



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Serverless Computing Strategies on Cloud Platforms

December 2020

Author: Diana María Naranjo Delgado

Directors: Dr. Ignacio Blanquer Espert
Dr. Germán Moltó Martínez

Acknowledgements

First of all I want to thank my parents for always being there for me and supporting me in all the decisions I have made throughout my life. To my sister, who is my role model, for encouraging me to embark on this completely new adventure for me. To my husband for always being by my side and making my days happy just by his presence.

I would like to thank all my GRyCAP colleagues for their support throughout this thesis and my career in the research group. Especially all my gratitude to Ignacio Blanquer and Germán Moltó for trusting me from the first day and providing me with all the tools and help necessary for the development of this research. With all of you I will be eternally grateful.

I want to express all my gratitude to all my friends who in one way or another have always supported and encouraged me in difficult moments. I would also like to thank all the people who have participated in my training as a person. Thanks to all of them I have been advancing in my life and I have reached this point.

Abstract

With the development of Cloud Computing, the delivery of virtualized resources over the Internet has greatly grown in recent years. Functions as a Service (FaaS), one of the newest service models within Cloud Computing, allows the development and implementation of event-based applications that cover managed services in public and on-premises Clouds. Public Cloud Computing providers adopt the FaaS model within their catalog to provide event-driven highly-scalable computing for applications.

On the one hand, developers specialized in this technology focus on creating open-source serverless frameworks to avoid the lock-in with public Cloud providers. Despite the development achieved by serverless computing, there are currently fields related to data processing and execution performance optimization where the full potential has not been explored.

In this doctoral thesis three serverless computing strategies are defined that allow to demonstrate the benefits of this technology for data processing. The implemented strategies allow the analysis of data with the integration of accelerated devices for the efficient execution of scientific applications on public and on-premises Cloud platforms.

Firstly, the CloudTrail-Tracker platform was developed to extract and process learning analytics in the Cloud. CloudTrail-Tracker is an event-driven open-source platform for serverless data processing that can automatically scale up and down, featuring the ability to scale to zero for minimizing the operational costs.

Next, the integration of GPUs in an event-driven on-premises serverless platform for scalable data processing is discussed. The platform supports the execution of applications as severless functions in response to the loading of a file in a file storage system, which allows the parallel execution of applications according to available resources. This processing is managed by an elastic Kubernetes cluster that automatically grows and shrinks according to the processing needs. Certain approaches based on GPU virtualization technologies such as rCUDA and NVIDIA-Docker are evaluated to speed up the execution time of the functions.

Finally, another solution based on the serverless model is implemented to run the inference phase of previously trained machine learning models on the Amazon Web Services platform and in a private platform with the OSCAR framework. The system grows elastically according to demand and is scaled to zero to minimize costs. On the other hand, the front-end provides the user with a simplified experience in obtaining the prediction of machine learning models.

To demonstrate the functionalities and advantages of the solutions proposed during this thesis, several case studies are collected covering different fields of knowledge such as learning analytics and Artificial Intelligence. This shows the wide range of applications where serverless computing can bring great benefits. The results obtained endorse the use of the serverless model in simplifying the design of architectures for the intensive data processing in complex applications.

Resumen

Con el desarrollo de la Computación en la Nube, la entrega de recursos virtualizados a través de Internet ha crecido enormemente en los últimos años. Las Funciones como servicio (FaaS), uno de los modelos de servicio más nuevos dentro de la Computación en la Nube, permite el desarrollo e implementación de aplicaciones basadas en eventos que cubren servicios administrados en Nubes públicas y locales. Los proveedores públicos de Computación en la Nube adoptan el modelo FaaS dentro de su catálogo para proporcionar computación basada en eventos altamente escalable para las aplicaciones.

Por un lado, los desarrolladores especializados en esta tecnología se centran en crear marcos de código abierto serverless para evitar el bloqueo con los proveedores de la Nube pública. A pesar del desarrollo logrado por la informática serverless, actualmente hay campos relacionados con el procesamiento de datos y la optimización del rendimiento en la ejecución en los que no se ha explorado todo el potencial.

En esta tesis doctoral se definen tres estrategias de computación serverless que permiten evidenciar los beneficios de esta tecnología para el procesamiento de datos. Las estrategias implementadas permiten el análisis de datos con

la integración de dispositivos de aceleración para la ejecución eficiente de aplicaciones científicas en plataformas cloud públicas y locales.

En primer lugar, se desarrolló la plataforma CloudTrail-Tracker. CloudTrail-Tracker es una plataforma serverless de código abierto basada en eventos para el procesamiento de datos que puede escalar automáticamente hacia arriba y hacia abajo, con la capacidad de escalar a cero para minimizar los costos operativos.

Seguidamente, se plantea la integración de GPUs en una plataforma serverless local impulsada por eventos para el procesamiento de datos escalables. La plataforma admite la ejecución de aplicaciones como funciones serverless en respuesta a la carga de un archivo en un sistema de almacenamiento de ficheros, lo que permite la ejecución en paralelo de las aplicaciones según los recursos disponibles. Este procesamiento es administrado por un clúster Kubernetes elástico que crece y decrece automáticamente según las necesidades de procesamiento. Ciertos enfoques basados en tecnologías de virtualización de GPU como rCUDA y NVIDIA-Docker se evalúan para acelerar el tiempo de ejecución de las funciones.

Finalmente, se implementa otra solución basada en el modelo serverless para ejecutar la fase de inferencia de modelos de aprendizaje automático previamente entrenados, en la plataforma de Amazon Web Services y en una plataforma privada con el framework OSCAR. El sistema crece elásticamente de acuerdo con la demanda y presenta un escalado a cero para minimizar los costes. Por otra parte, el front-end proporciona al usuario una experiencia simplificada en la obtención de la predicción de modelos de aprendizaje automático.

Para demostrar las funcionalidades y ventajas de las soluciones propuestas durante esta tesis se recogen varios casos de estudio que abarcan diferentes campos del conocimiento como la analítica de aprendizaje y la Inteligencia Artificial. Esto demuestra que la gama de aplicaciones donde la computación serverless puede aportar grandes beneficios es muy amplia. Los resultados

obtenidos avalan el uso del modelo serverless en la simplificación del diseño de arquitecturas para el uso intensivo de datos en aplicaciones complejas.

Resum

Amb el desenvolupament de la Computació en el Núvol, el lliurament de recursos virtualitzats a través d'Internet ha crescut granment en els últims anys. Les Funcions com a Servei (FaaS), un dels models de servei més nous dins de la Computació en el Núvol, permet el desenvolupament i implementació d'aplicacions basades en esdeveniments que cobreixen serveis administrats en Núvols públics i locals. Els proveïdors de computació en el Núvol públic adopten el model FaaS dins del seu catàleg per a proporcionar a les aplicacions computació altament escalable basada en esdeveniments.

D'una banda, els desenvolupadors especialitzats en aquesta tecnologia se centren en crear marcs de codi obert serverless per a evitar el bloqueig amb els proveïdors del Núvol públic. Malgrat el desenvolupament alcançat per la informàtica serverless, actualment hi ha camps relacionats amb el processament de dades i l'optimització del rendiment d'execució en els quals no s'ha explorat tot el potencial.

En aquesta tesi doctoral es defineixen tres estratègies informàtiques serverless que permeten demostrar els beneficis d'aquesta tecnologia per al processament de dades. Les estratègies implementades permeten l'anàlisi de dades amb

la integració de dispositius accelerats per a l'execució eficient d'aplicacions científiques en plataformes de Núvol públiques i locals.

En primer lloc, es va desenvolupar la plataforma CloudTrail-Tracker. CloudTrail-Tracker és una plataforma de codi obert basada en esdeveniments per al processament de dades serverless que pot escalar automàticament cap amunt i cap avall, amb la capacitat d'escalar a zero per a minimitzar els costos operatius.

A continuació es planteja la integració de GPUs en una plataforma serverless local impulsada per esdeveniments per al processament de dades escalables. La plataforma admet l'execució d'aplicacions com funcions serverless en resposta a la càrrega d'un arxiu en un sistema d'emmagatzemaments de fitxers, la qual cosa permet l'execució en paral·lel de les aplicacions segons els recursos disponibles. Este processament és administrat per un cluster Kubernetes elàstic que creix i decreix automàticament segons les necessitats de processament. Certs enfocaments basats en tecnologies de virtualització de GPU com rCUDA i NVIDIA-Docker s'avaluen per a accelerar el temps d'execució de les funcions.

Finalment s'implementa una altra solució basada en el model serverless per a executar la fase d'inferència de models d'aprenentatge automàtic prèviament entrenats en la plataforma de Amazon Web Services i en una plataforma privada amb el framework OSCAR. El sistema creix elàsticament d'acord amb la demanda i presenta una escalada a zero per a minimitzar els costos. D'altra banda el front-end proporciona a l'usuari una experiència simplificada en l'obtenció de la predicció de models d'aprenentatge automàtic.

Per a demostrar les funcionalitats i avantatges de les solucions proposades durant esta tesi s'arreglen diversos casos d'estudi que comprenen diferents camps del coneixement com l'anàlítica d'aprenentatge i la Intel·ligència Artificial. Això demostra que la gamma d'aplicacions on la computació serverless pot aportar grans beneficis és molt àmplia. Els resultats obtinguts

avalen l'ús del model serverless en la simplificació del disseny d'arquitectures per a l'ús intensiu de dades en aplicacions complexes.

Contents

Abstract	iii
Contents	xv
List of Figures	xvii
1 Introduction	1
1.1 Motivation	7
1.2 Objectives.	10
1.3 Methodology	11
1.4 Chapter Conclusions.	16
2 Background and State of the Art	17
2.1 Cloud Computing.	18
2.2 Containerized computing.	22

2.3 Learning Analytics in Cloud Computing	24
2.4 Serverless Computing	26
2.5 GPUs in the Cloud.	31
2.6 Chapter Conclusions.	33
3 Serverless architecture and CloudTrail-Tracker	35
3.1 Function as a Service Model.	36
3.2 CloudTrail-Tracker: A Serverless Platform for Enhanced AWS Usage Insights	40
3.3 Chapter Conclusions.	60
4 Availability of GPUs on Serverless On-Premises Platforms	63
4.1 Components used	64
4.2 Architecture	69
4.3 Use Case: Availability of GPUs on Serverless On-Premises Platforms	74
4.4 Chapter Conclusions.	89
5 Serverless services for machine learning model inference	91
5.1 Components used	92
5.2 Architecture	101
5.3 Use Case: Serverless services for machine learning model inference.	105
5.4 Chapter Conclusions.	121
6 Conclusions and Future Work	123
6.1 Conclusions.	123
6.2 Future Works	128
6.3 Scientific production and fundings for this thesis	129
6.4 Software Developments	135

Bibliography	139
Glossary	167

List of Figures

1.1	Popularity of the term “serverless computing” reported by Google Trends.	5
1.2	Number of publications per year in Google Scholar and Web of Science (WOS) with the keyword “serverless computing”.	5
1.3	Summary diagram of the methodology to follow in the development of the research.	12
1.4	Stages of development of the thesis.	15
3.1	General structure of an event-driven Serverless Architecture. . .	36
3.2	Architecture of CloudTrail-Tracker. Source: [95]	45
3.3	Cloud Computing training at the Universitat Politècnica de València, in Spain. Source: [95]	49
3.4	CloudTrail-Tracker start panel.	52

