A cloud architecture for the execution of medical imaging biomarkers[⋆]

Sergio López-Huguet[1], Fabio García-Castro[3], Angel Alberich-Bayarri[2,3], and Ignacio Blanquer[1]

[1] Instituto de Instrumentación para Imagen Molecular (I3M) Centro mixto CSIC - Universitat Politècnica de València Camino de Vera s/n, 46022, Valencia
{serlohu@upv.es, iblanque@dsic.upv.es}

[2] GIBI 230 (Biomed. Imag. Res. Group), La Fe Health Research Institute, Valencia, Spain

[3] QUIBIM (Quantitative Imaging Biomarkers in Medicine) SL, Valencia, Spain
{fabiogarcia,angel}@quibim.com

Abstract. Digital Medical Imaging is increasingly being used in clinical routine and research. As a consequence, the workload in medical imaging departments in hospitals has multiplied by over 20 in the last decade. Medical Image processing requires intensive computing resources not available at hospitals, but which could be provided by public clouds. The article analyses the requirements of processing digital medical images and introduces a cloud-based architecture centred on a DevOps approach to deploying resources on demand, adjusting them based on the request of resources and the expected execution time to deal with an unplanned workload. Results presented show a low overhead and high flexibility executing a lung disease biomarker on a public cloud.

Keywords: Cloud Computing · Medical Imaging · DevOps.

1 Introduction

Traditionally, medical images have been analysed qualitatively. This type of analysis relies on the experience and knowledge of specialised radiologists in charge of carrying out the report. This entails a high temporal and economic cost. The rise of computer image analysis techniques and the improvement of computer systems lead to the advent of quantitative analysis. Contrary to qualitative analysis,

[⋆] The work in this article has been co-funded by project SME Instrument Phase II - 778064, QUIBIM Precision, funded by the European Commission under the INDUSTRIAL LEADERSHIP - Leadership in enabling and industrial technologies - Information and Communication Technologies (ICT), Horizon 2020, project ATMOSPHERE, funded jointly by the European Commission under the Cooperation Programme, Horizon 2020 grant agreement No 777154 and the Brazilian Ministério de Ciência, Tecnologia e Inovação (MCTI), number 51119. The authors would like also to thank the Spanish "Ministerio de Economía, Industria y Competitividad" for the project "BigCLOE" with reference number TIN2016-79951-R.

the quantitative analysis aims to measure different characteristics of a medical image (for example, the size, texture, or function of a tissue or organ and the evolution of these features in time) to provide radiologists and physicians with additional, objective information as a diagnostic aid. In turn, the quantitative analysis requires image acquisitions with the highest possible quality to ensure the accuracy of the measurements.

An imaging biomarker is a characteristic extracted from medical images, regardless of the acquisition modality. These characteristics must be measured objectively and should depict changes caused by pathologies, biological processes or surgical interventions [21] [27]. The availability of large population sets of medical images, the increase of their quality and the access to affordable intensive computing resources has enabled the rapid extraction of a huge amount of imaging biomarkers from medical images. This process allows to transform medical images into mineable data and to analyze the extracted data for decision support. This practice, known as radiomics, provides information that cannot be visually assessed by qualitative radiological reading and reflects underlying pathophysiology. This methodology is designed to be applied at a population level to extract relationships with the clinical endpoints of the disease that can be directed to manage the disease of an individual patient.

The execution of medical image processing tasks, such as biomarkers, is a process that sometimes requires high-performance computing infrastructures and, in some cases, specific hardware (GPUs) that is not available in most medical institutions. Cloud service providers make it possible to access specific and powerful hardware that fits the needs of the workload [29]. Another interesting advantage of the Cloud platforms is the capability of fitting the infrastructure capacity to the dynamic workload, thus improving cost contention. The seamless transition from local image processing and data analytic development environments to cloud-based production-quality processing services implies a Continuous Integration and Deployment DevOps problem that is not properly addressed in current platforms.

1.1 Motivation and Objectives

The objective of this work is to design and implement a cloud-based platform that could address the needs of developing and exploiting medical image processing tools. In this sense, the work focuses on the development of an architecture focusing on the following principles:

– Agnostic to the platform, so the same solution can be deployed on different public and on-premise cloud offerings, adapting to the different needs and requirements of the users and avoiding lock-in.
– Capable of integrating High-performance computing and storage back-ends to deal with the processing of massive sets of medical images.
– Seamlessly integrating development, pre-processing, validation and production from the same platform and automatically.
– Open, reusable, extendable, secure and traceable platform.

1.2 Requirements

A requirement elicitation and analysis process was performed, leading to the identification of 13 requirements, classified into 9 mandatory requirements, 3 recommendable requirements and 3 desirable requirements. The requirements are described in Tables 1, 2 and 3.

