Infrastructure Manager: A TOSCA-based Orchestrator for the Computing Continuum

Miguel Caballer[1*], Germán Moltó[1], Amanda Calatrava[1], Ignacio Blanquer[1]

[1]Instituto de Instrumentación para Imagen Molecular (I3M). Centro mixto CSIC - Universitat Politècnica de València.

*Corresponding author(s). E-mail(s): micafer1@upv.es;
Contributing authors: gmolto@dsic.upv.es; amcaar@i3m.upv.es; iblanque@dsic.upv.es;

Abstract

The edge-to-cloud continuum involves heterogeneous computing resources, including low-power physical devices, Virtual Machines (VMs) in cloud management platforms and serverless computing services based on the FaaS (Functions as a Service) model. This requires novel strategies to describe and efficiently deploy complex applications that execute across the computing continuum. To this end, this paper introduces the developments in the Infrastructure Manager (IM), an open-source TOSCA-based orchestrator to provision and configure virtualized computing resources from a wide range of cloud platforms. By supplementing TOSCA with additional types, the IM can also provision from FaaS platforms across the computing continuum by leveraging public cloud services such as AWS Lambda and on-premises serverless platforms, such as OSCAR. This allows event-driven data-processing applications across multiple computing platforms and architectures. The evolution of the Infrastructure Manager is described to accommodate the definition in TOSCA of complex applications that span across the computing continuum and their automated provisioning and configuration using Infrastructure as Code (IaC) approaches. Its effectiveness is assessed through a real use case involving a machine-learning classifier application for assisting in the early diagnosis of Rheumatic Heart Disease (RHD). The results show that the new developments enable the IM to efficiently deploy complete application architectures described in TOSCA across the computing continuum, from VMs to FaaS services.

Keywords: Cloud Computing, Computing continuum, TOSCA, Virtual Infrastructures

1 Introduction

In the quest for reduced latency and increasing security and privacy, processing data as close as possible to where it was generated paved the way for edge computing, in which typically low-powered devices are responsible for gathering data at the edge of the network to perform specific data processing activity [1]. However, the reduced computing capacity of these devices and sensors, together with the requirements of workflow-based processing, requires the ability of data processing delegation across upper layers of the computing continuum. Indeed, suppose the resources of an OpenStack-based on-premises Cloud fail to satisfy the computing requirements of an application. In that case, a typical scenario includes cloud bursting, where a public cloud provider is employed to dynamically provision the required computing resources.

The approach of combining multiple computing layers (edge, on-premises Cloud and public Cloud) encompasses an execution flow among disparate distributed computing infrastructures, commonly known as the edge-to-cloud computing continuum [2]. Allowing data to be processed both at the edge and in the cloud, depending on the specific needs and design of the application, can enable more flexible and efficient data processing. Indeed, some tasks may be better suited for processing at the edge. In contrast, others may require more powerful capabilities from a public cloud. This scenario arises in Artificial Intelligence (AI), where applications can use pre-trained AI models to analyze and process data coming from sensors in real time using edge computing [3]. However, model training can be performed in the cloud, where more powerful computing resources are available.

In this scenario, orchestrating and managing the underlying computing resource pool is not trivial. Cloud computing features several service models, such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). However, the increased trend of abstracting away from the developers the details of resource allocation and provisioning from public Cloud infrastructures, failure management together with fine-grained pay-per-use approaches, supported the rise of the Functions as a Service (FaaS) service model, an approach to implement serverless computing [4]. Indeed, serverless computing allows event-driven execution of managed services where the Cloud provider allocates the computing and storage resources, which are triggered in response to certain events happening in the infrastructure (such as a file upload into a storage system or an invocation of an HTTP-based request).

The serverless computational paradigm has permeated in recent years through different domains such as astronomy [5], agriculture [6] and genomics [7]. The ability to execute functions in response to events, scale quickly and efficiently, and minimize operational overhead allows scientific applications to process large amounts of data and perform complex simulations, making FaaS a versatile and powerful computing model. Event-driven executions across the computing continuum allow supporting multiple uses cases that can benefit from: i) reduced latency by partially processing data close to where it is generated; ii) delegating additional processing into a more powerful computing platform managed by an organization to comply with specific privacy requirements and iii) cloud bursting into a public cloud platform to profit from the highly-elastic capabilities of serverless services.

In a previous work [8], we created the Functions Definition Language (FDL) to define data-driven serverless computing workflows for the OSCAR [9] and the SCAR [10] serverless frameworks. However, this required the computing resources needed to develop and execute the functions to be already deployed and configured beforehand. This paper introduces the new developments performed in the Infrastructure Manager (IM) [11] to support the definition and automated deployment of complex application architectures in TOSCA [12] that need to be executed across the computing continuum using the frameworks above. By using Infrastructure as Code (IaC) and application deployment procedures based on DevOps tools such as Ansible, together with container orchestration platforms such as Kubernetes, the Infrastructure Manager has evolved into a turnkey solution for standards-based application architecture description and deployment in the computing continuum. This new functionality has been made available in the production-ready IM instance[1] available in EGI, the largest international federation in Europe delivering open solutions for advanced computing and data analytics in research and innovation.

After the introduction, the remainder of the paper is structured as follows. First, section 2 reviews and analyzes the related work in the area. Next, section 3 presents the proposed architecture of the IM orchestration tool, together with its key components. Then, section 4 shows the extension made in the IM for the edge-to-cloud continuum support. Next, section 5 evaluates the proposed solution through the execution of a real use case involving a machine-learning classifier application for assisting in the early diagnosis of Rheumatic Heart Disease (RHD. Finally, section 6 summarizes the main achievements and points to the future actions and improvements of the proposed solution.

2 Related Work

One of the most popular Infrastructure as Code (IaC) tools is Terraform, an open-source software to safely and predictably change and improve infrastructure. However, Terraform does not support the OASIS TOSCA (Topology and Orchestration Specification for Cloud Applications). Using open specifications introduces easier extensibility and portability across different TOSCA runtime engines. In addition, the European Code of Conduct for Research Integrity [13] advocates for the usage of standards of the discipline to facilitate verification and reproducibility. Authors such as Santana-Perez et al. [14] mention using TOSCA to recreate execution environments for enhanced reproducibility.

As presented in Table 1, there are different TOSCA-compliant runtime engines, such as Cloudify, which offers some of the functionality for free (an integration with Terraform, to access several cloud providers). It also has support for serverless computing through a plugin[2]. Still, a pricing plan is required to unlock the support for other technologies, such as Kubernetes. Puccini[3] is a cloud topology management and deployment tool based on TOSCA. It is mainly a TOSCA processor supporting several TOSCA compilations (TOSCA 2.0 support is in progress) with connection with some

[1]Infrastructure Manager (IM) - https://im.egi.eu
[2]Cloudify serverless plugin - https://github.com/cloudify-incubator/cloudify-serverless-plugin/
[3]Puccini - https://puccini.cloud

	Cloudify	Puccini	xOpera	Yorc	MiCADO	IM
Open Source	Partially	Yes	Yes	Yes	Yes	Yes
GUI	Yes	No	Yes	Yes (Alien4Cloud)	Yes	Yes
Clouds	OpenStack, vCloud, Azure, vSphere, AWS, Google Cloud, Terraform	OpenStack, Ansible	Ansible	AWS, Google, OpenStack	OpenStack, Azure, AWS, Google, Oracle, CloudSigma, CloudBroker	OpenNebula, AWS, Google, Azure, OTC, Orange, Linode, OpenStack, FogBow, EGI Cloud Compute.
FaaS	Yes	No	Yes	No	No	Yes
TOSCA compliance	Uses its own version schema: cloudify_dsl_1_X	TOSCA 2.0* TOSCA 1.3	TOSCA 1.3	TOSCA 1.2	TOSCA 1.3	TOSCA 1.0
Publicly available as a service	Yes (pay-per-use)	Demo	No	No	No	Yes
GitHub Stars	139	83	34	60	18	54

Table 1 Comparative of different TOSCA-compliant runtime engines with our IM solution.

orchestrators, such as Kubernetes and OpenStack. xOpera[4] is a lightweight TOSCA orchestrator used in the RADON, SODALITE and PIACERE projects. It features a CLI, API and a web-based GUI. In xOpera all templates are translated to Ansible to access several cloud providers and even allows to define functions as a service, but this requires creating the corresponding Ansible playbooks. Yorc[5] is a hybrid Cloud/HPC TOSCA orchestrator, used in Alien4Cloud[6], a visual composition tool for TOSCA, which is no longer maintained. Finally, MiCADO [15] is an application-level cloud orchestrator designed to work in a cloud-to-edge computing continuum environment. Their system is based on KubeEdge as the orchestrator, a Kubernetes solution for the execution of containerised applications in non-cloud workers. Thanks to the integration with Terraform and Occopus, the tool can access several cloud providers. However, they do not take advantage of public serverless platforms such as AWS Lambda, as opposed to our approach.

It is essential to point out that most of the TOSCA-compliant runtimes use specific node types to define the resources in each cloud provider, and, moreover, none of them supports abstract node types. This approach facilitates the translation from TOSCA to the particular cloud provider but makes TOSCA templates specific for a cloud provider. This prevents the TOSCA descriptions, using concrete types, from being cloud agnostic and, therefore, a concrete TOSCA description with enough details to fully describe user requirements cannot be used to deploy the same infrastructure in different providers. Instead, the IM uses a different approach; it uses generic types, "Compute", "Storage", etc., to specify the topologies in a cloud-agnostic way, and it is in charge of translating them into particular cloud provider API calls.