3.5	Details of services used by all users in the CloudTrail-Tracker <i>Dashboard</i>	53
3.6	Services used by a student in a certain period of time (CloudTrail-Tracker <i>Search by User</i> panel).	54
3.7	Percentage of completion for each hands-on lab for a specific student. Source: [95]	54
3.8	Missing actions for each hands-on lab for a specific student. Source: [95]	55
4.1	Overview of the rCUDA virtualization strategy.	68
4.2	OSCAR architecture to access external GPU resources through rCUDA. Source: [96]	70
4.3	Integration of rCUDA into the OSCAR architecture. Source: [96]	72
4.4	Processing pipeline for the use case. Source: [96]	75
4.5	Workflow of the selected use case on the OSCAR platform using remote GPUs with two different functions. Source: [96]	79
4.6	Workflow of the selected use case on the OSCAR platform using remote GPUs with all the code in one function. Source: [96]	80
4.7	Average time importing the required modules and loading the pre-trained model in the analyzed scenarios. Source: [96]	81
4.8	Comparison classifying different length videos in the analyzed scenarios. Source: [96]	82
4.9	Time segmentation according to the main execution phases in scenarios 3 and 4. Source: [96]	84
4.10	Execution times for processing up to 4 videos (180 frames) in scenario 3 and 4. Source: [96]	85

5.1	Typical AWS Batch architecture for job processing.	100
5.2	Architecture for the integration of Machine Learning models in AWS.	102
5.3	Settings tab to configure access to MinIO.	107
5.4	Simplified file processing workflow. Functions with different deployment methods selected, either on AWS or in a local cluster with OSCAR.	108
5.5	<i>Select Method of Deployment, Select Model</i> and <i>Upload Files</i> panels of the Web Interface.	110
5.6	Web interface for the status of jobs and user files.	111
5.7	Lambda function code to check the state of jobs through the API of AWS Batch.	112
5.8	Process to querying, from the web interface, the status of the jobs in AWS Batch.	113
5.9	Example of the result obtained using the Body Pose Detection model.	113
5.10	Execution times for 10 images in AWS Lambda.	114
5.11	Execution times for 10 images in AWS Batch.	115
5.12	Execution times for 10 images in OSCAR cluster.	116

Chapter 1

Introduction

The development of technologies that have emerged in recent times have allowed us to face and solve many scientific challenges, fundamentally those related to Big Data and Artificial Intelligence, two of the most popular innovative technologies in recent times. For this, it is extremely important to have data centers with great computing capacity and original problem-solving approaches.

The last few years have witnessed unprecedented advances in the field of Cloud Computing. This model refers to ubiquitous, convenient and on-demand network access to a shared group of computing resources. These resources can be networks, servers, storage, applications and services, and can be provisioned and launched quickly with minimal administrative effort or the interaction of the service provider [89].

The open range of possibilities of access to computing, storage and network resources in the Cloud, established a wide range of service models such as IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software

as a Service). The IaaS model (lowest level) basically constitutes a virtual provisioning of computing infrastructures such as storage, servers, networks, maintenance and support through the Cloud. The PaaS model (intermediate level) is a framework where the Cloud provider provides the client with a development environment with the necessary tools for the implementation of applications through a secure connection. On the other hand, the SaaS model (highest level) allows quick access to cloud-based web applications.

The investment in the use of resources in the Cloud is going up remarkably, with the forecast that by 2023 more than half of all the infrastructure spending of the IT (Information Technology) buyers will be in IaaS, and the computation IaaS in the public Cloud accounts for more than 25% of the computation in an average company [64]. Therefore, reducing costs in all fields of Cloud Computing will be a priority.

Virtualization is the combination of hardware and software that creates virtual machines (VM) and allows multiple operating systems to run on the same physical machine [80]. Virtual machines are an emulation of a physical computer and are considered the foundation of Cloud Computing. Each virtual machine includes an operating system, a virtual copy of the hardware required to run an application, as well as associated libraries and dependencies. This causes that the main limitations of virtual machines are aimed at their large size and the boot time of the operating system. This deployment system ensures greater security benefits by executing the processes inside the virtual machine with greater isolation from the processes running on the host system.

Containers are an abstraction of the application layer that packages all the code and dependencies required to run the application instead of creating a full virtual machine. Containers virtualize the operating system kernel that can be shared among multiple containers, each running as isolated processes in user space [141]. Containers have provided a better way for application delivery, as they are lighter and by being encapsulated processes that use the Linux kernel primitive, they provide a faster startup time than virtual machines [5]. When it comes to managing complex infrastructures,

the microservices approach, in which an application can be decomposed into smaller, loosely coupled, and independent services, allows systems to be scaled and deployed with greater ease and flexibility through the use of containers. In microservices-oriented architectures, containers allow a certain degree of isolation in terms of dependencies and configurations as each service is embedded in a custom environment.

Containers as an emerging technology in the Cloud, are presented as the most suitable solution to serve as a runtime environment where virtualization processes run quickly and lightly [83]. Most commercial Cloud Computing systems make use of containers to manage applications in isolated environments so that the effort and time required to scale the deployment is significantly reduced. The arrival of Linux containers together with the increase in process automation led to the emergence of other service models such as FaaS (Functions as a Service).

Serverless computing can be seen as the latest emerging Cloud execution model, where the Cloud provider dynamically manages resource allocation. One of the fundamental challenges in the transition to serverless computing is that applications must be designed in the form of functions, which requires the adoption of a microservices-based architecture, where processes are independent of each other. Microservices architecture became the main candidate to maximize the performance of Cloud applications [144].

Serverless computing, and in particular the FaaS model, have become a convincing paradigm for the deployment of applications in the Cloud, largely due to the recent change of architectures from enterprise applications to containers and microservices [49].

In the changing world of Cloud Computing, serverless computing and the FaaS model are presented as two new categories that, among other advantages, save money on infrastructure and, above all, reduce developer time. Although these terms are often used interchangeably, they are not the same. Serverless computing includes applications that use services from other providers, for

example, database management systems or API services [46]. These services are commonly known as BaaS (Backend as a Service) and are hosted in the Cloud where the provider is in charge of managing the servers and logic.

Now the FaaS model is considered the most recent category of serverless Cloud Computing services. In this model, application programming remains with the developer, but the server logic is run in stateless execution environment called functions. These functions are event triggered and fully managed by the Cloud service provider. These types of platforms allow developers to focus on the application logic and not on back-end management. One of the more attractive potentials is that the underlying provisioned computing infrastructure, allocated for the execution of the multiple invocations of the function, is dynamically resized by the Cloud provider.

An important element in the adoption of serverless computing by IT companies was their pay-as-you-go model of short-term computing resources as needed [74] [36]. This paradigm gives the customer the possibility of paying only for leased resources to the public Cloud provider, without worrying about the excessive provisioning of a service, whose popularity does not meet its original predictions [12]. The improved scalability in serverless computing is a fundamental element that has allowed its wide adoption. The applications created on these platforms are automatically scaled adapting to the demand, that is, it is not necessary to provision an approximate amount of resources since the resources are flexible and managed by the service provider.

Figure 1.1 shows the growing popularity of the search term *serverless computing* in the last 5 years reported by Google Trends¹. Also, Figure 1.2 shows the publications per year with the search keyword “serverless computing” in Google Scholar² and the Web of Science³ (WOS). These are indicators that demonstrate the increasing attention that serverless computing has generated lately in trade shows, meetings and in the research community in general.

¹Google Trends - <https://trends.google.es/>

²Google Scholar - <https://scholar.google.com/>

³WOS - www.webofknowledge.com



Figure 1.1: Popularity of the term “serverless computing” reported by Google Trends.

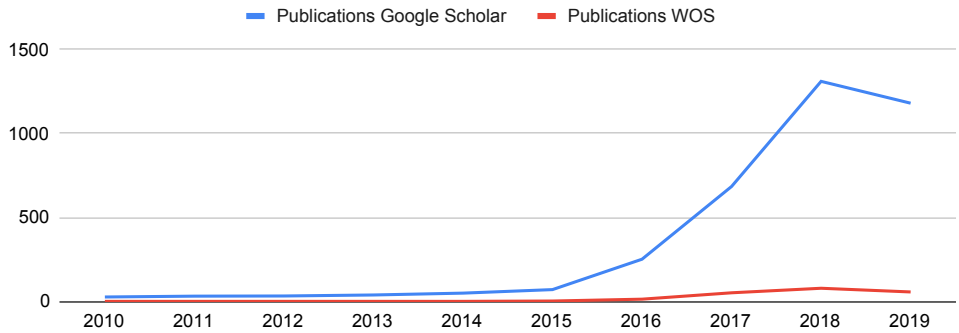


Figure 1.2: Number of publications per year in Google Scholar and Web of Science (WOS) with the keyword “serverless computing”.

Serverless computing offers the ability to define applications as a workflow of event-triggered functions that run without requiring the user to manage the necessary resources. Serverless platforms provide developers with a simplified programming model for their applications, where cost is reduced by charging for runtime rather than resource allocation.

When discussing about Cloud Computing and serverless computing, it is inevitable to cover the major public Cloud providers such as: AWS (Amazon Web Services) [7], GCP (Google Cloud Platform) [125] and Microsoft Azure [126]. These providers are able to offer users a large computing capacity along with storage, network resources and databases, among other features, through pay-per-use services. In addition, open-source CMP (Cloud Management

Platforms) such as OpenNebula [103] and OpenStack [26] adopt the main benefits of the Cloud in on-premises infrastructures.

Major public Cloud providers have included support for FaaS services for defining and executing functions. This is the case of AWS with AWS Lambda⁴, GCP with Google Functions⁵ and Microsoft Azure with Azure Functions⁶. Functions must be coded in a supported programming language and can be executed in response to certain events, such as uploading a file to a storage service such as Amazon S3 [6] (an object-based data store) or an HTTP request to an API Gateway [124].

When using the services of public Cloud providers there is a risk of becoming dependent on the services and products of that provider, a term known as “vendor lock-in”, since changing a technology or provider is costly. This phenomenon, together with the need to design new application architectures that embrace good practices in serverless computing, constitutes one of the greatest challenges that has slowed down the adoption of serverless technology [36]. With this aim, several open-source solutions have emerged, as is the case of OpenFaaS [45] and Knative [57].

In this sense, COPs (Container Orchestration Platforms) play a fundamental role as they make container management easier. Nowadays, it is possible to find several offers such as the case of Docker Swarm [134], Apache Mesos [10] and Kubernetes [79] to name a few. According to previous research [127] [90] where different container orchestration technologies are compared, Kubernetes is the main option to consider, if you have a large and complex deployment of infrastructures, since it has a large community of support and recognition. On the other hand, for small implementations it is recommended to use Docker Swarm for its simplicity and integration with Docker containers.

A link should be established between Cloud Computing, the development of serverless architectures and the analyses of large volumes of data. For example,

⁴AWS Lambda - <https://aws.amazon.com/lambda>

⁵Google Functions - <https://cloud.google.com/functions>

⁶Azure Functions - <https://azure.microsoft.com/services/functions>

in recent years, ML (Machine Learning) and AI (Artificial Intelligence) applications have been making their way into the world of Cloud Computing [32] [52] [60]. Cloud providers focus their efforts on including these types of services within their catalog, due to the high demand for these types of applications.

To meet the challenges posed by serverless technologies, this thesis aims to implement different serverless computing strategies that allow users to run scientific applications in public and private Clouds. This is coupled with the usage of accelerated computing devices that maximize performance in execution through the use of GPUs. This thesis focuses on proposing means to deploy Cloud applications transparently to the user, abstracting the configuration, management, and scaling details of the underlying infrastructure.

This chapter presents the scope of this doctoral thesis and the main lines of action. First, a brief motivation and description of the objectives to be achieved is explained. Finally, the methodology and the work plan followed are commented, to achieve the proposed objectives.

