Damian Segrelles Quilis received the B.Sc. and Ph.D. degrees in computer science from the Universitat Politècnica de València(UPV) in 2000 and 2008. He is a member of the Grid and High Performance Computing research group (GRyCAP) at the Institute for Molecular Imaging (I3M) since 2001. He is also associate professor at the Department of Computer Systems and Computation (DSIC) at UPV. He has been involved in Grid and Cloud Technologies and Medical Image processing since 8 years ago and participated more than 20 Regional, National and European Research Projects. He has co-authored more than 40 papers in international, national conferences and workshops, as well as more than 15 papers in high-impact journals referenced in the Journal Citation Reports (JCR) / Science Citation Index (SCI). He belongs to the ICAPA education research team. He has co-authored more than 15 papers in international, national conferences and workshops related to education.

Sergio López Huguet received his Ph.D. in Computer Science and his M.Sc. in Computer Engineering from Universitat Politècnica de València (UPV) in 2021 and 2017, and his B.Sc. in Computational Mathematics from Universitat Jaume I (UJI) in 2015. In 2016, he joined the Grid and High-Performance Computing Group (GRyCAP) in the Institute of Instrumentation for Molecular Imaging (I3M) at the UPV. Nowadays, he is a postdoc researcher in the mentioned Research Group. Besides, he has participated in various international projects, such as PRIMAGE or CHAIMELEON. His broad research interests are Cloud Computing, container management, scientific computing and Medical Image processing.

Pau Lozano received his B.Sc. in Computer Science from Universitat Politècnica de València (UPV) in 2010 and joined the Grid and High Performance Computing research group (GRyCAP) at the Institute of Instrumentation for Molecular Imaging (I3M). He started in the scope of high performance computing for structural analysis and simulation, and latter migrating that computing to the cloud in a European project (VENUS-C). Since then, he has participated in National and European Research Projects related to cloud computing for science, analysis of big data from IoT devices, medical databases and medical image processing in cloud. In the last year also joined the Biomedical Imaging Research Group (GIBI230) at La Fe hospital in Valencia for a short project in this last scope applied to COVID-19 (AVI-COVID).

Ignacio Blanquer is professor of the Computer System Department at UPV since 1999, has been a member of Research Group on Grid and High-Performance Computing of the Institute of Instrumentation for Molecular Imaging (I3M) since 1993 and becoming the leader of this group in 2015 and the vice-director of the I3M in 2019. He is currently a member of the Board of Directors of the European Open Science Cloud (EOSC) Association, the coordinator of the Spanish Network of e-Science and serves as an expert to the Spanish Ministry of Science in the areas of e-Science. He is also the Spanish delegate of e-IRG. He has coordinated three European projects on cloud applied to science and serves as work package coordinator in European projects related to scientific data such as EOSC-SYNERGY, PRIMAGE and CHAIMELEON. He is also a senior researcher in the Biomedical Imaging Research group at La Fe hospital in Valencia. He has been involved in Parallel Computation and Medical Image processing, participating in more than 60 national and European Research Projects, has authored and co-authored 50 articles in indexed journals, as well as more than 100 publications as book chapters, non-indexed journals and national and international conference proceedings.