The work of Tusa and Clayman [16] proposes the concept of end-to-end *slices*, a graph-based model for dynamic resource discovery, selection and mapping through algorithms and optimisation goals as the foundation for the dynamic allocation of compute and network resources and services in a cloud-to-edge continuum scenario. Rather than extending existing cloud orchestration solutions towards the edge, the authors propose a new orchestration strategy based on the concept of these slices that allow the separation of concerns between resource orchestration, in charge of

[4]xOpera - https://github.com/xlab-si/xopera-opera
[5]York - https://github.com/ystia/yorc
[6]Alien4Cloud - https://alien4cloud.github.io

the selection, configuration and management of compute and network resources and service orchestration, that is, the life-cycle management of the distributed components that deliver a service. The paper presents the formal definition of this concept and proves its effectiveness through experiments. Further developments are needed to test this proposal on a real scenario with a real implementation of the orchestration strategy in a geographically distributed scenario.

There also exist in the literature previous works to introduce serverless computing support in TOSCA. This is the case of the work by Wurster et al. [17], which introduces an event-driven deployment modeling approach using TOSCA that supports the standard lifecycle to provision and manage multi-cloud serverless applications. Also, the work by Yussupov et al. [18] presents a toolchain to model serverless function orchestrations in BPMN, enacting their deployment. The work by Dehury et al. [19] proposes an extension of the TOSCA standard for modeling data pipeline-based cloud applications, including serverless platforms, as done in the RADON project. Also, the Emerging Compute Models: Recommendations and Sample Profile Version 1.0 document provides an approach to extend the TOSCA specification with support for serverless computing and other computing paradigms. Their approach is based on RADON TOSCA, introducing a new type hierarchy.

Concerning our previous work, the Infrastructure Manager (IM), as initially described in the work by Caballer et al. [11] is an open-source[7] TOSCA-based service that simplifies the provisioning and configuration of complex cloud-based application architectures on multiple cloud back-ends using cloud agnostic TOSCA templates. The IM has been extensively developed in the framework of multiple European projects (e.g. INDIGO-DataCloud, EOSC-Hub, DEEP Hybrid-DataCloud, EGI-ACE, AI-SPRINT, DT-GEO, InterTwin, AI4EOSC, etc.) to achieve TRL 8[8], which refers to systems demonstrated in an operational environment, which involve rigorous testing and validation to ensure that the technology functions as expected. It is currently being used in production in EGI[9] to provide users with the ability to self-deploy, in a simple way, complex computing infrastructures and services (e.g. Galaxy portals, Kubernetes clusters, SLURM-based clusters, etc.). This paper introduces novel developments in the IM to orchestrate the deployment of complex application architectures across the edge-to-cloud continuum. By supplementing TOSCA with additional custom types, the IM supports the definition of application architectures that include FaaS-based applications that can be deployed using open-source serverless platforms such as SCAR[10] [10], which uses AWS Lambda and AWS Batch, and OSCAR[11] [8], a serverless platform which uses Kubernetes and Knative for event-driven data processing. Together, they provide an open solution for the standards-based definition of application architectures for the computing continuum and its automated deployment across multiple Clouds, platforms, and even different computer architectures.

[7]IM Server - https://github.com/grycap/im
[8]Technology Readiness Levels - https://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415-annex-g-trl_en.pdf
[9]EGI - https://www.egi.eu/
[10]SCAR - https://github.com/grycap/scar
[11]OSCAR - https://oscar.grycap.net

3 Infrastructure Manager: Architecture and Components

The Infrastructure Manager (IM)[12] [11][20] is an open-source TOSCA-based orchestrator that has been developed throughout the last decade to support the deployment of complex application architectures across a myriad of IaaS Cloud back-ends, which include the most popular public Cloud Platforms (Amazon Web Services, Microsoft Azure, Google Cloud Platform) as well as European public providers such as Open Telekom Cloud, Cloud & Heat and Orange Cloud, European federated platforms like EGI Cloud Compute and on-premises platforms like OpenStack or OpenNebula.

Since 2017 EGI has provided users with a web portal to deploy virtual infrastructures named the Virtual Machine Operations (VMOps) dashboard. The VMOps dashboard provides graphical interfaces and high-level services for users to manage Virtual Machines and Virtualised Appliances on federated cloud infrastructures, particularly on the EGI Federated Cloud and community-specific cloud federations (e.g. the ELIXIR Competence Centre Cloud Federation). In this context, the IM was adopted as the communication layer (initially OCCI and currently OpenStack), in charge of deploying and configuring the resources needed by the users [21]. Therefore the IM has been running in production for more than five years in EGI Cloud Compute[13], where it is the preferred solution to deploy complex infrastructures in the cloud. In fact, it is offered as the Cloud orchestration solution in EGI[14], thus being available to any user approaching the EGI Federation to provision virtual infrastructures. It is also part of the EOSC (European Open Science Cloud)[15], an environment for hosting and processing research data to support EU science. Also, the IM is in charge of interacting with the Cloud Management Frameworks to deploy and configure the resources in the sites selected by the INDIGO PaaS Orchestrator [22], a component to provision virtualized compute and storage resources in Clouds and to run Docker-based long-running jobs on Apache Mesos and Kubernetes clusters, used in several European projects such as INDIGO-DataCloud and C-SCALE. Finally, PROMINENCE [23] is a platform that allows users to exploit idle cloud resources for running scientific workloads with a simple batch system-style interface. From a user's perspective, PROMINENCE appears like a standard batch system, but jobs can be run from anywhere in the world on clouds anywhere in the world. All infrastructure provisioning is handled completely automatically and is invisible to the user by means of the IM.

Fig. 1 shows the architecture of the IM: The IM internal components are shown with a green background; the internal components that interact with either client apps or cloud providers are shown in blue; finally, in red are shown the external components that are used or use the IM to get enhanced functionality. IM provides two different APIs: XML-RPC and REST, which follows the OpenAPI standard specification[16]. Two main interfaces are available for the user: the IM Dashboard[17] and the IM CLI

[12]Infrastructure Manager (IM) - https://www.grycap.upv.es/im
[13]EGI Cloud Compute - https://www.egi.eu/service/cloud-compute/
[14]https://www.egi.eu/service/infrastructure-manager/
[15]IM in the EOSC Marketplace - https://marketplace.eosc-portal.eu/services/eosc.egi-fed.infrastructure_manager
[16]IM REST API - https://app.swaggerhub.com/apis-docs/grycap/InfrastructureManager/
[17]IM Dashboard - https://github.com/grycap/im-dashboard

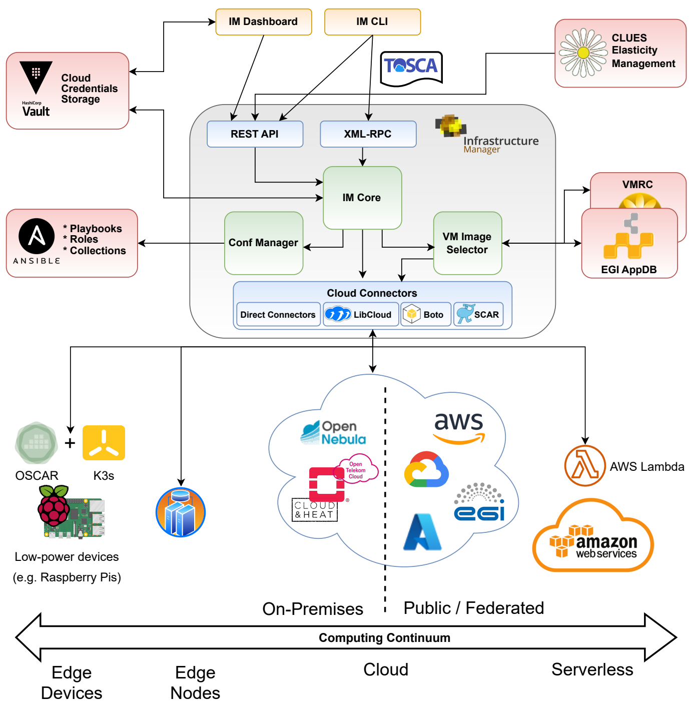

Fig. 1 Infrastructure Manager (IM) Architecture.

(command-line interface)[18]. The IM Dashboard is an easy-to-use web interface for non-advanced users to deploy predefined TOSCA templates that can be customized by specifying a set of input values. The IM Client is a Python-based command line tool and library that supports the entire functionality provided by the IM.

These APIs natively use the Resource and Application Description Language (RADL) [11] to describe the virtual infrastructure requirements and return the current infrastructure state. It addresses hardware (CPU number, CPU architecture, RAM size, etc.) and software requirements (applications, libraries, database systems, etc.). It includes all the configuration details needed for a fully functional and configured VM using Ansible's contextualization language.

The OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) simple YAML standard [12] is also supported by the IM to describe the topologies of the applications to be deployed. The usage of this standard provides several advantages because it is platform agnostic, independent of any specific cloud platform or technology, which reduces the risk of vendor lock-in in a public cloud scenario. Moreover, it provides reusability and interoperability of the produced templates, simplifying their maintenance.

The following sections provide more details about the main components of the Infrastructure Manager shown in Fig. 1. Each section gives a brief overview of the

[18] IM CLI - https://github.com/grycap/im-client

component and summarizes the recent enhancements made in each. The components of the platform, including the IM, are open-source. The IM documentation[19] provides a comprehensive description of the service. Finally, the external components used are also briefly described for the sake of completeness.