1.1 Motivation

As mentioned above, the rise of Cloud infrastructures and serverless technologies allows accessing large amounts of computing resources. These technologies are already being used in the enterprise, but their use is still limited in many areas of the scientific field where they could obtain great benefits.

The current computing landscape includes a large number of existing public Cloud providers as well as various on-premises CMP that can support containers and VMs. Because of this, it is necessary to identify serverless patterns for efficient data processing and provide support for accelerated hardware devices for efficient computing. In fact, there is a great challenge

in providing serverless computing with efficient and cost-effective capabilities of IaaS Cloud Computing.

The needs of computing resources for processing scientific applications [37] [21] have been covered by small local clusters that need to be installed and managed. With the emergence of virtualization technologies, the transition from local to virtual workloads was facilitated without substantial change. These clusters have limitations in terms of overheads in scalability and even in the maximum capacity itself, which is why public Cloud infrastructures are often used to mitigate these limitations. Virtual clusters implemented in a public Cloud achieve competitive price performance relationships, providing important benefits for organizations such as reducing management costs.

The balance between performance and cost is the fundamental factor to consider in the design of any architecture. There is a great challenge in providing flexible and cost-effective computing capabilities, so the architectures developed in this thesis are based on public and on-premises Cloud, for the implementation of systems and platforms capable of accelerating the execution of scientific applications and addressing larger problems. This computational strategy will be applied to two fields: learning analytics for educational Cloud Computing and intensive processing in complex ML models.

First, learning analytics studies the data generated by students in order to understand and optimize the learning process and the environment in which it occurs. In this sense, learning analytics is presented as an indispensable element in online learning with an impact in regulated learning [121]. In these learning environments we can identify two fundamental challenges that drive the development of this academic field: optimizing the analysis of the set of data related to learning, and the development of tools that allow visualizing in a simple way the data generated when carrying out the educational activities.

Second, Machine Learning is the area of artificial intelligence dedicated primarily to giving a computer the ability to learn without having been specifically programmed for each of the situations [25]. Years ago, this was

an unreachable challenge, but the development of new Artificial Intelligence paradigms and the availability of intensive computational solutions has allowed us to begin to face the challenge. Despite all the advances made in the field of artificial intelligence, much remains to be investigated. The limits of contemporary machine learning technology go through the complexity in the use of this type of model, where it is necessary to make it accessible to all types of users with and without experience, and in the computing infrastructure necessary for the training and the inference phases of the neural network systems, that make up this scientific application. Therefore, as part of this thesis, architectures have been designed to overcome these limits by integrating different machine learning models as use cases, executed on serverless platforms.

In ML applications, two types of computational models can be distinguished, one for training and the other for the inference phase. In the training of machine learning and artificial intelligence applications, fast response times are needed, making use of the most efficient hardware with parallel and heterogeneous computing techniques such as MPI and the use of fast interconnection technologies such as PCIe and InfiniBand [13] [132]. This computational model fits perfectly with the HPC paradigm.

In the inference phase of machine learning and artificial intelligence applications where high-volume and highly complex data processing is required, a winning solution is the combination of the FaaS model with the HTC (High Throughput Computing) [84]. Basically, the platform dynamically provisions the resources necessary for the execution of the applications, effectively exploiting all those available resources. Once the processing is finished, the platform is in charge of managing the resources if they are no longer needed. In terms of public Cloud providers, this translates into an economically profitable system and, in the case of on-premises Cloud, into a most efficient usage of resources, potentially leading to economical and energy savings.

For ML and AI applications where maximizing performance is required, using an execution strategy based on event-driven serverless computing to accelerate

times executions provides a cost-effective means of data processing. Integrating acceleration devices like GPUs (Graphics Processing Unit) into on-premises serverless platform can address these performance issues required in processing scientific applications.

1.2 Objectives

The main objective of this thesis is the design of different serverless computing strategies both in public and on-premises Cloud platforms, combined with the usage of accelerated computing devices, to efficiently tackle challenging data processing problems.

This general objective is broken down into several goals addressed in this thesis:

- First, conduct a study of container-based virtualization tools for HPC platforms. There are a wide variety of tools for container virtualization, and Docker stands out as one of the most widely used. In HPC environments the use of Docker has difficulties due to the need to run it with administrator privileges. Therefore, it is important to carry out a comparative analysis among other tools, taking into account certain metrics, to determine the behavior and limitations of different technologies in the context of HPC centers. This will allow to determine the appropriateness of the different container tools to be adopted as the execution runtime of the serverless platforms defined.
- Second, design a serverless application to perform real-time data processing of the activities taking place in an AWS shared account. This will allow to easily view the resources provisioned by the multiple users. It will also obtain aggregated information to control the resources used by users and the activities carried out by students who use the platform when carrying out educational activities in the Cloud.

- Third, integrate the use of specific accelerated devices like GPUs into a platform that supports serverless computing for scalable event-driven data processing that presents a multi-level elasticity approach.
- Fourth, integrate already trained ML models with a serverless platform to easily run predictions, based on these models, in a private and public Cloud, while featuring dynamic elasticity based on execution needs.

To this end, case studies made up of real scientific applications, will be carried out on public and on-premises Cloud platforms.

1.3 Methodology

Due to the highly changing nature of technologies related to Cloud Computing and its different service models, a work methodology based on identifying serverless computing challenges in technological development projects has been proposed. This will allow obtaining results in a timely manner and producing software to solve the identified challenges. This methodology proposes to divide the workload into small units that lead to the production of functional, documented and tested prototypes, whose functionality is increased iteratively.

Figure 1.3 shows a summary diagram of the methodology followed in the investigation. Firstly, it is necessary to carry out a bibliographic search and learn tools in the field of Cloud Computing and the orchestration of Cloud services, a key element for serverless computing. Once the basic knowledge is acquired, the key elements in the research on serverless computing in public and on-premises Clouds are identified. In this sense, it is important to find strategies to improve the performance of these platforms in scientific applications for intensive data analysis. Later, it is important to design solutions that solve those key points, where performance improvement can be obtained, resulting in proof-of-concept implementations. The results of these developments are evaluated in terms of cost and performance. Subsequently, the results are published in high impact scientific journals and conferences to

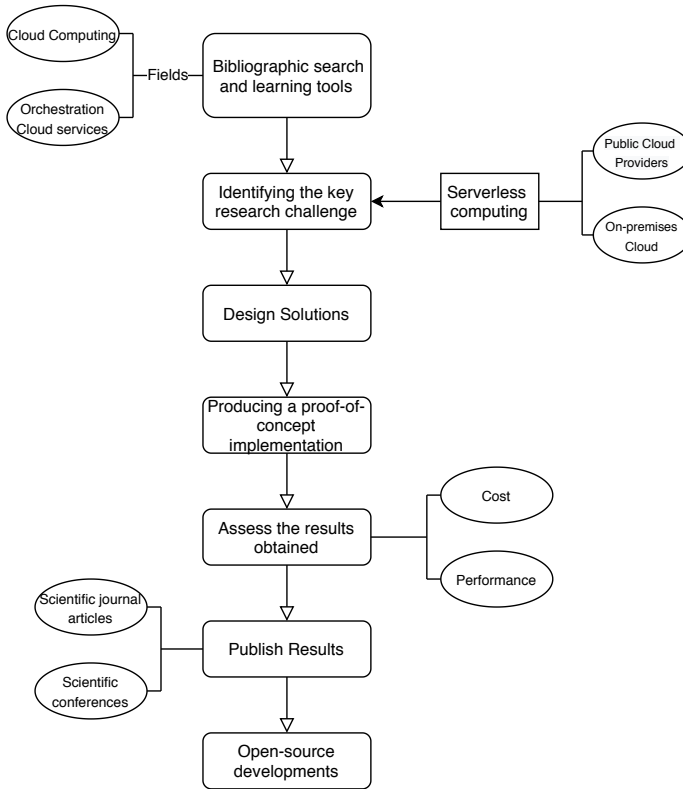


Figure 1.3: Summary diagram of the methodology to follow in the development of the research.

assess their acceptance and relevance, obtaining open-source software that can be used by other researchers.

1.3.1 Computing infrastructures

During the development of the thesis, different types of infrastructure will be used: computer clusters, private and public Clouds. The research group in which the thesis is carried out, GRyCAP⁷, contributes with three on-premises Cloud platforms that are currently used in production environments. The characteristics of these platforms are detailed below:

⁷GRyCAP - <https://www.grycap.upv.es/>

- The first infrastructure, called *Ramses*, has 11 nodes with a total of 22 Intel Xeon processors (for a total of 308 cores) and 896 GB of RAM, spread over 8 nodes with 64GB and 3 nodes with 128GB. All nodes have 10GbE connectivity and are connected to a SAN (Storage Array Network) with 16 TB in RAID6.
- The second infrastructure, called *OneCloud*, has 6 biprocessor nodes (Intel Xeon) with a total of 112 cores, 192 GB of RAM and Gigabit Ethernet connectivity. They are also connected to the previously mentioned SAN.
- The third infrastructure, called *Horsemen*, has 2 nodes. The first one features two 2.1 GHz Skylake Gold 6130s, 16 cores each, 768 GB of DDR4 @ 2666, 10 GbE and includes an Arria 10 GX115 8GB FPGA, a RADEON Instinct MI25, 16GB, a Tesla P40 24GB and a Tesla V100 32GB. The second node features the same processor and memory along with 4 Tesla V100 32GB.

In the first two platforms, OpenNebula 5.2 is used to manage the entire system, which in turn will use KVM (Kernel-based Virtual Machine) as hypervisor and the third platform uses OpenStack Rocky.

Regarding public Cloud providers, it is proposed to use the services of AWS. This decision was based on AWS being considered the leading public Cloud provider in the development and implementation of Cloud services. Today, AWS is the clear dominator of the Cloud market and is a pioneer in supporting serverless computing. For the development of the different environments, commercial range computers and open-source development frameworks were used.

Considering the objectives of the project and taking into account its duration, the work plan was organized in the following stages. Figure 1.4 shows a summary of the development of these stages over the duration of the research.

- A first stage was dedicated to studying 16 credits as prerequisites for admission to the Doctorate program in Computer Science. The subjects taken served as a basis to consolidate the knowledge, that would be put into practice in the development of the thesis.
- In a second stage, an exploration of solutions and external components was carried out. This step has been developed since the beginning of the research in December 2017 and culminates in the defense of the thesis. It is fundamentally based on being updated on the latest trends in the technologies developed for the purpose of the thesis, and thus using the most current technologies available in the development and implementation of the environments to be developed.
- Design of the environments. This stage is developed once basic knowledge is achieved and aims to propose novel designs that allow the initial objectives to be met.
- Prototype development. Once the architectures have been designed, the objective is to put them into practice, using the latest technologies and with practical use cases, that reflect the importance of the elements that make up the proposed architectures.
- During the months of May, June and July 2019, a research stay was carried out at the LIP⁸ (Laboratory of Instrumentation and Particle Physics) in Lisbon. The objective of the stay was to study *udocker* [55], a tool developed by LIP researchers to create containers in user space.
- Validation of the results. In this stage the main results obtained from the previous phases are analyzed and discussed taking into account the use cases implemented in the development of the prototypes.
- Dissemination of the results obtained. The dissemination of the results is carried out throughout the research process and consists of publishing

⁸LIP - <https://www.lip.pt/?section=home&page=homepage>

the results obtained in scientific journals and national and international conferences.

- Writing of the manuscript and defense of the thesis. At this stage, the preparation of a final report is carried out, setting out the results obtained during the duration of the research. It concludes with the oral defense of the results in front of a committee of experts on the subject.