RiD	Name	Description	Level
RI1	Resource provisioning	Resources should be automatically configured in the deployment. Provisioning should be performed with minimal intervention by system administrators and should be able to work in multiple IaaS platforms.	R
RI2	Resource isolation	Jobs should run on the system at the maximum level of isolation. Workload may have different and even incompatible software dependencies, and the failure of the execution of a job should not affect the rest of the executions.	M
RI3	Resource scalability	Virtual infrastructures should be automatically reconfigured when adding or removing nodes (limited by a minimum and maximum number of nodes for each type of resource). Elasticity could be triggered externally.	R
RI4	Manag. of Releases	Software should be easy to update and releases should be easy to deploy. This implies automation, minimal customer intervention, progressive rollouts, roll backs, and version freezing.	M
RI5	User Authentication	Users should be able to log-in the system using ad-hoc credentials or an external Identity Provider (IDP) such as Google or Microsoft LiveID.	D
RI6	User Authorisation	Access to the services should be granted only to authorised users. This implies access to data, services and resources. Only special users would be able to access resources.	D
RI7	High availability (HA)	Services must be deployed in HA to guarantee Quality of Service.	M

Table 1. Requirements of the Infrastructure (**M**andatory, **R**ecommended, **D**esirable)

RiD	Name	Description	Level
RE1	Batch execution	The system should run batch jobs. A job will comprise a set of files, software dependencies, hardware requirements, execution arguments, input and output sandbox, job type, memory and CPU requirements.	M
RE2	Workflow execution	A job may include several linked steps that need to be executed according to a data flow. The workflow will imply the automatic execution of the different stages as dependencies are solved.	M
RE3	Job customization	Linked to requirements RI2 and RI4, this requirement poses the need of jobs to run on a customizable environment requiring special hardware, specific software configuration, operating system, and licenses.	M
RE4	Execution triggers	Jobs could also be initiated by means of events. Uploading a file or messages in a queue can spawn the execution of jobs. These reactive jobs will be defined through rules.	D
RE5	Efficient Execution	Jobs should be efficiently executed in the platform. This performance is defined at two levels: a) minimum overhead with respect to the execution on an equivalent pre-installed physical node; b) capability of integrating high-performance resources as GPUs and multicore CPUs.	M

Table 2. Requirements for Job Execution.

RiD	Name	Description	Level
RD1	POSIX access	Jobs expect to find the data to be processed in a POSIX file system in a specific directory route.	M
RD2	ACLs access	Storage access authorization based on a coarse granularity (access granted/denied for both read & write).	M
RD3	Provenance and traceability	Traceability for the derived data is key to bound to the GDPR regulations (e.g. a trained model should be invalidated if the permissions for any part of the data used in the training is revoked).	R

Table 3. Requirements with respect to the Data.

2 State of the art

Since the appearance of Cloud services, a large number of applications have been adapted to facilitate the access of the application users to Cloud infrastructures. In the field of biomedicine we find examples in [22, 32]. On the other hand, there are works that offer pre-configured platforms with a large number of tools for bioinformatic analysis. An example is the Galaxy Project[7], a web platform that can be deployed on public and on-premises Cloud offerings (e.g. using CloudMan [5] for Amazon EC2 [1] and OpenStack [15]).

Another example is Cloud BioLinux [4], a project that offers a series of pre-configured virtual machine images for Amazon EC2, VirtualBox and Eucalyptus [6]. Finally, the solution proposed in [30] is specially designed for medical image analysis using ImageJ [8] in an on-premise infrastructure using Eucalyptus.

Before taking a decision on the architecture, an analysis has been done at three levels: Container technologies, Resource Managers and Job Scheduling.

Containers are a set of methodologies and technologies that aim at isolating execution environments at the level of processes, network namespaces, disk areas and resource limitations. Containers are isolated with respect to: the host filesystem (as they can only see a limited section of it, using techniques such as chroot or FreeBSD Jails), the processes running on the host (only the processes derived executed within the container are visible,for example, using namespaces), and the resources the container can use (as the processes in a container can be bound to a CPU, memory or I/O share, for example, using cgroups).

Containers are used in application delivery, isolation and light encapsulation of resources in the same tenant, execution of processes with incompatible dependencies and improved system administration. Three of the most prominent technologies in the market supporting Containers are Docker [23], Linux Containers [10] and Singularity [25]. Docker has reached the maximum popularity for application delivery due to its convenient and rich ecosystem of tools. However, Docker containers run under the root user space and do not provide multi-tenancy. On the other side, Singularity run containers on the user-space, but access to specific devices is complex. LxC/D is better in terms of isolation but have limited support (e.g. LxD only works in ubuntu [11]). The solution for container isolation selected will be Docker on top of isolated virtual machines.

Resources should be provisioned and allocated for deploying containers. Resource Management Systems (RMSs) deal with the principles of managing a pool of resources and splitting them across different workloads. RMSs manage the resources of physical and virtual machines reserving a fraction of them for a specific workload. RMSs deal with different functionalities, such as: Resource discovery, Resource Monitoring, Resource allocation and release and Coordination with Job Schedulers. We identify 3 technologies to orchestrate resources are: Kubernetes [9], Mesos [2] and EC3 [17][18].