3.1 VM Image Selector

The Virtual Machine Image (VMI), which contains a virtual disk with a bootable operating system and certain applications, is required to deploy a VM. The VMI Selector helps the users find and set the desired image. To this end, it relies on third-party information systems where the VMIs are registered with some additional metadata for enhanced search capabilities. The IM initially supported our development, Virtual Machine Image Repository and Catalog (VMRC), and support for the AppDB information system has been recently added. As described in the work by Caballer et al. [11], a URI naming convention was defined to enable the definition of VM images. A new type of URI has been defined to point to VM images in EGI Cloud Compute so that users can choose specific images from the AppDB. Internally, these will be converted into the OpenStack VMI identified that will be used to launch the VM in the particular EGI Cloud provider. In this format, the protocol is set to "appdb", the address field is used to specify the site name, the path field is used to set the VMI name (as defined in AppDB) and, finally, the query part of the URI is used to specify the VO name. For example, the URI `appdb://SCAI/egi.ubuntu.22.04?vo.access.egi.eu` refers to the EGI Ubuntu 22.04 image[20] in the SCAI site[21] used for the EGI Access VO (Virtual Organization). The address field is optional, so the IM will return the specified VMI in all the EGI sites with support to the specific VO.

3.2 Configuration Manager

The Configuration Manager (CM) configures the provisioned infrastructure to satisfy the user's requirements by deploying the precise runtime environment, application software, and configuration, commonly known as contextualization.

Ansible is employed as the underlying DevOps framework to enable contextualization. Therefore, the users must define the configuration of their applications using Ansible playbooks. In the case of RADL, they can be embedded in the recipe, and with TOSCA, they have mainly been described as URLs to the playbook YAML files.

The IM enables, by design, the deployment of hybrid infrastructures (with VMs in different Cloud providers, see the work by Caballer et al. [24] for more details). This increases the variety of configuration issues, as there are different administration domains and network configurations. Therefore, the IM has added a set of features to enable contextualization in some configurations where the security rules do not enable having public IPs in all the involved sites or sites requiring the usage of an SSH proxy/bastion host to access the VMs. The first one is addressed by enabling reverse SSH tunnels among all the VMs and the front-end node, thus enabling this node as an Ansible control node to configure all the involved VMs without requiring them

[19]IM Documentation - https://imdocs.readthedocs.io/
[20]https://appdb.egi.eu/store/vappliance/egi.ubuntu.22.04
[21]https://appdb.egi.eu/store/site/scai

```
network private_net (
    proxy_host = 'user@proxy.host.com' and
    proxy_key = '-----BEGIN RSA PRIVATE KEY-----
...
-----END RSA PRIVATE KEY-----'
)
```

```
priv_network:
  type: tosca.nodes.indigo.network.Network
  properties:
    network_type: private
    proxy_host: proxy.host.com
    proxy_credential:
      user: user
      token_type: private_key
      token: |
        -----BEGIN RSA PRIVATE KEY-----
        ...
        -----END RSA PRIVATE KEY-----
```

Table 2 How to specify an SSH Proxy host in RADL (left) and TOSCA (right).

```
system node (
  ...
  disk.0.applications contains (
    name="ansible.collections.community.crypto"
  ) and
  disk.0.applications contains (
    name="ansible.roles.grycap.kubernetes"
  )
)
```

```
software:
  type: tosca.nodes.SoftwareComponent
  artifacts:
    crypto_collection:
      file: community.crypto
      type: tosca.artifacts.AnsibleGalaxy.collection
    kubernetes_role:
      file: grycap.kubernetes
      type: tosca.artifacts.AnsibleGalaxy.role
```

Table 3 How to specify Ansible Roles and Collections in RADL (left) and TOSCA (right).

to have public IP addresses. The second one is addressed by introducing the support for specifying an SSH proxy/bastion host. In this case, the user has to specify it in the infrastructure description. In particular, the network definition describes that this network can only be accessed using the specified host. Table 2 shows how to describe it in RADL and TOSCA.

Ansible Roles and Collections allow to distribute contributions from various Ansible Developers. To help include them in the playbooks used in the infrastructure description, the IM has introduced a new way to specify the roles or collections needed, and it will be in charge of installing them before the playbook execution. In the case of roles, a special application requirement must be added, starting with: `ansible.roles` as shown in the following examples. Similarly, `ansible.collections` enable the specification of collections. In the same way, they can be set using TOSCA to define the cloud topology. In this case, two new artifact types have been defined: `tosca.artifacts.AnsibleGalaxy.role` and `tosca.artifacts.AnsibleGalaxy.collection` that can be added to any node type reflecting the need to use the specified role/collections. In the examples shown in Table 3, the Ansible Galaxy role is called `grycap.kubernetes` (to deploy a Kubernetes cluster) and the `community.crypto` (with a set of modules to help in SSL certificate management) collection is specified.

3.3 Cloud Connector

The Cloud Connector layer provides a homogeneous interface to provision the VMs in the cloud providers. It has been designed using a plug-in scheme to ease its extension. We currently provide plug-ins for on-premises Cloud Management Platforms (e.g. OpenNebula, OpenStack and CloudStack); public Cloud providers (e.g. Amazon EC2, Microsoft Azure, Google Cloud, Linode, Orange, Open Telekom Cloud, Cloud & Heat); federated environments such as EGI Cloud Compute or FogBow, and serverless providers/platforms such as AWS Lambda or OSCAR.

This layer was initially designed with a simple API with six functions (as shown in Caballer et al. [11]) to provide the basic functionality needed by the IM: Launch a VM, Terminate a VM, Get VM information, Stop a VM, Resume a VM and Modify VM. The implementation of these functions depends on the underlying Cloud. For example, connectors have been implemented from scratch to access the APIs of OpenNebula and OSCAR. Other connectors use some existing Python SDKs such as Boto (for Amazon EC2) or SCAR[22] (for AWS Lambda), and others use the Apache Libcloud library to facilitate the development of the connector, as is the case of OpenStack and Linode.

The Cloud Connector layer has been extended with some functions to enable the management of VMI snapshots (create and delete). They are used in some tools such as EC3 (Elastic Cloud Compute Cluster) [25] to create snapshots of the VMI once it has been fully configured to use it as a base image for future deployments of the same VM in order to reduce the time required to configure subsequent infrastructures. It has also been extended with additional functions required for federated Clouds (e.g. EGI Cloud Service) to enable querying cloud sites to gather information about the available images and quotas. They are used by the IM Dashboard to help the user select the VMI to deploy their infrastructures and to choose the cloud provider considering the available quotas, thus avoiding the deployment of application architectures in overloaded cloud sites.

3.4 External Components

3.4.1 Ansible

Ansible is an open-source software suite that enables software provisioning, configuration management, and application deployment functionality. It uses an agent-less solution based on OpenSSH for transport (with other transports and pull modes as alternatives) and a human-readable YAML language to describe the configuration steps. It has the concept of a control node and a managed node. The control node is where Ansible is executed from, for example, where a user runs the `ansible-playbook` command. Managed nodes are the devices being automated. The control node must be a Linux-compatible node, but the managed nodes can be almost any operating system with Python installed. This includes Red Hat Linux, Debian, Ubuntu, macOS, FreeBSD, Microsoft Windows, and more.

Furthermore, it offers Ansible Galaxy, Ansible's official hub for sharing Ansible content: roles and collections. Galaxy provides pre-packaged units of work known

[22]SCAR - https://github.com/grycap/scar

to Ansible as Roles and Collections. Roles can be referenced from Ansible PlayBooks. There are roles for provisioning infrastructure, deploying applications, and other common tasks. The new Collection format provides a comprehensive package of automation that may include multiple playbooks, roles, modules, and plugins.

3.4.2 Virtual Machine Image Catalogs

Virtual Machine Image Repository and Catalog (VMRC)

VMRC[23] is a catalog of Virtual Machine Images (VMIs) that enables users (and/or Cloud administrators) to index the VMIs of the different Cloud Management Platforms (such as OpenNebula or OpenStack) or on public Clouds (such as Amazon Web Services) together with the appropriate metadata that describes their hardware and software features.

Applications Database (AppDB)

AppDB[24], a central service that stores and provides public information about software solutions, publications, etc., is also used to index the images used by the EGI Cloud Compute service, thus facilitating the integration with the EOSC ecosystem.

AppDB is divided into two main sections: the Software Marketplace and the Cloud Marketplace. On the one hand, the Software Marketplace allows to register any kind of software, namely applications, tools, workflow frameworks and instances, science gateways, middleware products, etc., and it uses the repository-related capabilities offered for delivering and distributing binary software artifacts. On the other hand, the Cloud Marketplace allows users to register and publish Virtual Appliances (VA), search for software that is available as a complete solution in the form of a VA, and select VAs and make them available to the EGI Cloud Compute sites and resource providers that support their VO, for users granted with the role of 'VO manager'.

3.4.3 Cloud Credentials Storage

Cloud credentials are sensitive data that must be managed carefully. These are sent to the IM service in the authorization header using secure HTTPS calls. However, the IM can also directly retrieve these credentials from an external secrets storage such as Hashicorp Vault.

Vault[25] is an identity-based secrets and encryption management system. A secret is anything that requires tightly controlled access, such as API encryption keys, passwords, and certificates. Vault provides encryption services that are gated by authentication and authorization methods. Using Vault's UI, CLI, or HTTP API, access to secrets and other sensitive data can be securely stored and managed, tightly controlled (restricted), and audited.