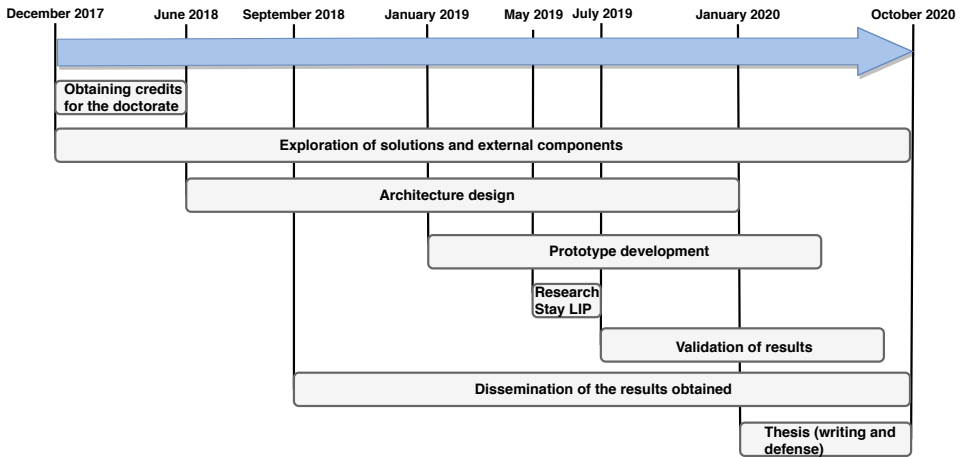


Figure 1.4: Stages of development of the thesis.

To cover all the proposed objectives, this thesis is based on inter-related chapters. First, Chapter 2 reviews the most recent research related to the context of the thesis. Possible solutions to problems related to container computing, serverless computing, and the use of GPUs in Cloud applications are discussed. In the following chapters, a reference is made to the proposed serverless computing strategies that enable the execution of scientific and educational applications. In Chapter 3, the general architecture of the FaaS model is first discussed, followed by the introduction of CloudTrail-Tracker, an AWS cloud-based serverless platform to process learning analytics. Chapter 4 refers to integrating acceleration devices into a serverless on-premises platform. Chapter 5 introduces a serverless platform to run the inference phase of machine learning and artificial intelligence models, also on AWS and in a

private Cloud. Finally, Chapter 6 summarizes the main contributions of the thesis, highlighting the future lines in which it can be improved and citing the scientific articles and publications produced as outcome of this research.

1.4 Chapter Conclusions

The elements that mark the beginning of the research were introduced in this chapter. The elements that motivated the development of the thesis are explained. Subsequently, the objectives that allow the orientation of the topic were analyzed and the procedures carried out for the development of the research were clearly and detailed described. The main phases through which the research passed since its inception were also analyzed and, finally, a summary of the structure presented by the thesis was made.

The work has produced a total of 6 contributions on international publications and conferences, including 2 articles in journals indexed in Q1 and Q2 of the Journal Citation Reports (JCR), and 1 communication listed in the CORE B, Class 2 category in the GII-GRIN-SCIE index. Section 6.3 develops more extensively all the scientific production generated in the development of the research.

Chapter 2

Background and State of the Art

The wide adoption of Internet-based Information Technologies has made it possible to develop of the anything-as-a-service (XaaS) paradigm. In the development of this chapter, the background that has allowed the adoption of this paradigm in the Cloud is addressed, emphasizing the technologies that represent the basis of serverless computing strategies that are part of this research. In addition, a study of previous research related to the scope of the thesis is carried out. The study of the information collected in this chapter allows identifying the advantages of the technologies used, as well as the main limitations that are the object of study of this research.

2.1 Cloud Computing

In the world of computing, the most widespread definition of the Cloud is provided by the National Institute of Standards and Technology (NIST) [99]:

“Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.”

In addition to the concept of Cloud Computing in [99], essential features, service models and deployment models are defined.

The basic characteristics defined are:

- *On-demand self-service:* A consumer can automatically provision resources when needed and with minimal human interaction from the service provider.
- *Broad network access:* Computing capabilities are available over the network and access is via standard mechanisms such as mobile phones, tablets, laptops, etc.
- *Resource pooling:* The computing resources will be available to all users through a multi-tenant platform, dynamically reassigned according to consumer demand. The client has no control over the exact location of the resources (e.g., storage, processing, bandwidth, and active user accounts), but can specify it at a higher level of abstraction such as country or region.
- *Rapid elasticity:* Provisioning of computing capabilities is done quickly and elastically according to demand.
- *Measured service:* The use of resources is automatically controlled and monitored transparently for the user and the service provider.

The service models identified by NIST are:

- *Software as a Service (SaaS)*: This model allows the user to use the applications that are hosted by the Cloud service provider. Access to applications is done through a client interface such as the web browser, and the consumer can only change some configuration parameters, but does not manage the virtual or physical infrastructure.
- *Platform as a Service (PaaS)*: This model is based on implementing within the Cloud infrastructure the applications created or acquired by the user, using programming languages and tools compatible with the provider. The user does not control or manage the infrastructure but has control over its applications and some configuration variables of the computing environment.
- *Infrastructure as a Service (IaaS)*: In this model the user can provision computing resources such as storage, processing, networks and other computing elements to deploy and execute arbitrary software. The service provider controls the physical infrastructure, granting the user very limited access to network components. The user controls the virtual infrastructure where he manages the operating system, storage and implemented applications.

Finally, the following deployment models are defined:

- *Private Cloud (also known as on-premises)*: The Cloud infrastructure is provisioned for the use of a single organization. It can be managed and operated by the organization, by a third party, or a combination of both.
- *Community Cloud*: This type of infrastructure is shared by several organizations within the same community that have common concerns such as: mission, security requirements, policies and compliance considerations. It can be managed by one or more organizations in the same community by a third party or by a combination of both.

- *Public Cloud:* These infrastructures are openly provisioned for the general public. They can be managed by a business, academic or government organization or a combination of both.
- *Hybrid Cloud:* This infrastructure is made up of two or more different Cloud (private, community, public) that have their own entities but coexist through a standardized technology that allows data and applications to be shared between them.

With the expansion of Cloud technologies, there are many providers within the service models described above. Currently, three providers stand out: AWS, GCP and Microsoft Azure. AWS is the pioneer in the development of Cloud infrastructures and was named in 2019 for the 9th consecutive year as a leader in Gartner’s Infrastructure as a Service (IaaS) Magic Quadrant [16] [87]. GCP and Microsoft Azure, originally focused on providing services that could be labeled as PaaS, although they have now incorporated services of all kinds into their catalog. In previous studies [76] [12] [62] [40], the main specifications of each of these providers are discussed and compared. In addition to these three major Cloud providers, other companies have been included in the world of Cloud Computing, as is the case of IBM¹ , Salesforce.com², Linode³, Alibab Cloud⁴, Oracle⁵ and others. All these platforms offer access to infrastructure in public Clouds and charging services through the pay-as-you-go model.

Cloud Computing has the potential to transform a large part of the IT industry by facilitating the deployment of applications through service models. Developers with innovative ideas for new Internet services no longer require capital investments in hardware or human expenses in maintaining the physical infrastructure. The development of Cloud Computing has meant that developers do not have to worry about wasting resources on a service whose popularity does not meet expected predictions, or insufficient provisioning of one that is becoming very popular. For this, Cloud infrastructures have the

¹IBM - <https://www.ibm.com/es-es/products/category/technology/cloud-computing>

²Salesforce - <http://www.salesforce.com/>

³Linode - <http://www.linode.com/>

⁴Alibaba Cloud- <https://eu.alibabacloud.com/>

⁵Oracle- <https://www.oracle.com/es/index.html>

capacity to grow the resources according to the needs of the application and reduce it when they are no longer required, this is what is defined as elasticity in the Cloud.

With the development of Cloud Computing, access to users of computing, storage and network resources has been provided with great flexibility in both public Cloud providers and on-premises Cloud infrastructure [30]. Many of the features that Cloud Computing presents justify its growth and popularity in the last decade.

Virtualization technologies are the key element in Cloud Computing. Virtualization is a technique that allows to abstract a computing resource (CPU, storage, network, memory) from the physical layer of that service, which allows the execution of multiple operating systems and applications on the same machine and with the same hardware, thus increasing its flexibility and use. These features make it one of the leading technologies in the efficient delivery of infrastructure solutions in Cloud Computing. [145].

Cloud platforms are based on the use of virtualization software (also known as hypervisors) for the execution of virtual machines on physical resources. Examples of these software are: KVM, XEN, VMWare, Hyper-V, etc. Virtualization can be implemented at different levels and, indeed, those mentioned above refer to hardware based virtualization. However, in recent years a technology that has caused great expectation is virtualization at the level of the operating system, also known as virtualization based on containers, to which we will refer later.

Virtualization technologies have enabled the development of scalable and multi-tenant Cloud infrastructures. Cloud Computing applications have reached virtually every technology sector from event driven conversational agents [18], to IoT (Internet of Things) [65], Artificial Intelligence [17] even Big Data processing [1] and education [38].

2.2 Containerized computing

The widespread adoption of containers has enabled to distribute and run services widely with a lightweight virtualization approach. Containers are software units that pack applications and all their dependencies, allowing them to run quickly and reliably from one computer environment to another [141]. The main reason for the success of containers is the flexibility and efficiency they offer to pack and run specific software, which has boosted the development of architectures based on microservices virtualization.

To achieve multi-tenant isolation, Linux containers can be run on VMs, this being the security boundary and additional resource allocation limit for applications. The main benefits of containers arise when they are used on bare metal, in order to obtain higher performance than VMs [2]. Containers are actually encapsulated kernel processes that use a layered storage system and do not pre-reserve memory space beyond active processes, requiring much less memory and disk space. Containers facilitate application development and distribution, hence the rapid growth of this technology and the interest of developers [11].

Among the different existing container platforms, Docker⁶ stands out as one of the most used solutions. It allows developers to encapsulate a software along with all the necessary dependencies to run, ensuring portability and execution regardless of the environment. This is possible through the use of advanced Linux features, namely *control groups* and *namespaces* isolation [122].

While it is true that Docker is the most widely used container technology, its model does not fit well with HPC platforms. HPC clusters are multi-user systems in which users should only have access to their own data and computing resources. The problem is that the Docker daemon runs with administrator privileges, so it would be possible for a Docker daemon vulnerability to be exploited by a container to escalate privileges.

⁶Docker - <https://www.docker.com>

Docker cannot safely use the host network stack and requires a separate network namespace, an inconvenience when using parallel applications that require network communication efficiently. The host network mode is not appropriate in Docker, as it does not cause isolation and adds to the risks of running in supervisor mode. Due to these limitations the use of Docker on multi-user systems is very restricted [55].

As a consequence of this, alternative solutions have been developed to mitigate the limitations of Docker in HPC environments. Some of these alternatives are: *udocker*⁷ [55], *Singularity*⁸ [81], *CharlieCloud*⁹ [113], *Shifter*¹⁰ [69] and *Podman*¹¹. As part of the research in the section 5.1.2 an analysis of these tools is performed, presenting a comparative table (Table: 5.1) of the main functionalities.

In the research conducted by Arango et al. [11] four environments are tested: LXC (Linux Containers), Docker, Singularity and bare metal, using different metrics such as MPI support, network traffic, GPU support and RAM performance. As a result, Singularity was determined to be more suitable for HPC application deployment than Docker or LXC.