Finally, Job schedulers manage the remote execution of a job on the resources provided by the RMS. Job Schedulers retrieve job specifications from different interfaces, run them on the remote nodes using a dedicated input and output sandbox for the job, monitor its status and retrieve the results.

Job Schedulers may provide other features, such as fault tolerance, complex jobs (bag of tasks, parallel or workflow jobs) and deeply interact with the RMS to access and release the needed resources. We consider in this analysis Marathon [12] Chronos [3], Kubernetes [9] and Nomad [13]. Marathon and Chronos require a Mesos Resource Management System and can deploy containers as long-term fault-tolerant services (Marathon) or periodic jobs (Chronos). Kubernetes has the capacity of deploying containers (mainly Docker but not limited to it) as services, or running batch jobs as containers. However, any of them deal seamlessly with non-containerised and container-based jobs. In this sense, Nomad can deal with multi-platform hybrid workloads with minimal installation requirements.

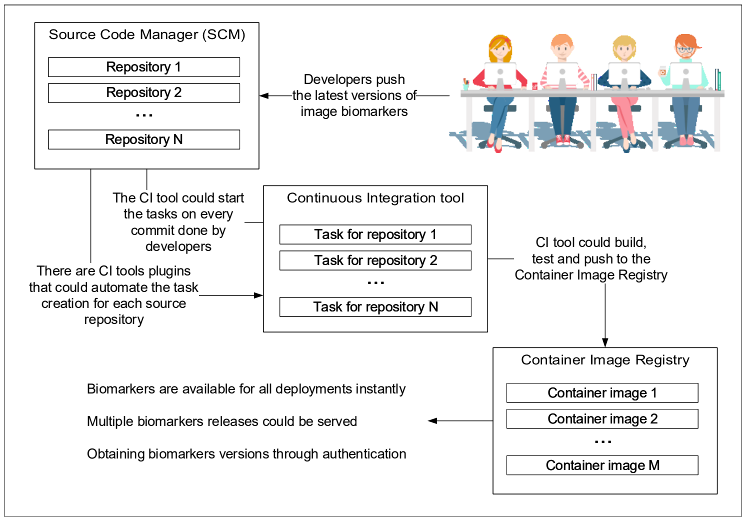
Source Code Manager (SCM)
Repository 1
Repository 2
...
Repository N
Developers push the latest versions of image biomarkers
The CI tool could start the tasks on every commit done by developers
Continuous Integration tool
Task for repository 1
Task for repository 2
...
Task for repository N
There are CI tools plugins that could automate the task creation for each source repository
CI tool could build, test and push to the Container Image Registry
Container Image Registry
Container image 1
Container image 2
...
Container image M
Biomarkers are available for all deployments instantly
Multiple biomarkers releases could be served
Obtaining biomarkers versions through authentication

Fig. 1. Overview of Container Delivery architecture

3 Architecture

The service-oriented architecture is described and implemented in a modular manner, so components could easily be replaced. In Section 3.1, the architecture is described in a technology-agnostic way, so different solutions could fit into the architecture. Section 3.2 describes all architecture components and how they fulfill the use case requirements identified in Section 1.2. Finally, Section 3.3 shows the final version of the architecture including the technologies selected and how each one addresses the requirements.

3.1 Overview of the Architecture

The system architecture addresses the requirements described in section 1.2. The architecture can be divided into two parts: Container Delivery (CD), and Container Execution (CE). It should be noted that although all the components of the architecture can be installed in different nodes, in some cases they could be installed in the same node to reduce costs. As it can been seen in Figure 1, the CD architecture is composed of three components: the Container Image Registry, the Source Code Manager (SCM), and the Continuous Integration (CI) tool.