Vault provides centralized key management to simplify encrypting data in transit and stored across clouds and data centers. It can encrypt/decrypt data stored elsewhere, essentially allowing applications to encrypt their data while storing it in the primary data store.

[23] VMRC - https://www.grycap.upv.es/vmrc
[24] AppDB https://appdb.egi.eu/
[25] Vault - https://www.vaultproject.io/

The Vault service must be configured to enable the JWT authentication method used by the EGI Check-in OIDC provider, thus enabling any EGI Check-in user to store their credentials. EGI Check-in is a proxy service that is a central hub connecting federated Identity Providers (IdPs). Since EGI Check-In interfaces with the major academic/social IdPs, there is no entry barrier to access the service, which can be freely used without requiring a specific registration beforehand.

Any authenticated user can directly access the Vault service to manage their credentials. But the IM Dashboard also allows the user to add these credentials using the UI, facilitating the process and hiding the internal format of the credentials. It accesses the Vault service on behalf of the user that has logged in to the IM Dashboard. Similarly, the IM service can get user credentials from a Vault server. In this case, the user must provide the IM with the Vault endpoint and a valid EGI Check-in access token in the authentication header of the requests.

3.4.4 CLUES Elasticity Management

The IM supports multi-level elasticity management. On the one hand, horizontal elasticity allows adding or removing VMs to an infrastructure. On the other hand, vertical elasticity allows modifying the features of an individual VM, mainly in terms of CPUs, RAM and disk). The IM exposes these features to external frameworks such as CLUES or PROMINENCE, which provide the needed monitoring and decision-making systems to trigger the elasticity.

In particular, CLUES [26][26] is an open-source elasticity management system developed by our team, initially aimed for High-Performance Computing (HPC) clusters (either physical or virtual) but later extended to cloud-native systems such as container management platforms. The primary function of the system is to remove internal cluster nodes when they are not being used (scale in) and, conversely, to deploy them when they are needed (scale out). CLUES system integrates with the cluster management middleware, such as batch-queuing systems (e.g. Local Resource Management Systems such as SLURM) or container management platforms (e.g. Kubernetes or Nomad) using different connectors. It enables the creation of elastic cluster-like infrastructures that automatically scale in and out, in terms of the number of nodes, depending on the system workload. This creates the illusion of a real cluster without requiring investment beyond the actual usage. Therefore, this approach aims at delivering a cost-effective elastic Cluster as a Service on top of an IaaS Cloud.

3.4.5 SCAR

SCAR [10] is an open-source framework for transparently executing containers out of Docker images in both AWS Lambda and AWS Batch. It supports a High Throughput Computing Programming Model to create highly-parallel event-driven file-processing serverless applications that execute on customized runtime environments provided by Docker containers run on AWS Lambda. SCAR is integrated with API Gateway to expose an application via a highly-available HTTP-based REST API that supports synchronous and asynchronous invocations. It is also integrated with AWS Batch, a

[26]CLUES - https://www.grycap.upv.es/clues

managed service for elastic cluster-based computing. This way, the highly elastic capabilities of AWS Lambda can be used for the execution of large bursts of short requests. In particular, Lambda functions can support up to 3000 concurrent invocations, up to 10 GB of RAM, and up to 10 GB of ephemeral storage. Resource orchestration is delegated into AWS, even if provisioned concurrency can be pre-defined by the user to mitigate the first-invocation delay (cold start) at the expense of increasing the cost. In contrast, long-running executions are delegated to AWS Batch for their execution on virtual computing clusters, even with GPU support. These clusters can dynamically scale depending on the number of pending jobs, from a set of user-defined elasticity rules that can be pre-defined by SCAR. SCAR pioneered the introduction of container runtime support in AWS Lambda in 2017, years before AWS Lambda introduced the native support for the Docker runtime. With more than 585 stars in GitHub, SCAR is featured in the CNCF Cloud Native Interactive Landscape[27]. SCAR has been used to support the scalable inference from pre-trained AI models in AWS Lambda in both the DEEP Hybrid-DataCloud and AI-SPRINT European projects.

3.4.6 OSCAR

OSCAR [9] is an open-source platform to support the serverless computing model for event-driven data-processing applications. The IM automatically deploys and configures this component on multiple Cloud back-ends to create highly-parallel event-driven data-processing serverless applications that execute on customized runtime environments provided by Docker containers that run on an elastic Kubernetes cluster. With the ability to deploy OSCAR clusters in minified Kubernetes distributions such as K3s, event-driven workflows across the computing continuum can be executed since they can also run on constrained devices such as Raspberry Pis. While OSCAR can be configured on an existing Kubernetes cluster via Helm charts[28], it can be easily deployed via the IM, which provisions the Virtual Machines on behalf of the user and uses Ansible Roles to configure them as an OSCAR cluster.

With OSCAR, users upload files to MinIO[29], an object storage server with an Amazon S3-compatible API that acts as a data storage back-end. This action automatically triggers the execution of parallel invocations to a service responsible for processing each file. Output files are delivered into a data storage back-end for the user's convenience. The user only specifies the Docker image and the script to be executed inside a container created out of that image to process a file that will be automatically made available to the container. The deployment of the computing infrastructure and its scalability is abstracted away from the user. OSCAR also allows deploying scalable services invoked via HTTP-based synchronous requests to perform low-latency requests. These are handled by Knative, which orchestrates the requests into a set of pods whose number can automatically scale or always be available depending on a user-defined configuration. OSCAR is used for scalable inference across the continuum in the AI-SPRINT and AI4EOSC European projects. OSCAR provides similar event-driven data-processing capabilities of SCAR, but instead of running on AWS

[27]SCAR in the CNCF Cloud Native Interactive Landscape - https://landscape.cncf.io/serverless?selected=scar
[28]OSCAR Helm charts - https://github.com/grycap/helm-charts/tree/master/oscar
[29]MinIO - https://min.io/

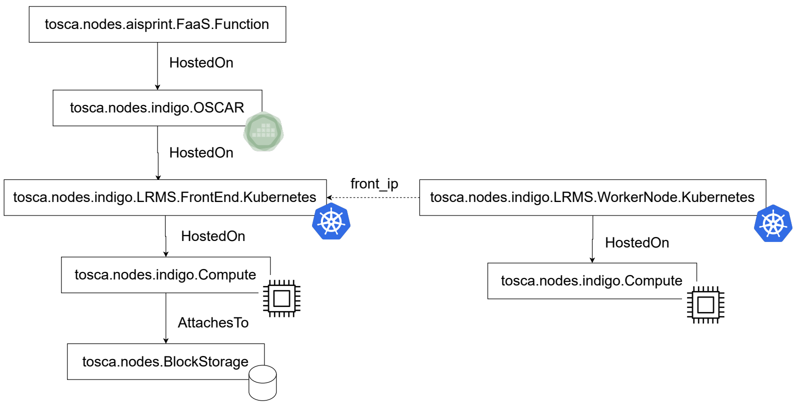
tosca.nodes.aisprint.FaaS.Function
HostedOn
tosca.nodes.indigo.OSCAR
HostedOn
tosca.nodes.indigo.LRMS.FrontEnd.Kubernetes
front_ip
tosca.nodes.indigo.LRMS.WorkerNode.Kubernetes
HostedOn
tosca.nodes.indigo.Compute
AttachesTo
tosca.nodes.BlockStorage
HostedOn
tosca.nodes.indigo.Compute

Fig. 2 TOSCA model to describe an OSCAR Function/Service running on a Kubernetes cluster.

Lambda, it runs on scalable Kubernetes clusters, which can be provisioned on any Cloud platform/provider, thus being Cloud-agnostic.

3.4.7 FDL

To support the deployment of data-driven workflows of serverless functions that require complex data processing in OSCAR and SCAR a YAML-based Functions Definition Language (FDL)[30] [8] was defined to specify the requirements for each function and how they are linked. It defines the functions to be dynamically created across the computing continuum. Two top-level resources are defined in an FDL document:

- Functions/Services, which are created in a Cloud provider and are assigned a name, a certain amount of computing resources together with a shell script that will be executed, as part of the function invocation, inside a container, created out of a specific Docker image that may available in a container registry such as Docker Hub or GitHub Container Registry. The function will be triggered whenever a file is uploaded to a specific folder within a storage provider, and the shell script will process the data file.
- Storage Providers become sources of events for input data processing and store the output data results from a function invocation. Using the input storage provider from another function as output from a function, a precedence relationship is established among them, and a data-driven link is created.

4 Extending the IM for the Computing continuum

The IM has been extended to support the deployment through TOSCA templates of FaaS services across the computing continuum. For this, we have relied on the serverless capabilities of the two open-source developments from our research group, described in the previous section OSCAR (for on-premises FaaS solutions) and SCAR (to interact with FaaS public providers).

[30]FDL - https://docs.oscar.grycap.net/fdl/

4.1 Supplementing TOSCA for the Computing Continuum

The TOSCA standard enables the definition of new custom types, thus supporting adding new components to the templates. Using this feature of TOSCA, we have defined new custom types for FaaS services to translate FDL concepts to TOSCA. Fig. 2 shows an object diagram of the full FaaS TOSCA template:

The FaaS function is deployed on (HostedOn relation) top of the OSCAR platform, which is also "HostedOn" a Kubernetes cluster. It is composed of one front-end node and a set of worker nodes configured to connect with the front-end node IP. All the Kubernetes nodes are "HostedOn" a Compute node and, in particular, the front-end node, has also attached a BlockStorage that is used by the persistent volumes. Notice that the namespaces used in the figure correspond to the research projects where these custom types are currently being used. For example, in the `tosca.nodes.aisprint.FaaS.Function` box, the `aisprint` namespace directly refers to the AI-SPRINT project[31]. Since the original FDL uses the YAML format, the translation into TOSCA is straightforward. The full definition of this node type and all the associated data types can be found in the custom types definition[32]. As an example, Listing 1 shows the definition of the anonymisation function used as the first step of the workflow specified in the use case described in section 5.