A recent study by Canon and Younge [34] describes the current challenges and approaches in using containers for HPC applications. For the identified gaps in container integration in HPC environments the authors propose solutions such as implementing a container runtime library that manages specific devices and libraries between host and container images, plus dual-virtualization methods to avoid incompatibilities with previous and later versions of libraries. Although the authors demonstrate that these approaches enable the high-performance execution of containers at scale with adequate execution times, they report that there are still many gaps in research in this field.

⁷udocker - <https://github.com/indigo-dc/udocker>

⁸Singularity - <https://sylabs.io/singularity/>

⁹CharlieCloud - <https://github.com/hpc/charliecloud>

¹⁰Shifter - <https://github.com/NERSC/shifter>

¹¹Podman - <https://podman.io/>

In the work carried out by Jha et al. [70] an evaluation of the performance of Docker containers running heterogeneous HPC microservices is made, focusing mainly on how inter-container interference influences performance. Taking into account the results obtained, the authors plan to carry out the same analysis using other container technologies such as *udocker*, *Singularity* or *Socket*¹² [19].

Some research has focused on studying the advantages of using HPC containers, mainly analysing the performance when using the different virtualization technologies. As an example, the article Rudy et al. [123] analyses the performance overhead induced by the use of three different container technologies (Docker, Singularity and Shifter) by comparing it with native execution. Another example is the work done by Hu, Zhang, and Chen [61] where the CPU performance, memory and network bandwidth between Singularity and bare metal are evaluated in detail by means of different benchmarks. In both articles the results showed that with Singularity almost native performances are obtained.

The use of containers in computing offers significant benefits including a lightweight, consistent execution environment and low overhead to maintain and scale applications with high efficiency. This provides the landscape for simple and efficient implementation of scientific applications in HPC environments.

2.3 Learning Analytics in Cloud Computing

The last few years have seen unprecedented progress in the use of online educational platforms and MOOC (Massive Open Online Courses) with great success. This means that education is not exempt from the use of information technologies, and certainly not from the use of Cloud Computing.

For the analysis of the large amounts of data generated in online courses, a new research field called learning analytics emerges. Learning analytics is a

¹²Socket - <https://github.com/unioslo/socket>

technique that enables learning to be understood and optimized through the collection and analysis of general data [44].

The rise of automated data collection and processing techniques along with the rise of MOOCs have led to the development of learning analytics. As an example, the authors of [135] have designed LASyM, a learning analytics system that automatically detects students at risk of dropping out, based on the large amount of data generated by MOOC platforms.

Learning analytics has cleared the way for the use of learning dashboards to provide visual interpretation of student progress. The work of Verbert et al. [138] presents a review of dashboard applications to support students and teachers in online and classroom environments.

One of the most recent works developed in the field of visual learning analytics is that of Vieira et al. [139]. In this article the authors refer to the lack of studies that employ sophisticated visualizations and engage deeply with educational theories, a statement also supported by the authors of [72], who analyze learning panels from the students' point of view. However, although there are several learning panels in the literature, none address the field of Cloud Computing studies.

Shorfuzzaman et al. [130] presents a cloud-based mobile learning framework that utilizes Big Data analytics technique to extract values from huge volume of mobile learners' data. As a result, it is shown that learning analytics can greatly benefit from the analysis of large amounts of data using computing and storage resources provided by the Cloud.

In the work developed by Wong et al. [143], a review is made of the analytical intervention of learning in higher education from 2011 to 2018. In this study it is concluded that intervention practices have been more often focused on increasing student study performance, providing personalized feedback and improving student retention.

2.4 Serverless Computing

The growth in recent years of serverless computing is largely due to the dynamic allocation of computing resources managed by the Cloud provider, along with the fine-grained pay-as-you-go billing model. Serverless computing encompasses two different areas that overlap at the same time: BaaS and FaaS. Serverless since its inception was used to describe third-party applications and services hosted in the Cloud to handle server state and logic. In the literature this branch is known as Backend as a Service (BaaS) and is based on applications that use, for example, database systems accessible in the Cloud (Amazon DynamoDB¹³, Cloud Datastore¹⁴, Azure SQL Database¹⁵) or authentication services like AWS Cognito¹⁶.

The other branch of serverless computing is functions as a service, FaaS. This architecture allows the execution of applications through ephemeral execution environments triggered by events. These environments are created on the spot so that developers focus on the functionality of their application, and not worry about managing the infrastructure on which the function is executed.

Most major Cloud providers offer the FaaS model as part as their serverless infrastructure package such as AWS Lambda [8], Google Cloud Functions [56], Microsoft Azure Functions [91], Alibaba Cloud Function Compute [4], and IBM Cloud functions [63].

Some of the benefits of adopting the FaaS model for multiple scientific domains (e.g. computer graphics, cryptology, mathematics, and meteorology) are presented by Spillner et al. [131]. Considering some of the intrinsic features of the FaaS model like predefined function environments and short-lived executions, there are many scientific applications that can benefit from migrating to this model. Another study by Baldini et al. [20] analyses the

¹³Amazon DynamoDB - <https://aws.amazon.com/dynamodb>

¹⁴Cloud Datastore - <https://cloud.google.com/datastore>

¹⁵Azure SQL Database - <https://azure.microsoft.com/es-es/services/sql-database/>

¹⁶AWS Cognito - <https://aws.amazon.com/>

challenges, use cases and key characteristics of serverless platforms in the industry, including open-source projects.

There are works in the literature that successfully use the paradigm of function as a service. One of the pioneers in this field is the article published by Jonas et al. [73] who introduce the PyWren framework in order to perform Python-based distributed computing on AWS Lambda. Stateless features running in the Cloud represent a viable platform for users by eliminating the overhead of cluster management. Shortly afterwards, this study was expanded in the contribution made by Shankar et al. [128]. This article presents `numpywren` a system for linear algebra built on a serverless architecture. In addition, `LambdaPACK` is introduced, a domain-specific language to implement linear algebra algorithms that are highly parallel, evaluating the higher computational efficiency achieved and highlighting the limitations of the Cloud provider.

A recent study by Eismann et al. [42] compiles 89 use cases, from different sources, related to serverless platforms. Use cases are analyzed taking into account characteristics such as: workload, application, requirements and other general characteristics. The objective of this study is to establish a guide for the design of serverless applications that allow a better understanding of this paradigm. The authors refer to the usefulness of the study in academia and industry to stimulate the design of new architectures based on the comparative evaluation of these use cases.

Serverless computing is also used in environments for the analysis of large amounts of data, i.e. Big Data. The article developed by Giménez-Alventosa, Moltó, and Caballer [53] describes a high performance serverless architecture called MARLA¹⁷ to run MapReduce works on AWS Lambda using Amazon S3 as storage back-end. The SCAR¹⁸ (Serverless Container-aware ARchitectures) framework [108] also uses AWS Lambda as a platform to run general scientific applications based on Docker containers that are event-driven. Event Driven

¹⁷MARLA - <https://github.com/grycap/marla>

¹⁸SCAR - <https://github.com/grycap/scar>

Architecture (EDA) is a software model for application design based on event generation and handling. These architectures allow to increase the versatility of the system, hence they are my used in modern and distributed applications.

2.4.1 On-premises Serverless Frameworks

In the adoption of serverless architecture, an important element to consider is to avoid lock-in with a public Cloud provider. This term in Cloud Computing refers to the fact that the implementation of an application in a specific Cloud provider depends on the technology of that provider, and cannot be easily migrated to a different provider without substantial costs due to technical incompatibilities. [102].

Several open-source frameworks to build an on-premise local FaaS environment have become popular in the last years. Some of the most widely used are: OpenFaaS [45], Knative [57], Fission [48], Nuclio [100], Apache OpenWhisk [9], Oracle Cloud Fn [105] and Riff [112], to name a few. These platforms support the definition and execution of functions in response to events. Generally, the difference between them lies in the compatibility with different programming languages, the multiple sources of events and the use of a specific container orchestration platform, such as Kubernetes.

It is possible to find in the literature works related to these kind of frameworks. This is the case of the work carried out by Hendrickson et al. [59], one of the first in this field, where an open-source platform, called OpenLambda to build applications and web services using the serverless architecture is presented.

In the work carried out by Kaviani, Kalinin, and Maximilien [75], a common execution model is proposed. This model can bring us closer to a unified serverless platform based on Knative. This avoids the possibility of users being locked into a particular provider when they move their application to a serverless platform.

A more recent work [106] evaluates four serverless open-source frameworks such as Kubeless, Apache OpenWhisk, OpenFaaS and Knative in a

resource-constrained edge computing environment. The investigation presents some typical scenarios on the edge of a network IoT for the implementation of these frameworks. As a result, it is stated that in these scenarios, Kubeless surpasses the other frameworks in terms of response, time and performance, on the other hand, with OpenWhisk the worst performance is obtained. In addition, new research opportunities in this line are discussed.

In the work done by Li et al. [82], an analysis of the performance of the most popular open-source serverless platforms is made taking into account the design problems. The different elements that affect the performance of the different frameworks are analyzed and the conclusion is that simple automatic scaling based on resources or workloads is not adequate to meet the needs of serverless platforms.

2.4.2 Machine Learning and Artificial Intelligence in Serverless Computing

On the one hand, the use of ML and artificial intelligence applications has led to the adoption of these models as part of the services available on Cloud platforms. The importance of using predictive analysis to address a variety of socially and environmentally significant problems, requires the development of these models to be accessible to users with and without experience in cloud-based technologies [23].

Public Cloud providers have included machine learning and artificial intelligence applications within their services. An example of this is Amazon SageMaker¹⁹, a fully managed service to quickly create, train and implement machine learning models. These fully managed solutions involve the use of instances that are expensive for many use cases.

The article by Corral-Plaza et al. [39] offers an analysis of the main options for ML in the Cloud, focusing on Amazon Machine Learning and the BigML²⁰ platform. Another study by Ishakian et al. [68] evaluates the suitability of

¹⁹Amazon SageMaker - <https://aws.amazon.com/sagemaker/>

²⁰BigML - <https://bigml.com/>

a serverless computing environment for the inference of large neural network models in the AWS Lambda environment. The work carried out by Rausch et al. [116] proposes a serverless platform to build and operate state-of-the-art artificial intelligence applications. Different use cases are discussed that illustrate the challenges in building serverless applications in edge Cloud scenarios.

In another study, conducted by Yan et al. [146], a chatbot is implemented on a serverless on-premises platform using the OpenWhisk framework. The solution is based on functions that are used to coordinate cognitive services. The serverless model implemented in this research improves the extensibility of the chatbot with the automation of the services of scalability and maintenance of the infrastructure.

The work carried out by Rao Divate Kodandarama, Danish Shaikh, and Patnaik [115] studies the feasibility of the machine learning model inference phase on a serverless platform. In conclusion, it is considered that serverless platforms can satisfy the high performance and low latency requirements required in the inference of machine learning models.

The main limitations in the execution of machine learning models and artificial intelligence applications are aimed at the limited size of the calculation and memory and the execution times. Models that require GPUs are not supported by FaaS providers at this time. These limitations and possible solutions are studied in this investigation.

Serverless computing services enable developers and organizations to implement machine learning models in a more cost-effective way, especially in the inference phase where large amounts of resources are required in a short runtime. Instead of managing operational expenses and running the application 24 hours a day, it is possible to take advantage of fast scale-to-zero elasticity provided by serverless computing.

2.5 GPUs in the Cloud

In recent years, the use of acceleration devices has become popular to reduce the runtime of computer-intensive applications. The exploitation of Cloud Computing resources generally involves the use of VMs. Hardware such as CPUs and hard drives are well virtualized in this regard, but other more specific high performance devices such as network devices, FPGA or GPUs do not have the same support and performance. The use of GPGPU (General Purpose Computing on GPUs) is determined by the massively parallel computing power provided by GPUs.