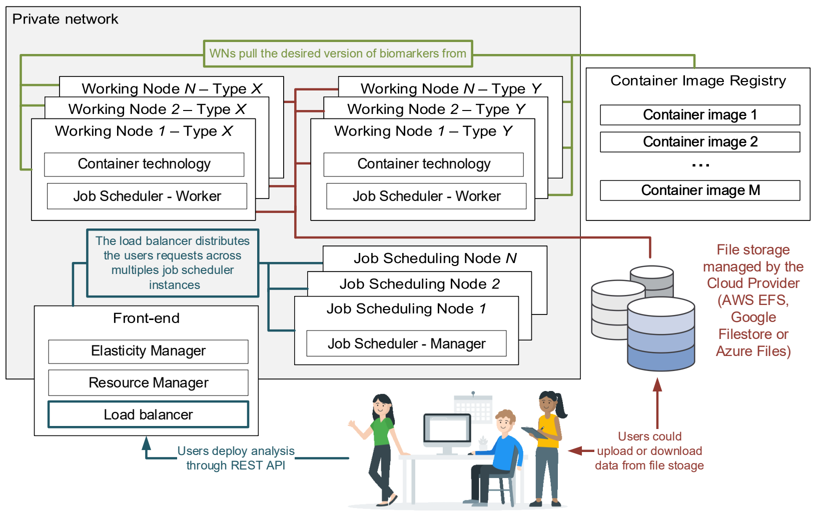

Fig. 2. Overview of Container Execution architecture

Figure 2 depicts the CE architecture. The system consists of four types of logical nodes: Front-end, job schedulers nodes, working nodes and Container Image Registry (this component also appear in CD architecture describe above). The Front-end and job schedulers nodes are interconnected using a private network. The Front-end logical node exposes a REST API, which allows load-balanced communication with the REST API of the job scheduler nodes. Furthermore, it contains the Resource Management Service, which is composed of the Cloud Orchestrator and the horizontal elasticity manager. Job schedulers nodes comprise the master services of the Job Scheduler (JS). Furthermore, as the Front-end is the gateway between users and the job scheduler service, a service for providing authorization and load balancing (between job scheduler nodes) is required. Different working nodes will run the Job scheduler executors. Working nodes mount locally a volume available from a global external storage. Is should be noted that the set of working nodes can be heterogeneous.

3.2 Components

Resource Management service (RMS): It is in charge of deploying the resources, configuring them and to reconfigure them according to the changes on the workload. The requirements stated in Section 1.2 focus on facilitating deployment, higher isolation, scalability, application releases management and generic authentication and authorisation mechanisms. RMS may require to interact with the infrastructure provider to deploy new resources (or undeploy them), and to configure the infrastructure accordingly. Furthermore, the deployment should be maintainable, reliable, extendable and platform agnostic.

Job Scheduling service: It will perform the execution of containerised jobs requested remotely by a client application through the REST API. It is required that the job scheduler service includes a monitoring system to provide up-to-date information on the status of the jobs running in the system. Clients will submit jobs through the load balancer service providing a job description formed by any additional information required by the Biomarkers platform, the information related to the container image, the input and output sandbox, and the software and hardware requirements (such GPUs, memory, etc.).

Horizontal Elasticity service: It is necessary to fulfill the Resource scalability requirement, which is strongly related to the Job scheduler and the Resource Manager. Horizontal elasticity tool has to be able to monitor the job scheduler queue and running jobs, and current working node resources in order to scale in or scale out the infrastructure. It is desirable that the horizontal elasticity manager could maintain a certain number of nodes always idle or stopped to reduce the time that the jobs are queued.

Source Code Manager (SCM): It is required to manage the coding source for developers. Due to the release management requirement and the development complexity, it is mandatory to lean on this kind of tools.

Container Image Registry: In order to store and delivery the biomarker applications, it is necessary to use a Container Image Registry. Biomarker applications could be bound to Intellectual Property Rights (IPR) restrictions so Container Image Registry must be private. For this reason, authentication mechanisms are required for obtaining images from the registry. Working nodes will pull the application images when the container image not exists or recent version exists.

Continuous Integration (CI) tool: CI eases the development cycle because it automates building, testing and pushing to the Container Image Resgistry the biomarker application (with a certain version or tag). Developers or (the CI experts) define this workflow to do it. Furthermore, some of CI tools could trigger these tasks for each SCM commits.

Storage: Biomarker applications make use of legacy code and standard libraries which expect data to be provided in a POSIX filesystem. For this reason, the storage technology must mount the filesystem in the container.

3.3 Detailed Architecture

The previous sections describe the architecture and its components. After the technology study done in Section 2 and the feature identification for each architecture component in 3.2, the technologies selected for addressing the requirements are the following:

– Jenkins as the CI tool due to the wide variety of available plugins (such us SCM plugins). Furthermore, it allows you to easily define workflows for each task using a file named Jenkinsfile. Jenkins provides means to satisfy the requirements RI4, RI5 and RI6.
– GitHub as SCM because it supports both private (commercial license) and public repositories, which are linked to the CI tool. Furthermore, there is a Jenkins plugin that could scan a GitHub organization and create Jenkins tasks for each repository (and also for each branch) that contains a Jenkinsfile, addressing to requirement RI4. This component meets the requirements RI5, RI6 and RI7.
– Hashicorp Nomad is the Job scheduler. We selected Nomad instead of Kubernetes due to Kubernetes can only run Docker containers. Furthermore, Nomad incorporates job monitoring that can be consulted by users. Additionally, it is designed to work in High Availability mode, addressing requirement RI7. Hashicorp Consul is used to resources service discovery as Nomad could use it natively. By using this job scheduler, the architecture meets the requirements RI2, RI4 (job versioning), RE1, RE2, RE3 and satisfyies RI5 and RI6 using its Access Control List (ACL) feature.
– Docker is the container platform selected because it is the most popular container technology and it is supported by wide variety of Job schedulers. It provides the resources isolation required by RI2 and support version management (by tagging the different images) of RI4.
– As Docker is the container platform used in this work, Docker Hub and Docker Registry are used as, respectively, public and private container image registry. Requirements RI4 and RI6 are address by using Docker.
– Infrastructure Manager (IM) [17] is the orchestrator chosen because it is open source, cloud agnostic and provides the required functionality to fulfill the use case requirements RI1, RI3, RI5 and RI6. In [26], IM is used for deploy 50 simultaneous nodes.
– CLUES [19] has been chosen for addressing RI3 because it is open source and can scale up or down infrastructures using IM by monitoring the Nomad jobs queue. CLUES can auto-scale the infrastructure according to different types of workloads [16][26].