Edge devices are managed via OSCAR clusters. For example, a cluster of Raspberry PIs would be managed by an OSCAR cluster and services can be deployed on top, which are dynamically deployed by the Infrastructure Manager, as described in the TOSCA template.

Listing 1 Anonymisation Function in TOSCA

```
anon_service:
  type: tosca.nodes.aisprint.FaaS.Function
  properties:
    name: radiomicsanon
    memory: 128 MB
    cpu: 1.0
    image: grycap/radiomics:anonymise_arm64
    script: |
      FILE_NAME=$(basename $INPUT_FILE_PATH)
      OUTPUT_FILE=$TMP_OUTPUT_DIR/A_$FILE_NAME
      python anonymise.py -i $INPUT_FILE_PATH -o $OUTPUT_FILE
    input:
      - storage_provider: minio.default
        path: radiomicsin
    output:
      - storage_provider: minio.oscar-radiomics
        path: radiomicsin
    storage_providers:
      minio:
        oscar-radiomics:
          access_key: minio
```

[31] AI-SPRINT - https://www.ai-sprint-project.eu/

[32] https://github.com/grycap/ec3/blob/tosca/tosca/custom_types.yaml

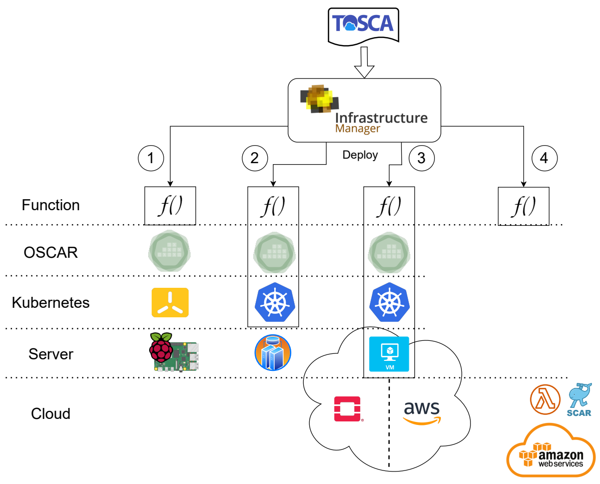

Fig. 3 Computing Continuum Scenarios.

```
                endpoint:
                  concat:
                    - 'https://minio.'
                    - get_input: cluster_name
                    - '.'
                    - get_input: domain_name
                region: us-east-1
                secret_key: { get_input: minio_password }
        requirements:
          - dependency: prep_service
```

As shown in previous sections, the FDL enables the definition of FaaS services together with their relationship to create data-driven serverless computing workflows. But this requires the OSCAR clusters to be previously deployed and properly configured. We have also defined new TOSCA custom types to include FaaS services in a TOSCA cloud topology for the user to define not only the FaaS services but also all the underlying software and infrastructure using TOSCA, that is, the OSCAR clusters. Furthermore, the IM service has been extended to support these new custom types, thus enabling the FaaS deployment in the computing continuum. IM is responsible for translating these new TOSCA custom types into the correct API calls to OSCAR to create the services with the defined features or contact AWS Lambda using SCAR to translate FDL concepts to AWS Lamba API calls.

Notice how the expressivity of the newly created TOSCA custom types is related to the functionality exposed by the underlying execution engines employed, in this case, OSCAR and SCAR. Using OSCAR facilitates not only the execution along the computing continuum, where different OSCAR clusters can be deployed but also the migration among different Clouds since the deployment of the functions is not directly performed in a managed service from the underlying Cloud provider.

The IM supports four different computing continuum scenarios (see Fig. 3):

1. An already existing OSCAR cluster: This is the typical scenario of a set of edge devices (e.g. a Raspberry Pi cluster) or computers where OSCAR has been installed and configured on top of a minified version of Kubernetes (i.e. K3s, Kind, microk8s, etc.). In this case, only the FaaS services must be defined in the TOSCA topology, and the OSCAR API endpoint and the credentials must be provided to the IM, enabling the deployment of the FaaS service.
2. A set of nodes is already deployed (virtual or physical) with a base OS: This is the case of a set of edge computers (also known as fog nodes in the OpenFog Reference Architecture for Fog Computing [27]) or edge devices with a plain base OS with an SSH connection available. In this case, not only the FaaS service must be declared in the TOSCA topology but also the OSCAR and Kubernetes node types. Therefore, the IM will orchestrate the installation and configuration of Kubernetes and OSCAR (using Ansible roles and playbooks assigned by the TOSCA custom types) and finally deploy the FaaS services. SSH connection details must be provided to enable the IM to access the nodes and perform the software installation. It will obtain the OSCAR endpoint and credentials from the installation process to finally deploy the OSCAR services.
3. IaaS Cloud provider: In this case, regardless of the cloud provider chosen, the TOSCA topology must specify all the components depicted in Fig. 2, from the FaaS service to the VMs to deploy. Thus, the IM will be in charge of orchestrating the deployment of the full stack of components on the Cloud, which involves mainly three phases: i) create the Cloud resources, including VMs, storage, networks, and any kind of resource requested by the user/application, by contacting directly with the cloud provider (on-premises or public); ii) install and configure Kubernetes in all the VMs and then deploy OSCAR on top of Kubernetes by means of Ansible; and iii) create the FaaS services interacting directly with the OSCAR API.
4. FaaS Cloud provider: As in the first case, only deploying the FaaS service directly in AWS Lambda is required. Because this service allows users to abstract from the underlying computing infrastructure, only the FaaS services are defined in the TOSCA topology, and AWS Lambda credentials are specified in the authorization data sent to the IM, allowing the IM to create the Lambda functions on behalf of the user.

In these scenarios, function placement is defined statically in the TOSCA document (with the “HostedOn” relation) in cases 2 and 3. In cases 1 and 4, function placement is made by the IM, at deployment time, using the same approach defined in [11] based on the Cloud providers list sent by the user to create a topology.

5 Evaluation and Use Case

This section performs an assessment of the IM as a general-purpose TOSCA-based orchestrator. First, a performance analysis has been carried out to evaluate the time required to provision popular infrastructures, such as Kubernetes clusters, including elasticity capabilities to add and remove additional nodes. Second, a use case involving the use of AI/ML techniques for Rheumatic Heart Disease classification involving data pre-processing, preparation and analysis across the computing continuum. The

TOSCA document with the full description of the topology used in this use case can be found in the examples folder of the IM GitHub repository[33].

5.1 Assessment of Deployment Time

We performed an updated performance analysis of the IM, similar to the one included in the work by Caballer et al. [11], to assess the deployment time on multiple Cloud providers/frameworks. However, in this case, a Kubernetes cluster has been used as the deployed infrastructure, since it is the most popular platform deployed through the IM in the EGI Cloud Service. This section shows the time needed to deploy a set of small or medium-sized Kubernetes clusters on several IaaS Clouds: OpenNebula (ONE), EGI Cloud Compute and Amazon Web Services (AWS). It also shows the ability to manage the elasticity provided by the IM and the time needed to add and remove working nodes (Virtual Machines) of the cluster. For the sake of reproducibility, the description of the complete performance assessment results and the underlying computing platforms have been made available in Zenodo[34]. The time needed to deploy the infrastructure is decomposed into the following steps:

- *VMs accessible*: Time elapsed until the SSH server is accessible in the master VM. This step requires the VM to be created and start running and the OS of the VMs to boot and start the SSH server.
- *Ansible configured*: After the VM is accessible, this is the time needed to install and configure Ansible in the master node. This is a relatively simple process that involves downloading the software and a small list of requirements, installing them, and copying all the recipes needed to configure the infrastructure.
- *Fully configured*: After Ansible is configured, this process involves the installation of all the needed software packages to install Kubernetes in all the nodes and the whole configuration process. It also applies a set of YAML files to configure a set of basic applications into the Kubernetes cluster (ingress controller, metrics-server, nfs-client-provisioner, ...). This process is made simultaneously in all the nodes, so it may cause some bottlenecks in the network or in the disk access. Also, certain synchronization is needed among the master and the worker nodes: e.g., the worker nodes cannot join the master node until it is properly initialized.

The *VM Addition* test includes the time needed to create the VM, the booting process to have the SSH server active, the configuration of the added node, and the reconfiguration of the rest of the nodes. In the rest of the nodes, Ansible playbooks are executed again, but since the Ansible modules are idempotent, they only check that the current configuration is correct, thus making no changes or just minor ones (e.g. adding new node IP in /etc/hosts file) without any significant overhead. The *VM Removal* test includes the time needed to terminate the VM and the reconfiguration of the rest of the nodes. As in the previous case, this implies no changes or just minor ones without significant overhead.