The work of Yu and Rossbach [148] reports that GPU virtualization is a very active research area in which software solutions based on API remoting (or API forwarding), PCI-passthrough, and virtualizing the access to the GPU can be proposed. In addition, there is a review of GPUvm [133], a complete virtualisation design based on the XEN hypervisor to support VM access to NVIDIA discrete GPUs. With this solution, a node cannot use more GPUs than those hosted locally, and an idle GPU cannot be shared with other nodes. The work done by Tan et al. [136] proposes a multi-channel GPU virtualization architecture (VMCG) that provides a separate virtualized channel for each VM that competes with other VMs for the resources of the same physical GPU. The idea of virtualizing access to the GPU is to create virtual hardware that accesses the physical device as it does with other components such as the CPU or network devices. In this sense, providers have built support to virtualize GPU access, as is the particular case of NVIDIA, which has introduced vGPU [101] for some of its devices. Most public Cloud providers support this kind of virtualization.

PCI-passthrough is a technique that provides single VMs with exclusive access to a PCI device. Several common hypervisors support GPU passthrough and these have been used in production to obtain a high efficiency rate in virtual machines, close to native performance [140]. In this paper, the authors compare the performance of two generations of NVIDIA GPUs within the Xen, VMWare

ESXi²¹ and KVM²² hypervisors. Results show that GPU passthrough to KVM achieves better results than Xen and VMWare.

GPU performance is similar on native machines and virtual machines that use PCI-passthrough [147]. While it obtains better performance, such exclusive and native access has several security implications [88], and allocating one GPU for one VM results in an exclusive usage of that GPU, thus reducing the efficiency.

In the case of API forwarding, rCUDA [41], vCUDA [129], GViM [58] and gVirtuS [54] are existing solutions that achieve good performance under certain circumstances. These approaches provide access to the GPU devices from a virtualization platform that does not have direct access to them, that is, communication between the GPU and the VM is done over the network. From a performance standpoint, specific software is responsible for sharing device usage (GPUs) between different client virtual machines. An important element to keep in mind is that the network interconnection restricts the performance of the executions. Analysis in [107] reveals that improving network infrastructure can make a big difference to GPU virtualization.

In the work conducted by Reaño et al. [117], an evaluation of the different solutions available for remote virtualization of GPUs compatible with CUDA (Compute Unified Device Architecture) [137] is carried out. In the article, the authors describe the characteristics of some of the technologies, as well as some advantages and disadvantages. In addition, a comparative analysis is performed taking into account bandwidth and latency. The analysis showed that rCUDA presents the best overall results in terms of latency and bandwidth.

The work carried out by Iserte et al. [67] introduces a solution based on GPU virtualization and shareability through rCUDA, to strike a balance between supply and demand for application accelerators. In addition, the feasibility of

²¹VMWare ESXi - <https://www.vmware.com/es/products/esxi-and-esx.html>

²²KVM - https://www.linux-kvm.org/page/Main_Page

this approach is studied in the AWS public Cloud, and a module for OpenStack is developed to support virtualized devices and the logic to manage them.

A recent study by Reaño et al. [119] presents an efficient mechanism designed for memory copies between remote GPUs located on different nodes through the rCUDA framework. The results obtained with this mechanism refer to an improvement in performance with respect to the original performance achieved by CUDA when taking advantage of GPUs located in the same node of the cluster.

There is few literature referring to the use of acceleration devices on serverless platforms. One of the examples found is the work conducted by Kim et al. [77] featuring a serverless platform with GPU support, using NVIDIA-Docker to allow function containers to access GPUs. This architecture has the disadvantage that each GPU can only be accessed by invoking a function simultaneously.

Applications like deep learning, that focus on solving complex application-related algebraic operations, benefit from the use of specialized hardware such as GPUs. The use of accelerators has been limited in the Cloud despite offering improvements in both performance and energy efficiency compared to traditional CPUs. GPU virtualization helps provide the scalability, power efficiency, and high performance required for GPGPU applications in serverless computing.

2.6 Chapter Conclusions

This chapter has reviewed the most recent contributions on the subject of container computing, serverless computing, and the use of GPUs in Cloud applications. These topics are related to the central axis of the research and constitute the information base for the development of novel designs.

As a result of the research carried out, we can conclude that the development of Cloud Computing has fostered the creation of different types of services

through the Internet. Using these services can reduce IT costs by shifting away from investments in hardware, as services can be automatically scaled on demand. The development of virtualization technologies and containers, led to the emergence of a new paradigm in the Cloud, serverless computing.

In summary, serverless computing is an innovative approach that has been extended to all areas of research related to Cloud Computing. As a cloud-based model, it allows developers to run applications without direct contact with the underlying technology, thus making it possible to focus more on the functionalities of their application. Through its pay-as-you-go model, costs compared to other Cloud services are minimal for certain workloads.

Despite all the identified advantages of serverless computing, there is still a long way to go to fill gaps related to the integration of acceleration devices in the execution of applications, the development of open-source frameworks that avoid vendor lock-in by public Cloud providers and the integration of this technology with other areas of research such as machine learning and artificial intelligence.

This service model reduces the investment in infrastructure and costs are only incurred when the service is used, that is, when the function is executed. These characteristics make it an area in which efforts are invested and in which new open-source frameworks are developed that allow moving away from public serverless platforms into on-premises Cloud platforms.

Taking into account the advantages and limitations of the research analyzed in this chapter, the fundamental ideas for the development of the solutions presented below were compiled.

Chapter 3

Serverless architecture and CloudTrail-Tracker

In the development of different knowledge areas such as industry, research and education, computer tools are a key element. In this sense, two fundamental techniques such as simulation and modeling allow us to reduce waiting times and costs by providing effective solutions to scientific problems. Usually, these solutions are obtained through intensive applications, that require specialized computing infrastructures to obtain results in a reasonable time.

In these chapters we refer to different strategies of serverless computing in public and on-premises Clouds, starting from a general model, which allows providing new and effective solutions to challenging problems.

3.1 Function as a Service Model

In the world of data-intensive computing cost management and dynamic scaling are the main elements behind the success of any analysis platform. This is why many platforms and technologies have adopted serverless services where application execution is automatically managed on demand incurring in costs only for the time the service is used. The main advantage of serverless computing is that the developer does not have to think about where the code will run, and can concentrate only on its implementation.

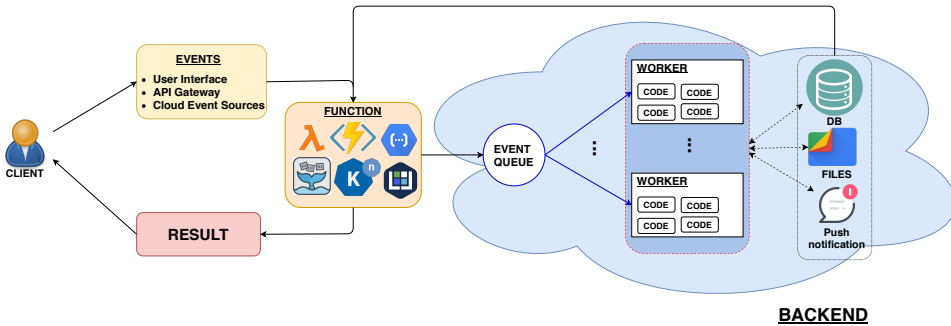


Figure 3.1: General structure of an event-driven Serverless Architecture.

As shown in Figure 3.1, the main responsibility of a FaaS platform is to provide event-based processing capabilities on a dynamically managed computing infrastructure. The design, first manages the actions performed by users that trigger events that invoke functions. These functions are generally stateless, that is, each function processes its input, performs the calculation, and writes its output back to shared storage, so that each function’s processing is independent of each other. The stateless functions execute specific code developed in one of the programming languages supported by the FaaS platform. The functions can be called synchronously or asynchronously. In the case of asynchronous functions, the client is not blocked waiting for the result, and on the contrary, synchronous functions do so regardless of whether they produce results or not. The functions can be invoked from triggers such as an event generated via HTTP or an event source. After determining to which

function or functions the event should be sent, an existing instance of the function is executed or used to process it. Once the event has been processed, the execution logs are collected, the result is made available to the user, and the function is stopped when the execution finishes. The main challenge of this architecture is to implement these functionalities taking into account cost, scalability, fault tolerance and latency.

The efficiency of a serverless platform lies in processing the input element in a function that starts quickly and efficiently. In addition, an event queue is used to trigger the executions. Depending on the status of the event queue, the execution of a function is scheduled and the resource allocation is managed. The code runs in ephemeral execution environments using functions that communicate with the back-end as a service to drive data storage, file management, or notification needs. Applications created with serverless infrastructures automatically scale as usage increases. Therefore, if a function needs to be run in multiple instances, the provider will start, run, and terminate them as needed.

Regarding the cost model, users pay for the time and resources used when executing the functions. The ability to scale-to-zero is one of the most important elements of serverless platforms. The prices of resources that are measured like memory or CPU depend on each provider. Scale-to-zero can bring some latency problems commonly called *cold start* [20]. In general, the life cycle of a function involves 4 processes: download the code of the application, start new execution environment, execute init code and execute handler code, which is the main function that is executed after the invocation.

For the first invocation of a function, the first thing is to download the application code and start a new execution environment based on the configuration information of the function (amount of memory and maximum execution time, for example in the case of AWS Lambda). When a function is executed for the first time, all these events occur and it is what is commonly called a full *cold start*. If another request is received, only the handler function is executed again and this execution is much faster, which is called

warm start. Scaling a function to zero means having no available execution environments, so when an invocation is received, the service identifies if a warm execution environment is available, when it does not find any available computing resources, the process of downloading the code begins, start a new runtime environment, execute the init code and invoke the handler function [104]. This bootstrapping time also occurs in the scale up by provisioning more instances to handle traffic. Although it is true that scaling to zero introduces latency in the execution of the function, it allows to reduce operational costs by not having active execution environments when the function is not used.

It is important to note that the *cold start* phenomenon is more common to be found in development environments than in production environments, since in these environments the functions are executed more frequently and the execution environments remain active for a time window before scaling to zero. Functions with higher memory allocation are generally less affected by *cold start* [104]. The package size of the function is another element to consider to reduce this phenomenon [104]. The size of the package impacts the time it takes for the server to download the application code, which is the first process discussed earlier in the function's life cycle, so having packages as small as possible can help in this regard. How often a function is called influences how long the function stays warm. The most frequently used functions target warm environments more often than when they are used less frequently [104]. To mitigate *cold start* issues, Cloud providers are beginning to implement countermeasures such as provisioned concurrency at the expense of incurring in additional cost [98].

The code implemented by the developer can have a high impact on the *cold start* of the function. In the init code, all the libraries and elements necessary to execute the handler function are specified, so it is possible to optimize this code, for example, by trim the libraries to import the necessary services or not load unnecessary files, to name a few.

A short run stateless function represents the level of abstraction provided by FaaS models. This model fits data processing applications that require the

processing of dynamic workloads and allow the platform to automatically scale in an application independent way [36].

Serverless platforms today share many points in common. They share similar pricing, implementation, and scheduling models. The difference between them lies in the Cloud ecosystem considering that only their own services can be used and the choice of the platform will probably force developers to use the native services of that platform [20]. When using the serverless services of a public Cloud provider in particular, the provider is the one who hosts, supplies and manages the computing resources to run the application. With this, an important concern that arises is the “vendor lock-in”, which refers to the impossibility or difficulty of migrating the application to another provider’s Cloud platform. The emergence of open-source serverless frameworks has changed this idea as these solutions can work well on multiple Cloud platforms.