– The RMS selected is EC3, which is a tool for system administrators that combines IM and CLUES to configure, create, suspend, restart and remove infrastructures. By using EC3 the system could address the requirements RI, RI3, RI5 and RI6.
– Due to the experimentation will be done in Azure and the current storage solution of QUIBIM is Azure Files, it has been selected as storage. It allows to mount (entirely o partially) the data as a POSIX filesystem, which is the requirement RD1. Also, it provides the mechanisms required to fulfill RI5, RI6 and RD2. Additionally, it allows to mount the same filesystem concurrently.
– HAProxy is used for load balancing because it is reliable, open source and support LDAP or OAuth for authentication.

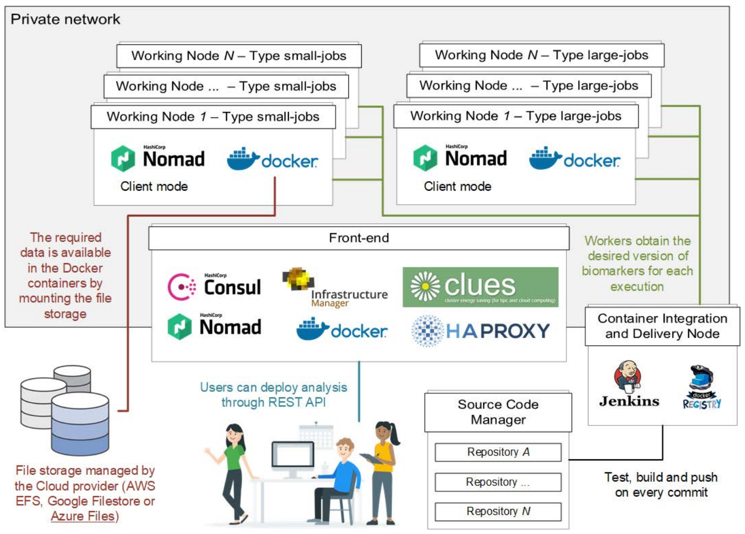
Private network
Working Node N – Type small-jobs
Working Node ... – Type small-jobs
Working Node 1 – Type small-jobs
Nomad
docker
Client mode
Working Node N – Type large-jobs
Working Node ... – Type large-jobs
Working Node 1 – Type large-jobs
Nomad
docker
Client mode
The required data is available in the Docker containers by mounting the file storage
Front-end
Consul
Infrastructure Manager
clues
Nomad
docker
HAPROXY
Workers obtain the desired version of biomarkers for each execution
Container Integration and Delivery Node
Jenkins
Registry
Users can deploy analysis through REST API
Source Code Manager
Repository A
Repository ...
Repository N
Test, build and push on every commit
File storage managed by the Cloud provider (AWS EFS, Google Filestore or Azure Files)

Fig. 3. Proposed architecture with selected technologies.

It should be remarked that the proposed architecture (which is depicted in Figure 3) is a simplification of Figure 2. For the experiment performed, the job scheduling services are deployed in the Front-end node but users connect with the job schedulers using the load balancer service. So, this simplification does not affect to the users-services communication. Furthermore, in order to avoid costs, the CI tool (Jenkins) and the Docker Private Registry are in the same resource.

4 Results

The experiments have been performed on the public Cloud Provider Microsoft Azure. The infrastructure is composed by three type of nodes. The *front-end* node corresponds with the A2 v2 instance, which has two Intel(R) Xeon(R) CPU E5-2660 2.20GHz, 3.5GB of RAM memory, 20 GB of Hard Drive disk and two network interfaces. IM version 1.7.6 and CLUES version 2.2.0b has been installed in this node. HAProxy 1.8.14-52e4d43 and Consul 1.3.0 are running on Docker containers also in that node. The second type of node, *smallwn*, corresponds with the NC6 instance with six Intel(R) Xeon(R) CPU E5-2690 v3 2.60GHz, 56 of memory RAM, 340 GB of Hard Drive disk, one NVIDIA Tesla K80 and one network interface. Finally, the *largewn* node type corresponds with the D13 v2 instance, which has eight Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz, 56GB of RAM memory, 400 GiB of Hard Drive disk and one network interface. The operating system is CentOS Linux release 7.5.1804. Nomad version 0.8.6 and Docker version 18.09.0 build 4d60db4 are installed in all nodes.