Table 4 shows the time needed in each individual step. These times are the average values of three tests performed in each case. As shown in the results, the average

[33]https://github.com/grycap/im/blob/master/examples/tosca_radiomics.yaml
[34]IM Evaluation - https://zenodo.org/record/7426579

	5 Nodes (ONE)	5 Nodes (EGI)	5 Nodes (AWS)	10 Nodes (ONE)	10 Nodes (AWS)	20 Nodes (ONE)	30 Nodes (ONE)
VMs Accessible	1:06	1:10	1:02	2:16	1:09	2:34	1:50
Ansible Conf.	2:01	1:44	1:56	2:00	1:58	2:16	2:27
Fully Conf.	7:42	6:44	7:04	10:15	9:16	9:59	14:19
Total Time	10:49	9:38	10:02	14:31	12:23	14:49	18:36
VM Addition	5:38	4:26	6:55	5:54	6:50	7:07	6:18
VM Removal	2:06	2:27	3:15	2:41	4:00	4:31	7:27

Table 4 Deployment time (in minutes) for the Infrastructure Manager to provision a Kubernetes cluster of several sizes in different Clouds.

time needed to deploy a small or medium-sized fully-functional Kubernetes cluster is between 10 or 20 minutes and the most time-consuming step is the configuration since it requires to install a relatively large list of packages in every node and performs the Kubernetes configuration. The time required to add a new node is slightly lower since just one node must be totally configured, and the other nodes just need to add the new one to the configuration. The reconfiguration of the rest of the nodes may require a longer time in the case of larger infrastructures. However, in this case, the results show that the time needed to add a VM for 30 nodes infrastructure is slightly lower than the smaller ones (10 and 20). It is not the expected behavior and can be explained by an unusual saturation of the cloud platform during the tests. Finally, node removal is the quickest operation since terminating a VM can be done very quickly, and it only needs to remove the node from the configuration of the other nodes. The results show that the VM removal operation time increases a bit when the size of the infrastructure grows. This is caused by the reconfiguration made to the rest of the nodes, and the larger the number of nodes, the longer it is needed to reconfigure them.

The time needed to add a node to the virtual infrastructure is reasonably long. If deployment time is an issue (e.g., dealing with an unpredicted workload), the configuration process could be reduced by using a pre-configured Virtual Machine Image (VMI) with some (or most) of the software requirements.

5.2 Use Case: Deployment Across the Computing Continuum

To demonstrate and validate the new IM capabilities for deployment across the computing continuum, we have chosen a real use case involving a pilot application on Medical Imaging Biomarkers based on artificial intelligence and machine learning techniques. This application (described in detail in the work by Blanquer et al. [28]) focuses on the early diagnosis of Rheumatic Heart Disease (RHD). Nowadays, RHD is the most frequent cardiovascular disease in children and young adults under 25 years old. An early diagnosis of the disease has proven to be relevant for improving the patient's quality of life. Screening programs can detect this disease in children, but early detection is currently limited since no single test accurately provides an early diagnosis of RHD. Therefore, RHD remains the leading cause of heart valve disease in the developing world. The usage of machine learning classifiers for assisting in the screening of RHD images can facilitate early diagnosis and treatment of such diseases, reducing morbidity and treatment costs and, at the same time, improving patient outcomes and quality of life.

The application utilizes texture analysis and filtering techniques to identify frames with significant data from medical imaging videos. These frames are then processed to extract relevant image features that can be used to classify rheumatic heart disease into normal, definite, or borderline categories. The application workflow[35] comprises the following steps:

- *Frame Splitting and Anonymization*: an essential action when using sensitive data like medical data. In this step, the frames are extracted from the videos during the anonymization process, and a black mask is applied in the right-upper corner to hide any sensitive data. Also, an automatic classification into Doppler and anatomical images is made in this step.
- *View Classification*: Not all the frames have the same weight for the final screening. Only the frames that relate to the parasternal long-axis view have proven to be relevant for accurate classification. Since there is no annotation on the view metadata, this should be estimated through an automatic classifier based on CNNs (Convoluted Neural Networks), previously trained for this purpose. This step uses tools such as Keras or TensorFlow that can be configured to use GPU to accelerate the computation.
- *Colour-based Segmentation*: To obtain trustworthy information from Doppler images, it is necessary to analyze only the color distribution and avoid any potential source of noise. For this purpose, color-based segmentation is performed through k-means clustering to extract a mask containing only the color information.
- *Texture Analysis*: Afterwards, key features must be extracted from the images to be fed into the classifier. For that purpose, first and second-order texture analyzes are applied to characterize the images by their spatial variation of pixel intensities. First-order analysis extracts features such as the mean, median, variance, kurtosis or skewness of the pixel distribution along the image. Second-order provides characteristics such as contrast, dissimilarity, homogeneity, or correlation, among others. Besides these features, blood velocity information is also obtained by processing the color distribution in the Doppler images.
- *Features Classification*: Finally, a classifier has been built through machine learning techniques using all the information extracted in the previous steps. This classifier has been trained according to the instructions previously described and can provide a classification (normal, definite or borderline RHD) for each patient's video.

After analyzing the application, we can identify three different computing scenarios to cover the whole pipeline. Thus, these steps have been grouped into three phases to be executed in different layers of the computing continuum (Fig. 4):

- Phase 1. Data pre-processing (anonymization): this phase involves *Frame Splitting and Anonymization* and it needs to be executed in the edge device (close to where the data are generated) to allow a secure process of the original data with sensitive information. Moreover, sensitive data cannot typically cross the boundaries of the organization to comply with regulations like the GDPR (General Data Protection

[35] https://github.com/eubr-atmosphere/radiomics

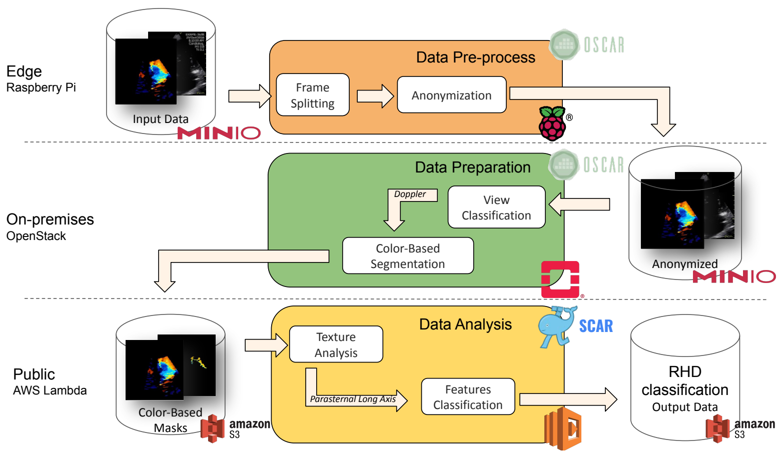

Fig. 4 Use case phases and the associated computing platform.

Regulation)[36]. Once the images are anonymized, they can be moved to the next phases that can be executed outside the boundaries of the data owner.

- Phase 2. Data preparation: This phase groups the *View Classification* and the *Colour-based Segmentation* steps. These steps enable the speedup of the execution using GPUs, so it will be executed in the on-premises cloud where flavors with GPUs are available.
- Phase 3. Data analysis: This phase includes *Texture Analysis* and *Features Classification*. The application used in this final phase does not support using GPUs, and it will be executed in AWS Lambda to profit from its high level of parallelism.

The next subsections specify the details of the platform used and the data involved in the experiment and present the results of the execution of this application.

5.2.1 Platform specification

As described previously, three main phases were identified where each one is more appropriate to be executed in a specific layer of the edge-to-cloud computing continuum, as depicted in Fig. 4. To this end, we have set up three different computing platforms distributed across this continuum:

- Raspberry Pi cluster: composed of one front-end and three worker nodes, all of them using the Raspberry Pi 4 model B (with 4 cores and 4 GB of RAM). This platform is pre-configured with K3s (a minimalistic Kubernetes distribution) and OSCAR. It is in charge of executing the first phase of the use case, performing the first phase, i.e. data pre-processing, including frame splitting and the anonymization of the videos, close to the origin of the data.
- OpenStack on-premises cloud platform: used to deploy a virtual cluster involving five VMs, with one front-end (4 CPUs and 4 GB of RAM) and four worker nodes (8 CPUs and 8 GB of RAM). All of the nodes use an Ubuntu 20.04 vanilla image. The IM

[36]GDPR - https://gdpr-info.eu/

automatically deploys all the cloud resources and configures the required software: Kubernetes and all the components of the OSCAR stack. This infrastructure is in charge of executing the second phase of the application, i.e. the data preparation that includes the view classification and color-based segmentation. The hardware resources have been empirically estimated through basic application performance profiling.

- AWS Lambda: we also involve the usage of a FaaS provider to perform the third phase of the use case, i.e., the data analysis which includes texture analysis and features classification. We have used AWS Lambda with functions configured with 1 GB of RAM, all of them deployed in the North Virginia (us-east-1) region. The run-time has been set up for using a container image previously prepared[37] and uploaded to AWS Elastic Container Registry (ECR) to make it available in the Lambda environment.

Although we have involved three different computing platforms, the Raspberry PI cluster was physically located in the same data center as the OpenStack infrastructure, and the virtual cluster deployed on top of it, for the sake of reproducibility during the execution of the case study. However, this does not affect the results and validation performed, since spreading the first two computing platforms across distributed infrastructures would only affect the network latency. OSCAR is being used both in the Raspberry Pi cluster (where it was installed beforehand) and the OpenStack on-premises cloud (where it was dynamically deployed beforehand), while SCAR is employed to facilitate the usage of AWS Lambda.

5.2.2 Data

The data used in these tests is a subset of 100 exams from the ones used in the work of Blanquer et al. [28]. The data comes from the Program of Rheumatic Valve Disease Screening (PROVAR - Programa de Rastreamento da VAlvopatia Reumática) initiative. PROVAR [29] is a collaboration between the Universidade Federal de Minas Gerais and the Telehealth Network of Minas Gerais, in Belo Horizonte, Brazil, and the Children's National Health System, in Washington DC, USA. The PROVAR team conducted a prospective cross-sectional study in Brazilian primary and secondary schools between October 2014 and December 2015 [30].