The advantages of the serverless paradigm extend to both users and providers. From the user’s perspective, the developer no longer needs to provision or manage the computational infrastructure (servers, VMs, containers), their job is to focus on the logic of the application defining a set of functions that allows their application to offer the necessary services. On the other hand, the stateless programming model allows the provider to have more control over the applications that are running, which enables the optimization of the platform [20].

From this perspective, this thesis introduces three serverless computing strategies that are developed throughout this chapter and in chapters 4 and 5. First, a completely open-source serverless platform is presented that allows obtaining information about the use of an AWS shared account. The platform has been applied in the educational sector, to carry out an analysis of learning in subjects related to AWS. One of the consequences of designing and implementing serverless platforms at a particular provider is the “vendor lock-in”. Another difficulty when using public Cloud providers is that currently public serverless frameworks (e.g. AWS Lambda, Google Cloud Functions, IBM Cloud Functions, Azure Functions) do not offer access to the use of

acceleration devices like GPUs. To solve these challenges, the second strategy implemented in this thesis allows integrating the use of acceleration devices in a serverless on-premises platform, which allows functions to access GPUs by the functions, in addition to avoiding the lock-in of Cloud providers. Finally, in the AWS environment and in a private Cloud, another totally serverless platform is presented that facilitates the inference phase in machine learning models in a user-friendly way.

3.2 CloudTrail-Tracker: A Serverless Platform for Enhanced AWS Usage Insights

Accounts in AWS are associated with credit cards, so it is very common in research groups, companies and courses associated with AWS to use delegated accounts on the same account. All users in the organization use the same AWS base account, and the administrator sets limits on user access to services. For this AWS has the AWS IAM (Identity and Access Management)¹ service that allows us to manage access to AWS services and resources securely.

When an AWS account is shared by multiple users, it is important to be able to monitor the activity of each user to supervise the use of resources. In this sense, AWS offers the CloudTrail service that allows the related activity to be retained in the AWS infrastructure. From the CloudTrail console it is only possible to view the last 90 days of activity and events in text format. Therefore, through AWS it is not possible to observe the activity carried out over a longer period in a fast and efficient way, such as through graphical means.

The proposed platform, CloudTrail-Tracker, is fully serverless and runs entirely on AWS and provides information on the use of an AWS account shared by multiple AWS IAM users. The logs collected by the AWS CloudTrail service are stored in a DynamoDB table through a Lambda function, which allows obtaining information in periods of time longer than the 90 days that is presented in the CloudTrail console. Logs are queried with precise timestamps

¹AWS IAM - <https://aws.amazon.com/es/iam/>

through a REST API created with Amazon API Gateway. The front-end is a web portal based on Vue.js² to visually represent the information of different users. This platform has been applied in learning analytics for Cloud instruction.

This section explains in detail the architecture of CloudTrail-Tracker³ [95] developed for the processing of students' data that take different courses that use the AWS platform. The main contribution of CloudTrail-Tracker is materialized in a learning dashboard that combines information about the use of resources in an AWS account shared by several students.

Also, the learning dashboard allows students to know the degree of progress of laboratory activities that must be carried out by them. In addition to academic interest, the data collected by the platform can be especially useful for monitoring the resources used by users who share the same AWS account. This research was carried out jointly with José Ramón Prieto Fontcuberta, at that time a student of the Degree in Computer Engineering at the Universitat Politècnica de València. Partial results of this research were included in his Final Degree Project, available in [114]. The main contribution of the author of this thesis on this platform, was mainly focused on the development of a serverless front-end that would allow to easily visualize the information stored on the events generated by users on the AWS platform. The author also participated in the overall design of the architecture, fine-tuning the performance requirements, establishing a security configuration and, finally, performing a cost analysis of the implemented platform.

3.2.1 Components used

The development of the CloudTrail-Tracker architecture made use of a series of underlying technologies that will be described in this section. Firstly, the back-end of the application in charge of storing the actions carried out by users (students, teachers and administrators) is described. Secondly, the technologies

²Vue.js - <https://vuejs.org/>

³CloudTrail-Tracker - <https://www.grycap.upv.es/cloudtrail-tracker>

used in the educational panel that constitute the front-end of the application will be addressed.

Back-end services

The AWS services used to implement the architecture are described below:

- *AWS IAM*⁴ (*Identity and Access Management*): It is a service that allows you to securely manage the users, groups, permissions and roles that can access the services of an AWS account. IAM is an AWS account feature that is provided at no additional charge. With the use of AWS IAM it is possible to enforce access policies to AWS services, allowing the same account to be shared in a workgroup.
- *Amazon S3*⁵ (*Simple Storage Service*): It is an object storage service offered by AWS. The files are stored in *buckets*, accessible from any location on the Internet, which offers scalability, high durability, security and performance at low cost.
- *AWS Lambda*⁶: This service allows creating event-driven functions without the need for explicit server management. To use this service, we have to define the function code using one of the available programming languages (NodeJS, Ruby, .NET, Java, Go, Python and custom runtime).

Furthermore, it is possible to trigger a Lambda function indicating an input event, for example, the upload of a file to an S3 *bucket*. Lambda allows thousands of functions to run in parallel. The cost of this service depends on the allocated amount of RAM, from 128 to 3008 in 64 MB increments, and the execution time in blocks of 100 ms.

- *AWS CloudTrail*⁷: This service allows you to register, monitor continuously and retain activity on the infrastructure of an AWS account.

⁴AWS IAM - <https://aws.amazon.com/es/iam/>

⁵Amazon S3 - <https://aws.amazon.com/es/s3/>

⁶AWS Lambda - <https://aws.amazon.com/es/lambda/>

⁷AWS CloudTrail - <https://aws.amazon.com/es/cloudtrail/>

When using this service, a series of files are generated periodically in an Amazon S3 *bucket* that describe the events executed in the AWS account.

- *AWS DynamoDB*⁸: It is a NoSQL database service offered by Amazon, that stores key value pairs with very low data access latency. Provides optimal performance and storage scaling automatically. This service is useful as a fully managed database system in the Cloud. This service enables two capacity modes to be activated with specific billing options: on-demand capacity mode and provisioned capacity mode. The provisioned capacity mode was used in this platform since it allows specifying the number of read and write operations per second. The cost in this mode depends on the number of reading and writing units contracted.
- *API Gateway*⁹: It is a service that allows the creation, maintenance, monitoring of REST APIs on a large scale. Its use is usually combined with other AWS services such as AWS Lambda so that a request to the REST API created with the API Gateway triggers the execution of a Lambda function to process the invocation.
- *AWS Cognito*¹⁰: It is a service that allows incorporating functionalities of access control, registration and user login in web or mobile applications. It allows managing millions of users and the login can be done through a group of defined users, through social identity providers such as Facebook or Google, or through external providers of OpenID Connect¹¹.

⁸AWS DynamoDB - <https://aws.amazon.com/es/dynamodb/>

⁹API Gateway - <https://aws.amazon.com/es/api-gateway/>

¹⁰AWS Cognito - <https://aws.amazon.com/es/cognito/>

¹¹OpenID Connect - <https://openid.net/connect/>

Front-end components

Once the data generated with the activities carried out by the users in the AWS account has been collected, it is necessary to display this data in a simple and understandable way. To do this, a web interface was developed using Vue.js a JavaScript framework to create user interfaces with intuitive, modern and easy-to-use features. Vue.js has a very active user community due to all the advantages that it provides, such as the reuse of components in the code, it does not require the implementation of special models or collections. Above all, it allows the execution of a developer view where you can observe in real time the changes developed on the web. It is also possible to generate a static website in a comfortable way that can be served from any server.

An important element in the web interface is the use of the *AWS SDK for JavaScript*¹². In this sense, it is used to tackle authentication and authorization workflows of users through Amazon Cognito, thus obtaining credentials to access other AWS services such as API Gateway that allows consulting the data of each user.

3.2.2 Architecture

Figure 3.2 shows the architecture of the application. When a user employs a AWS service, an event is generated and stored in an S3 *bucket* using CloudTrail. These logged events describe who, what and when a particular service was used. CloudTrail typically adds multiple events in the same JSON file. Every time a file of this type is stored in an S3 *bucket*, a Lambda function (*Write* in Figure 3.2) is activated. This function parses this file, removing unnecessary information and storing the relevant fields in a DynamoDB table.

In order to query the stored information, a REST API is created through the API Gateway service. API Gateway integrates with AWS Lambda so that once a request has been received by the API, a Lambda function (*Read* in Figure

¹²AWS SDK for JavaScript - <https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/index.html>

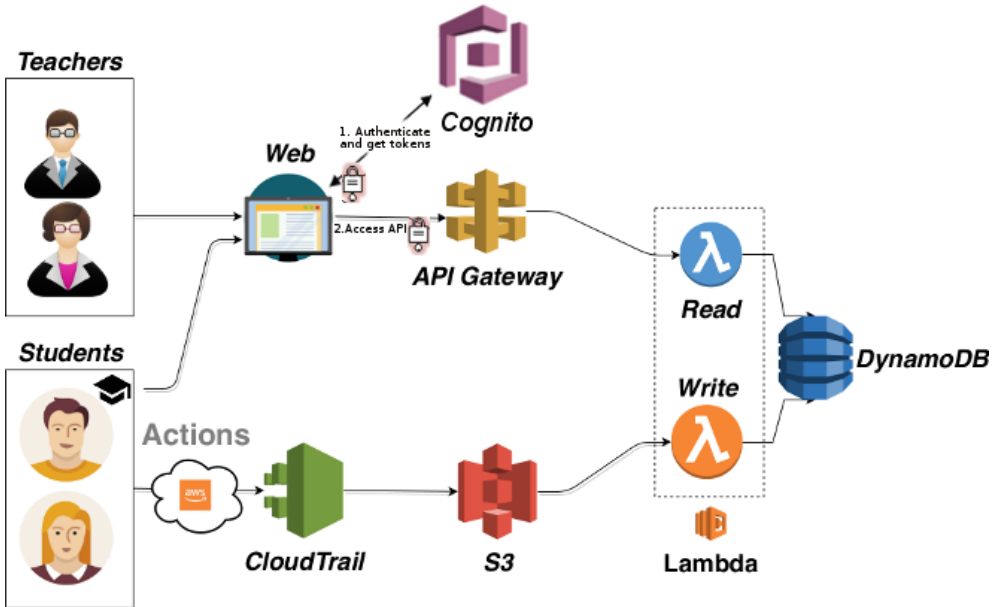


Figure 3.2: Architecture of CloudTrail-Tracker. Source:[95]

3.2) is triggered to query for the events in the DynamoDB table. This makes it possible to create a reactive Cloud service, which is automatically activated by user actions.

As explained above, in order to display this information in a user-friendly way, a web interface was developed that consults the API Gateway to produce information regarding the services used. This service has also been implemented in a serverless way and is made up of a static web page hosted in an S3 *bucket*, offering scalable and very low-cost access.

To obtain greater automation in the process of compiling and loading the web interface in the S3 *bucket*, Jenkins¹³ was used. Jenkins is an open-source server that enables automation of software development parts related to building, testing, and deploying. In our case, once developers make a Pull Request to the master branch of the project, two jobs are automatically created in Jenkins.

¹³Jenkins - <https://www.jenkins.io/>

The first one performs the process of compiling the code (command - `npm run build`) as a static web page and includes in the project a file with environment variables that are necessary for the configuration of AWS Cognito and that for security reasons is necessary to keep them in a safe environment. Jenkins' second job is performing the static web loading process in the S3 *bucket* that works as hosting. When the process is completed, the developer receives an e-mail indicating that the two jobs have been completed successfully.