4.1 Deployment

The infrastructure configuration is coded into Ansible roles[4] and RADL recipes, and they include parameters to differentiate among the deployment. The roles will reference a local repository of packages or specific versions to minimize the impact of changes in public repositories, as well as certified containers. Deployment time is the time required to create and configure a resource. The deployment time of the front-end takes 29 minutes 28 seconds on average. The time required to configure each worker node is 9 minutes 30 seconds.

4.2 Use case - Emphysema

The use case selected was the automatic quantification of lung emphysema biomarkers from computed tomography images. This pipeline features a patented air thresholding algorithm from QUIBIM [28] [24] for emphysema quantification and an automatic lung segmentation algorithm. Two versions were implemented. A fast one with rough lung segmentation can be used during the interactive inspection and validation of parameters. Another one with higher segmentation accuracy is implemented for the batch, production case. This brings the need of supporting short and long cases, which take respectively, 4 and 20 minutes.

The small cases are related with executions that take minutes to be completed. In order to provide QoS, CLUES is configured to provide always more resources than required (one node always free). As these type of jobs are very fast, the small-jobs nodes that are IDLE too much time are suspended for avoiding the deployment time. The large cases of QUIBIM biomakers could take many hours and use huge amount of resources, so the deployment time is negligible. For this reason, although the large case of this work takes the same amount of time that the deployment time and the application does not need the all resources of the VM, large-jobs nodes are not suspended and restarted, and only one large-job can run concurrently on the same VM. The “small” Emphysema

[4] All Ansible Roles used in this work are available in a GitHub repository. https://github.com/grycap/

used in this work consumes 15 GB of memory RAM and 2 vCPUs, so three Emphysema small-jobs could run simultaneously on the same node.

The main goal of the experiment is to demonstrate the capabilities of the proposed architecture. The experiment consists of submitting 35 small-jobs and 5 large-jobs in order to ensure that there are workload peaks that require starting up new VMs and idle periods long enough to remove (in case of large-jobs nodes) or suspend VMs (in case of small-jobs nodes). Table 4 shown the time frames were jobs are submitted.

Name	Time	Name	Time	Name	Time	Name	Time
small-1	0:01:19	*small-10*	0:47:25	*small-18*	1:00:31	*small-27*	1:10:42
large-1	0:01:20	*small-11*	0:49:25	*small-19*	1:01:32	*large-5*	1:11:44
small-2	0:01:50	*small-12*	0:50:25	*small-20*	1:01:33	*small-28*	1:11:46
small-3	0:05:50	*large-2*	0:51:27	*small-21*	1:02:33	*small-29*	1:14:47
small-4	0:08:51	*small-13*	0:51:27	*small-22*	1:03:34	*small-30*	1:15:48
small-5	0:11:51	*small-14*	0:52:28	*small-23*	1:03:36	*small-31*	1:16:48
small-6	0:16:23	*small-15*	0:55:28	*small-24*	1:08:38	*small-32*	1:17:49
small-7	0:16:24	*small-16*	0:55:29	*small-25*	1:08:38	*small-33*	1:18:49
small-8	0:19:24	*small-17*	0:58:30	*small-26*	1:08:39	*small-34*	1:18:50
small-9	0:22:25	*large-3*	0:59:30	*large-4*	1:09:40	*small-35*	1:21:50

Table 4. Scheduling of the jobs to be executed.

Figure 4 shows the number of jobs (vertical axis) along time (horizontal axis) in the different status: SUBMITTED, STARTED, FINISHED, QUEUED and RUNNING. The first three metrics denote the cumulative number of jobs that have been submitted, have actually started and have been completed over time, respectively. The remaining metrics denote the number of jobs that are queued or concurrently running at a given time.

As depicted in Figure 4, the length of the queue does not grow above ten jobs. The delay between the submission and the start of a job (the difference in the horizontal axis between submitted and started lines) is negligible during the first twenty minutes of the experiment. After a period without new submissions (between 00:24:00 and 00:47:00 minutes), the workload grows again due to the submission of new jobs, triggering the deployment of four new nodes (as it can be seen in Figure 5).

The largest number of queued jobs (10) is reached during this second deployment of new resources at 01:15:09, although it decreases to four only three minutes later (and to two at 1:23:00). Besides, the four last jobs queued were large-jobs. It should be pointed out that large-jobs have greater delays than short-jobs between submission and starting as they use dedicated nodes and this type of nodes are eliminated after 1 minute without jobs allocated (a higher value will be used in production).

Figure 5 depicts the status of the nodes along the experiment, which could be USED (executing jobs), IDLE (powered-on and without jobs allocated), POWON (being started, restarted or configured), POWOFF (being suspended or removed), OFF (not deployed or suspended) or FAILED. CLUES is configured to ensure that a small-node is always active (in status IDLE or USED). It can be seen in Figure 5 that two nodes are deployed at the start of the experiment. Then, during the period without new submissions, the running jobs end their executions, so these new nodes are powered off after one minute in the IDLE status. As the workload grows, the system deployed four new nodes between 00:50:00 and 01:13:00. Besides, CLUES tried to deploy one more node (a large-node) but, as the quota limit of the cloud provider's account has reached, the deployment was failed. It should be noted that the system is resilient to this type of problems and successfully ended the experiment. After two hours, all jobs are completed, so CLUES suspends or removes all nodes (except one small-node that has to be active always).