These 100 exams have an average of 14 videos each, with a total of 1,379 videos, from which:

- 90 exams are classified as normal (with a total of 1,217 videos);
- 5 exams are classified as borderline RHD (with a total of 82 videos); and,
- 5 exams are classified as definite RHD (with a total of 80 videos).

The video length ranges from 1 to 3 seconds, with a rate between 13 frames/second (Doppler videos) and 20 frames/second (anatomical videos).

[37] https://scar.readthedocs.io/en/latest/image_env_usage.html#use-already-prepared-ecr-images

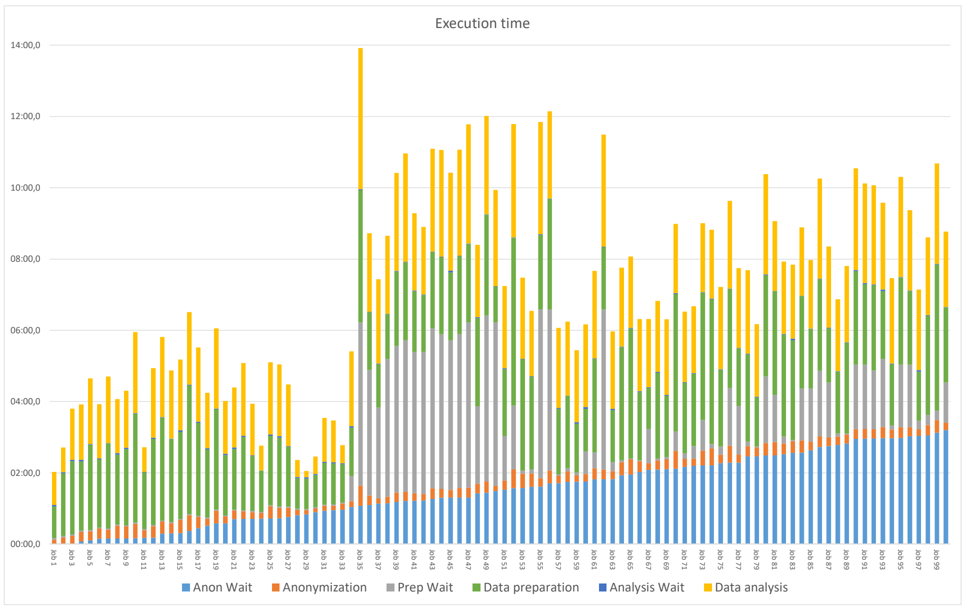

Fig. 5 Execution time of the use case. The X axis represents each executed job. The Y axis represents the elapsed time in each phase (format 'mm:ss,0').

5.2.3 Results

This section shows the results obtained after executing the 100 exams described above in the computing continuum scenario presented in subsection 5.2.1.

In the first step, we deployed and properly configured the whole infrastructure. This process has taken 12 min 3 s in total, including the creation of the VMs in the cloud provider (OpenStack) and the configuration of the OSCAR FaaS platform. The time needed to deploy the OSCAR service in the Raspberry Pi cluster and the Lambda function in AWS are not shown since they are done in parallel with the cluster creation, and the time required is considerably lower (less than a minute). Specifically, the time needed for each configuration step is the following:

- Create and wait for the VMs to boot in the OpenStack cloud provider (2 min 20 s).
- Configure Ansible in the front-end node (1 min 10 s).
- Configure the infrastructure (9 min), which involves:
  - Initial general configuration (1 min 15 s).
  - Install and Configure Kubernetes (5 min 5 s).
  - Install and Configure OSCAR (1 min 20 s).
  - Create OSCAR services (1 min 20 s).

Once the infrastructure is ready, we can start the execution of the use case application. As depicted in Fig. 4 the execution starts uploading the input files to a MinIO bucket deployed in the Raspberry Pi cluster that automatically triggers the execution of the full workflow. Fig. 5 shows the execution time of the total use case where 100 jobs (one job per exam, which involves an average of 14 videos to be processed) have been submitted and executed in parallel on the computing platform. One can also observe the time spent by each job in each one of the three phases of the execution. As labeled in the graph legend, we have represented in orange the first phase (data pre-processing, i.e., anonymization), which involves frame splitting and anonymization executed in the Raspberry Pi cluster; in green, the second phase (data preparation), which performs the view classification and the color-based segmentation steps on top of the virtual cluster deployed in OpenStack; and finally in yellow, we have depicted the third phase (data analysis), which involves texture analysis and features classification, that is executed in AWS Lambda.

Moreover, since the computing resources are limited, especially in the infrastructure used for running the first phase of anonymization, which represents the devices on the edge, Fig. 5 also shows the waiting time of each job. In light blue we see the waiting time of the jobs executing Phase 1; in grey, the waiting time for the jobs in Phase 2; and in dark blue, the waiting time in Phase 3 for each job. These times appear due to insufficient resources to process all the incoming jobs. In the case of the Raspberry Pi cluster, 12 parallel jobs (4 per node) are supported. In the case of the OpenStack OSCAR cluster, this is a total of 32 jobs (8 per node). Finally, in the case of AWS Lambda, the parallelism is large enough to execute all the incoming jobs (a peak of 31 simultaneous jobs), so the waiting time is minimal and barely appreciated in the figure.

Table 5 analyzes the whole execution of the use case, presenting the average, standard deviation (SD), maximum, minimum, and accumulated time for each phase, separating the execution time from the waiting time. The total execution time of the use case is set by the last finished job (in the graph, Job 35), which is 13 min 55,1 sec. Considering a scenario in which these jobs are executed sequentially, one after another, this would ideally require 7 h 31 min 53 s (the accum. execution time in Table 5), without considering any waiting time. Thanks to the concurrency supported by the FaaS platforms, the execution time was reduced to less than 14 min. From the results, we can also confirm that the most limiting phase is the first one, executed in the edge, due to the low computing capacity of these devices. The second phase is the most compute-intensive one, and its performance may vary depending on the number of nodes that compose the cluster and the computing capacity of the nodes (which can be specified in the TOSCA file). In the third phase, AWS guarantees a timely execution without significant delays, as long as the number of jobs does not exceed the default Lamba concurrency limit of 3000 executions[38]. The variability in the duration of each job, which can be noticed in Fig. 5, is determined by both the execution time, since the number of videos that compose the job, as well as their duration and number

[38] https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html

	Waiting Time				
	Avg	**SD**	**Max**	**Min**	**Accum.**
1-Anonymization	1min 31,9s	56,3s	3min 12,1s	0s	2h 33min 7,4s
2-Data preparation	1min 8,2s	1min 35s	4min 44,1s	1,4s	1h 53min 39,9s
3-Data analysis	1,5s	0,7s	3,2s	0,4s	2min 25,1s
				Total	4h 29min 12s
	Execution Time				
	Avg	**SD**	**Max**	**Min**	**Accum.**
1-Anonymization	16,4s	5,4s	34,3s	6,1s	27min 21,1s
2-Data preparation	2min 9,5s	46,2s	4min 42,2s	51,2s	3h 35min 49,5s
3-Data analysis	2min 5,2s	39s	4min 57,6s	10,7s	3h 28min 42,8s
				Total	7h 31min 53s

Table 5 Analysis of the use case time results.

of frames, affect this value, and the waiting time, which directly depends on the workload of the underlying computing infrastructure, as seen in the Standard Deviation (SD) values in Table 5.

Finally, it can be noticed that the Kubernetes scheduler follows a random selection of queued jobs, where some jobs that arrived at the cluster at the end of the execution (i.e. Job 94) are scheduled earlier than other previous jobs (i.e. Job 35). This has no impact on the final results, but it is not the standard behavior in other resource management platforms where the default job selection strategy used is FIFO (First In, First Out), as is the case of SLURM.

Notice that the whole execution starts when the video that the user wants to analyze is uploaded to the MinIO storage system in the OSCAR cluster of Raspberry Pis. After that, the execution is performed without the user needing to interact with the application or the infrastructure during the execution process. The execution is automatically orchestrated by OSCAR on top of the resources previously configured by the IM, allowing the integrated solution to deploy and execute applications in the computing continuum.

The ability to combine resources from multiple heterogeneous platforms (IaaS on-premises Cloud, public FaaS services) and disparate computer architectures (amd64 and arm64) allows to efficiently accommodate the execution of data-driven serverless workflows across the computing continuum thanks to the use of SCAR and OSCAR. Using the Infrastructure Manager to perform the automated provisioning of the OSCAR clusters and the dynamic deployment of both the OSCAR services and the Lambda functions via SCAR simplifies the process for the user.

6 Conclusions

This paper has introduced the new developments in the Infrastructure Manager (IM). This TOSCA-based orchestrator provides a unified management approach for application deployment, management, and scaling in a cloud-to-edge computing continuum scenario. These extensions have supported the deployment through TOSCA templates across the computing continuum of both on-premises FaaS services via OSCAR and public FaaS services such as AWS Lambda via SCAR. The IM leverages the TOSCA standard to provide a flexible and scalable runtime orchestrator that can handle the heterogeneity of edge devices and the dynamic nature of edge computing environments. We have validated the IM effectiveness with an actual use case focused on the early diagnosis of Rheumatic Heart Disease (RHD), showing that it abstracts the executions of applications in the computing continuum regarding deployment, resource configuration, and orchestration, maximizing application performance. Thus, we have proven the benefits of using the IM in cloud and edge computing environments, such as improved application performance (due to the use of FaaS in the continuum to exploit greater parallelism), reduced deployment costs, and enhanced support for data-driven serverless workflows across the computing continuum.