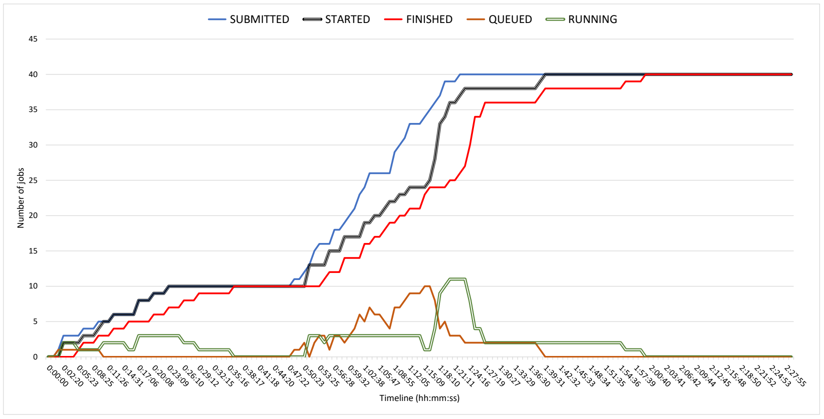

Fig. 4. Status of jobs during the experiment.

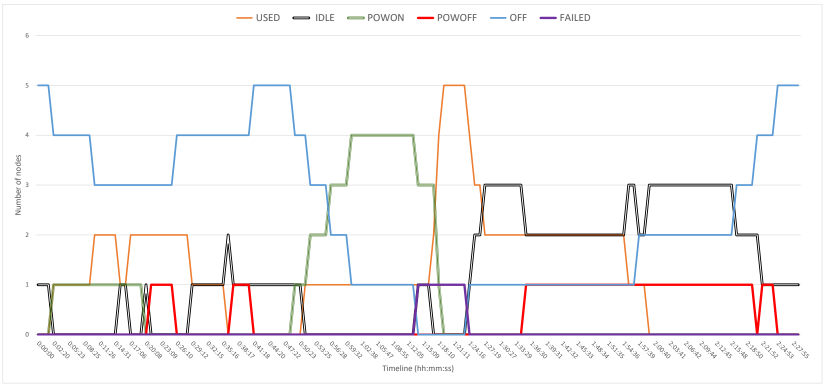

Fig. 5. Status of working nodes.

5 Conclusions and future work

This paper has presented a agnostic and elastic architecture and a set of open-source tools for the execution of medical imaging biomarkers. Regarding the technical requirements defined in Section 1.2, the experiment of a real use case and the results exposed, it can be concluded that all the requirements proposed were fulfilled by the architecture presented. In Section 4.2, a combination of 40 batch jobs was scheduled to be executed in a specific time and the cluster achieve to execute all of them by adjusting the resources available. Furthermore, when there are wasted resources too much time, the nodes are suspended (or eliminated). The architecture uses IM and CLUES which has proven a good scalability [26] and the capability to work with unplanned workloads [16].

The proposed architecture are not only related to the execution of batch jobs, it provides to developers a workflow to ease the building, testing, delivery and version management of their application.

Future work includes implementing the proposed architecture on QUIBIM ecosystem, testing other solutions for distributed storage as Ceph [33] or One-Data [20], the study of Function As a Services (FaaS) frameworks for executing batch jobs (SCAR [31] or OpenFaas [14]) or using Kubernetes for ensuring that services (Nomad, Consul and HAProxy) are always up.

References

1. Amazon EC2 web site. `https://aws.amazon.com/es/ec2/`, accessed: 29-12-2018
2. Apache Mesos web site. `http://mesos.apache.org/`, accessed: 29-12-2018
3. Chronos web site. `https://mesos.github.io/chronos/`, accessed: 29-12-2018
4. Cloudbiolinux web site. `http://cloudbiolinux.org/`, accessed: 29-12-2018
5. Cloudman web site. `https://galaxyproject.org/cloudman`, accessed: 29-12-2018
6. Eucalyptus web site. `https://www.eucalyptus.cloud/`, accessed: 29-12-2018
7. Galaxy Platform web site. `https://galaxyproject.org`, accessed: 29-12-2018
8. Imagej web site. `https://imagej.nih.gov/ij/`, accessed: 29-12-2018
9. Kubernetes web site. `https://kubernetes.io`, accessed: 29-12-2018
10. Linux containers. `https://linuxcontainers.org/`, accessed: 29-12-2018
11. Lxd documentation. `https://lxd.readthedocs.io/`, accessed: 29-12-2018
12. Marathon. `https://mesosphere.github.io/marathon/`, accessed: 29-12-2018
13. Nomad web site. `https://www.nomadproject.io/`, accessed: 29-12-2018
14. OpenFaas web site. `https://www.openfaas.com/`, accessed: 29-12-2018
15. OpenStack web site. `https://www.openstack.org/`, accessed: 29-12-2018
16. de Alfonso, C., Caballer, M., Calatrava, A., Moltó, G., Blanquer, I.: Multi-elastic Datacenters: Auto-scaled Virtual Clusters on Energy-Aware Physical Infrastructures. Journal of Grid Computing (jul 2018). https://doi.org/10.1007/s10723-018-9449-z
17. Caballer, M., Blanquer, I., Moltó, G., de Alfonso, C.: Dynamic Management of Virtual Infrastructures. Journal of Grid Computing **13**(1), 53–70 (2015). https://doi.org/10.1007/s10723-014-9296-5
18. Calatrava, A., Romero, E., Moltó, G., Caballer, M., Alonso, J.M.: Self-managed cost-efficient virtual elastic clusters on hybrid Cloud infrastructures. Future Generation Computer Systems **61**, 13–25 (2016). https://doi.org/10.1016/j.future.2016.01.018

19. De Alfonso, C., Caballer, M., Alvarruiz, F., Hernández, V.: An energy management system for cluster infrastructures. In: Computers and Electrical Engineering. vol. 39, pp. 2579–2590 (2013). https://doi.org/10.1016/j.compeleceng.2013.05.004
20. Łukasz Dutka, Wrzeszcz, M., Lichoń, T., Słota, R., Zemek, K., Trzepla, K., Łukasz Opioła, Słota, R., Kitowski, J.: Onedata – a step forward towards globalization of data access for computing infrastructures. Procedia Computer Science **51**, 2843 – 2847 (2015). https://doi.org/10.1016/j.procs.2015.05.445, international Conference On Computational Science, ICCS 2015
21. European Society of Radiology (ESR): White paper on imaging biomarkers. Insights into Imaging **1**(2), 42–45 (May 2010). https://doi.org/10.1007/s13244-010-0025-8
22. Hyungro Lee: Using Bioinformatics Applications on the Cloud. `http://dsc.soic.indiana.edu/publications/bioinformatics.pdf` (2013), online; accessed 29 December 2018
23. Inc., D.: Docker. `https://www.docker.com/`, accessed: 29-12-2018
24. Irene Mayorga-Ruiz, David García-Juan, A.A.B.F.G.C.L.M.B.: Fully automated method for lung emphysema quantification for Multidetector CT images. `http://quibim.com/wp-content/uploads/2018/02/ECR_Fully-automated-quantification-of-lung-emphysema-using-CT-images.pdf`, accessed: 22-03-2019
25. Kurtzer, G.M., Sochat, V., Bauer, M.W.: Singularity: Scientific containers for mobility of compute. PLOS ONE **12**(5), 1–20 (05 2017). https://doi.org/10.1371/journal.pone.0177459
26. López-Huguet, S., Pérez, A., Calatrava, A., de Alfonso, C., Caballer, M., Moltó, G., Blanquer, I.: A self-managed Mesos cluster for data analytics with QoS guarantees. Future Generation Computer Systems **96**, 449–461 (2019). https://doi.org/10.1016/j.future.2019.02.047
27. Martí-Bonmatí, L., Alberich-Bayarri, A.: Imaging biomarkers: Development and clinical integration. Springer-Verlag GmbH (2016). https://doi.org/10.1007/978-3-319-43504-6
28. Martí-Bonmatí, L., García-Martí, G., Alberich-Bayarri, A., Sanz-Requena, R.. QUIBIM SL.: Método de segmentación por umbral adaptativo variable para la obtención de valores de referencia del aire corte a corte en estudios de imagen por tomografía computarizada, ES 2530424B1, 02 September 2013
29. Marwan, M., Kartit, A., Ouahmane, H.: Using cloud solution for medical image processing: Issues and implementation efforts. In: 2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech). IEEE (Oct 2017). https://doi.org/10.1109/cloudtech.2017.8284703
30. Mirarab, A., Fard, N.G., Shamsi, M.: A cloud solution for medical image processing. Int. Journal of Engineering Research and Applications **4**(7), 74–82 (2014)
31. Pérez, A., Moltó, G., Caballer, M., Calatrava, A.: Serverless computing for container-based architectures. Future Generation Computer Systems **83**, 50 – 59 (2018). https://doi.org/10.1016/j.future.2018.01.022
32. Shakil, K.A., Alam, M.: Cloud Computing in Bioinformatics and Big Data Analytics: Current Status and Future Research, p. 629–640. Springer Singapore (Oct 2017). https://doi.org/10.1007/978-981-10-6620-7_60
33. Weil, S.A., Brandt, S.A., Miller, E.L., Long, D.D.E., Maltzahn, C.: Ceph: A scalable, high-performance distributed file system. In: Proceedings of the 7th Symposium on Operating Systems Design and Implementation. pp. 307–320. OSDI '06, USENIX Association, Berkeley, CA, USA (2006), `http://dl.acm.org/citation.cfm?id=1298455.1298485`