Future work includes extending the IM to interact with additional resource providers, especially additional FaaS services such as Google Functions or Microsoft Azure Functions. Moreover, with the uptake of TOSCA 2.0, additional alignments are expected to be developed in the IM to accommodate the changes introduced in newer versions of the specifications. Finally, re-configuration of dynamically provisioned infrastructure across the computing continuum opens up new avenues for research, which requires integration with QoS-aware mechanisms that trigger the reconfiguration process depending on the state of certain metrics such as workload increase, etc. For example, data ingestion at the edge of the network may require local preprocessing. This can be eventually supplemented with computing from an external Cloud to cope with temporary increased data processing requirements at the expense of an increased economic cost. This is the case, for example, in Agriculture 4.0 scenarios that require foliage analysis for decision-making on fully-connected tractors (a use case from the AI-SPRINT project).

Acknowledgements

Grant PID2020-113126RB-I00 funded by MCIN/AEI/10.13039/501100011033. Also, Project PDC2021-120844-I00 funded by MCIN/AEI/10.13039/501100011033 and by the European Union NextGenerationEU/PRTR. This work was partially supported by the project AI-SPRINT "AI in Secure Privacy-Preserving Computing Continuum" which has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant 101016577 and by the AI4EOSC "Artificial Intelligence for the European Open Science Cloud" that has received funding from the European Union's Horizon Europe Research and Innovation Programme under Grant 101058593. Finally, Amanda wants to thank the support received by the Universitat Politècnica de València for the project "Extensión de modelos de computacin en cloud en entornos confiables y altamente distribuidos", from the PAID-10-20 postdoctoral program.

References

[1] Ren, J., Yu, G., He, Y., Li, G.Y.: Collaborative cloud and edge computing for latency minimization. IEEE Transactions on Vehicular Technology **68**(5), 5031–5044 (2019)

[2] Jansen, M., Al-Dulaimy, A., Papadopoulos, A.V., Trivedi, A., Iosup, A.: The SPEC-RG Reference Architecture for the Compute Continuum. arXiv (2022). https://doi.org/10.48550/ARXIV.2207.04159 . https://arxiv.org/abs/2207.04159

[3] Deng, S., Zhao, H., Fang, W., Yin, J., Dustdar, S., Zomaya, A.Y.: Edge intelligence: The confluence of edge computing and artificial intelligence. IEEE Internet of Things Journal **7**(8), 7457–7469 (2020)

[4] Schleier-Smith, J., Sreekanti, V., Khandelwal, A., Carreira, J., Yadwadkar, N.J., Popa, R.A., Gonzalez, J.E., Stoica, I., Patterson, D.A.: What serverless computing is and should become: The next phase of cloud computing. Communications of the ACM **64**(5), 76–84 (2021)

[5] Momcheva, I.: Working with the hubble space telescope public data on amazon web services. Astronomical Data Analysis Software and Systems XXVII **523**, 671 (2019)

[6] Muhammad, W., Esposito, F., Maimaitijiang, M., Sagan, V., Bonaiuti, E.: Polly: A tool for rapid data integration and analysis in support of agricultural research and education. Internet of Things **9**, 100141 (2020) https://doi.org/10.1016/j.iot.2019.100141

[7] Cinaglia, P., Vázquez-Poletti, J.L., Cannataro, M.: Serverless computing for rna-seq data analysis. In: 2022 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pp. 2175–2181 (2022). IEEE

[8] Risco, S., Moltó, G., Blanquer, I.: Serverless workflows for containerised applications in the cloud continuum. Journal of Grid Computing **19**(3) (2021) https://doi.org/10.1007/s10723-021-09570-2

[9] Pérez, A., Risco, S., Naranjo, D.M., Caballer, M., Moltó, G.: On-premises serverless computing for event-driven data processing applications. In: 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), pp. 414–421 (2019). https://doi.org/10.1109/CLOUD.2019.00073

[10] 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

[11] Caballer, M., Blanquer, I., Moltó, G., 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

[12] Palma, D., Rutkowski, M., Spatzier, T.: Tosca simple profile in yaml version 1.0. OASIS Committee Specification **1**, 20 (2016)

[13] Foundation, E.S., Academies, A.E.: The European Code of Conduct for Research Integrity. European Science Foundation, Berlin, Germany (2011)

[14] Santana-Perez, I., Silva, R.F., Rynge, M., Deelman, E., Pérez-Hernández, M.S., Corcho, O.: Reproducibility of execution environments in computational science using semantics and clouds. Future Generation Computer Systems **67**, 354–367 (2017)

[15] Ullah, A., Dagdeviren, H., Ariyattu, R.C., DesLauriers, J., Kiss, T., Bowden, J.: Micado-edge: Towards an application-level orchestrator for the cloud-to-edge computing continuum. Journal of Grid Computing **19**(4), 1–28 (2021)

[16] Tusa, F., Clayman, S.: End-to-end slices to orchestrate resources and services in the cloud-to-edge continuum. Future Generation Computer Systems **141**, 473–488 (2023) https://doi.org/10.1016/j.future.2022.11.026

[17] Wurster, M., Breitenbücher, U., Képes, K., Leymann, F., Yussupov, V.: Modeling and automated deployment of serverless applications using tosca. In: 2018 IEEE 11th Conference on Service-oriented Computing and Applications (SOCA), pp. 73–80 (2018). IEEE

[18] Yussupov, V., Soldani, J., Breitenbücher, U., Leymann, F.: Standards-based modeling and deployment of serverless function orchestrations using bpmn and tosca. Software: Practice and Experience **52**(6), 1454–1495 (2022)

[19] Dehury, C.K., Jakovits, P., Srirama, S.N., Giotis, G., Garg, G.: Toscadata: Modeling data pipeline applications in tosca. Journal of Systems and Software **186**, 111164 (2022)

[20] Caballer, M., Antonacci, M., Šustr, Z., Perniola, M., Moltó, G.: Deployment of elastic virtual hybrid clusters across cloud sites. Journal of Grid Computing **19**, 4 (2021) https://doi.org/10.1007/s10723-021-09543-5

[21] Caballer, M., Chatziangelou, M., Calatrava, A., Moltó, G., Pérez, A.: IM integration in the EGI VMOps Dashboard. In: EGI Conference 2017 and INDIGO Summit 2017 (2017). https://indico.egi.eu/event/3249/contributions/7473/

[22] Salomoni, D., Campos, I., Gaido, L., Lucas, J.M., Solagna, P., Gomes, J., Matyska, L., Fuhrman, P., Hardt, M., Donvito, G., *et al.*: Indigo-datacloud: A platform to facilitate seamless access to e-infrastructures. Journal of Grid Computing **16**, 381–408 (2018)

[23] Lahiff, A., Witt, S., Caballer, M., La Rocca, G., Pamela, S., Coster, D.: Running htc and hpc applications opportunistically across private, academic and public clouds. In: EPJ Web of Conferences, vol. 245, p. 07032 (2020). EDP Sciences

[24] Caballer, M., Antonacci, M., Šustr, Z., Perniola, M., Moltó, G.: Deployment of

elastic virtual hybrid clusters across cloud sites. Journal of Grid Computing **19**(1), 4 (2021)

[25] Caballer, M., de Alfonso, C., Alvarruiz, F., Moltó, G.: Ec3: Elastic cloud computing cluster. Journal of Computer and System Sciences **79**(8), 1341–1351 (2013) https://doi.org/10.1016/j.jcss.2013.06.005

[26] Alvarruiz, F., Alfonso, C., Caballer, M., Hern'ndez, V.: An energy manager for high performance computer clusters. In: 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications, pp. 231–238 (2012). https://doi.org/10.1109/ISPA.2012.38

[27] OpenFog Consortium Architecture Working Group, A., *et al.*: Openfog reference architecture for fog computing. OPFRA001 **20817**, 162 (2017)

[28] Blanquer, I., Brasileiro, F., Brito, A., Calatrava, A., Carvalho, A., Fetzer, C., Figueiredo, F., Guimarães, R.P., Marinho, L., Meira, W., Silva, A., Alberich-Bayarri, Camacho-Ramos, E., Jimenez-Pastor, A., Ribeiro, A.L.L., Nascimento, B.R., Silva, F.: Federated and secure cloud services for building medical image classifiers on an intercontinental infrastructure. Future Generation Computer Systems **110**, 119–134 (2020) https://doi.org/10.1016/j.future.2020.04.012

[29] Lopes, E.L., Beaton, A.Z., Nascimento, B.R., Tompsett, A., Dos Santos, J.P., Perlman, L., Diamantino, A.C., Oliveira, K.K., Oliveira, C.M., Nunes, M.d.C.P., *et al.*: Telehealth solutions to enable global collaboration in rheumatic heart disease screening. Journal of telemedicine and telecare **24**(2), 101–109 (2018)

[30] Nascimento, B.R., Beaton, A.Z., Nunes, M.C.P., Diamantino, A.C., Carmo, G.A., Oliveira, K.K., Oliveira, C.M., Meira, Z.M.A., Castilho, S.R.T., Lopes, E.L., *et al.*: Echocardiographic prevalence of rheumatic heart disease in brazilian schoolchildren: Data from the provar study. International journal of cardiology **219**, 439–445 (2016)