In order to make the application accessible from any device (mobile, tablet, or laptop), a responsive theme that adapts to different screen sizes was used, offering an improved experience on multiple platforms. For user authentication on the web and API protection, Amazon Cognito is used. When a user authenticates to the application with their access credentials (username and password), Amazon Cognito verifies them and, if correct, an access token is obtained to communicate with the API Gateway.

The web interface allows obtaining high-level aggregated information for both students and teachers in a specified time, which is the basis of the learning dashboards. The designed architecture allows serverless management through a reactive event-based computing scheme in order to minimize costs and keep data access times low.

Computational Platform and Cost Analysis

The proposed solution is developed entirely in the AWS Cloud, it is robust and scalable and generates costs only for the used services specified in Section 3.2.1, without depending on having a configured server. The AWS account used in the development of the architecture is a shared account among the researchers of the GRyCAP research group.

To carry out the queries, it was decided to use the API Gateway since only the queries received by the API and the amount of outgoing data transferred are charged. It also provides a free layer of one million queries for up to 12 months. Subsequently the price for each million queries is 3.50 USD/month, in

case of making more queries the prices decrease. An API Gateway limitation is that it has a maximum of 30 seconds of waiting for a query, after this time the API returns a timeout error.

Due to the advantages that we have already discussed of the serverless architecture, we consider that the best solution for data processing was through AWS Lambda, where you only pay for the allocated resources and the function execution time. Depending on the query, the function only runs for a few seconds, unlike having a server that would be running all the time despite not making queries. The cost of Lambda functions is calculated using a table [15] depending on the duration with the allocated memory and the number of executions. It also includes one million free requests per month and 400000 GB/seconds of computing time per month that covers the needs of the proposed architecture. The function used to query the DynamoDB database has an allocated memory of 128MBs of RAM, and the write function has a memory of 128MBs of RAM. The memory that can be defined in AWS Lambda are in the range of 128 MB to 3,008 MB in 64 MB increments. The write function is triggered every time CloudTrail saves the generated events, in a few minutes, in a file with format *gz* in an S3 *bucket*. For the analysis of the information generated in the file and its writing in the database, it was considered that the minimum value of the memory range was sufficient. This allows saving in production costs and if it does not exceed the limits of the free Lambda layer, the cost will be zero. In the case of the reading function, a low memory intermediate range is decided since the amount of data to analyze depends on the query made to the DynamoDB table, so if the range of the query is one time greater, means that the amount of data to analyze is greater. Using Amazon CloudWatch Logs¹⁴, an AWS service to monitor and store log files, it has been verified that a query typically uses between 60 and 80 MB of memory, which has allowed us to adjust the function's resources to 128 MBs of memory, enough for the case study. If necessary, the characteristics of these functions can be improved by adding more processing memory.

¹⁴Amazon CloudWatch Logs - <https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/WhatIsCloudWatchLogs.html>

For the storage of the data generated by the events, we concluded that the best solution would be to use DynamoDB since there was a large amount of data and the response times are very low, requiring less than 29 seconds to comply with the response limitation of API Gateway. For the DynamoDB collection, the provisioned capacity mode was used, where the number of data write and read operations that the application will need is defined. The number of operations can be modified if it is considered that they can grow over time due to the use of the application. The service does not take into account the use of CPU and memory, as it does with other solutions. The free layer of this service includes, without expiration time, 25 read units, 25 write units for each index, and 25GB of indexed data, thus meeting the needs of the project.

When CloudTrail is configured to store the trace of events executed in the AWS account and deliver them to the S3 *bucket*, you pay only for the *bucket* storage and not for the CloudTrail service. In the case of Amazon S3, you pay for the storage used, these being relatively low. For example the first 50 TB/month at 0.023 USD/GB. The free layer has 5GB of storage, 20,000 GET requests, 2,000 PUT, COPY, POST, or LIST requests and 15GB of outbound data transfer per month for one year. These characteristics satisfy the needs of the project. In the case of Amazon Cognito, you only pay based on the number of MAU (Monthly Active Users). The first 50,000 MAUs are free, which largely meets the needs of the application.

From the analysis carried out previously, we can conclude that the cost of the proposed architecture is more than affordable. In other words, a zero-cost serverless architecture has been implemented that has benefits for all users who use it, whether they are account administrators, teachers or students who use AWS services. Therefore, the benefits of using a serverless architecture are demonstrated, in terms of provisioning the infrastructure transparently to the user and reducing costs.

3.2.3 Use cases: Processing student progress data with CloudTrail-Tracker

This section discusses the main use cases and results of processing student progress with CloudTrail-Tracker. The analysis is carried out in the context of different subjects in which AWS is used to exemplify public Cloud provider management concepts. The results are obtained through the data collected from the students taking these subjects. Firstly, we will analyze the general questions regarding the case studies, then the main results obtained are exposed and discussed.

Objectives of the experiments

CloudTrail-Tracker was developed during the academic year 2017/2018 and it was put into production since the academic year 2018/2019. CloudTrail-Tracker is used in three master's degree and one online course related to Cloud Computing topics that involve the use of AWS and that are taught in the *Master's in Big Data Analytics*, *Master's Degree in Parallel and Distributed Computing*, *Master's Degree in Information Management* and in the *Online Course in Cloud Computing with AWS*¹⁵, Figure 3.3.

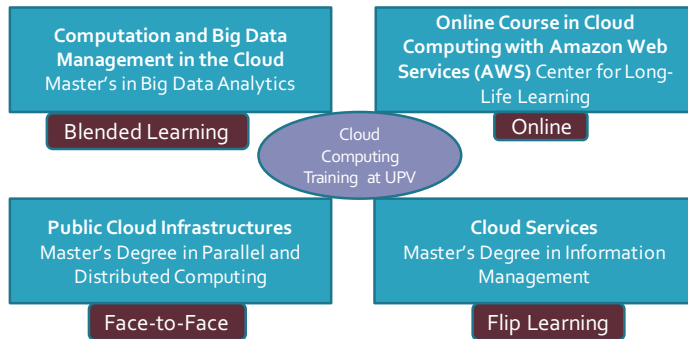


Figure 3.3: Cloud Computing training at the Universitat Politècnica de València, in Spain. Source:[95]

¹⁵CursoCloudAWS (in Spanish) - <http://www.grycap.upv.es/cursocloudaws>

In these subjects different learning schemes are used, ranging from the participative face-to-face lesson, to the flipped classroom in which either students visualize the theoretical material at home, and use the laboratory sessions to carry out the practices or viceversa. From the beginning of the assignments, the students have all the material (video lessons, exercises, learning guides), as well as a practice environment available, from any place, in the Cloud 24x7 so that the students can carry out the practices at any time. The practice environment is a virtual machine pre-configured with the students' accounts. The students connect via SSH to the virtual machine that, among other software, has installed `aws-cli`¹⁶ that allows managing AWS services from the command line.

CloudTrail-Tracker is particularly useful in online training because it allows monitoring the actions carried out by the students. The instructor, based on this information, makes decisions aimed at improving the course experience. For this reason, it has been integrated with the *Online Course in Cloud Computing with AWS*, an asynchronous online training course, offered publicly, that since July 2013 has trained more than 900 people from different countries (mainly Spain and Latin America). It is a totally online experience where students have laboratories to deploy their own virtual infrastructures. It also involves multiple learning materials and self-assessment tools, as described in the work by Moltó et al. [93].

The common nexus of all educational activities is the use of AWS as a public Cloud platform where practical activities are carried out. The teacher initially used the portfolio mechanism where the answers to questions related to practice were collected. However, this method was prone to copies among students. The fundamental purpose is to know if the student has successfully carried out the practices, so by using CloudTrail-Tracker the teacher pursues two fundamental objectives. Firstly, it prevents students from devoting time to create the portfolio by collecting evidences of the completion of the lab activities. Instead, this portfolio is now automatically generated by CloudTrail-Tracker and it is

¹⁶AWS-CLI - <https://aws.amazon.com/cli>

available for both, the student and the teacher. This way, the student no longer commits an additional effort preparing it, and the effort invested by the teacher in assessing it is greatly reduced, since evidences are collected by automated means. Secondly, it avoids the problem of copying among students because the system detects the degree of achievement of activities for each student assigned at the beginning of the course.

Experimental Results

In each of the subjects, the students carry out a set of guided practical laboratories that show the main functionalities of the AWS services involved. In order to extract the information generated by each student, we have worked mainly on the definition of the necessary indicators that make it possible to demonstrate the realization of practical activities based on the recorded information of the events. Therefore, a practical laboratory is defined as a set of ordered events, i.e. an action in an AWS service, which students must perform in order to classify a practice as complete.

CloudTrail-Tracker has three fundamental panels. The first panel, Figure 3.4, provides general information about all the resources used in the shared AWS account, in different time frames (last hour, last six hours, last day and last week). This dashboard provides service-specific information that may be more costly such as executed EC2 instances (RunInstances), database (CreateDBInstance), Lambda functions (CreateFunction) and load balancers (CreateLoadBalancer) creation. It also allows to know how many services each user has used in the selected time period, together with a table to know specifically which services the users have used, as shown in Figure 3.5. The utility of this panel is more for the AWS shared account administrator to know the allocated resources and who has used them.

Another display panel, *Search by user*, allows a more centralised search of the AWS services used by a specific user, Figure 3.6. Similarly, with the *Detail* button, it is possible to find out which specific event has been generated for each service. The administrator can control the resources used by each user

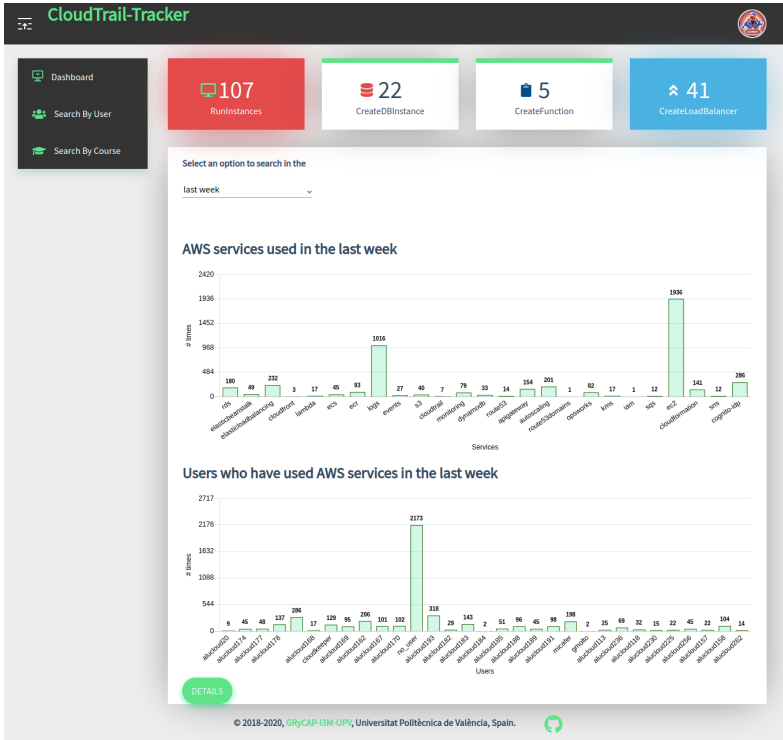


Figure 3.4: CloudTrail-Tracker start panel.

and if the teacher knows the number of resource creation operations in each practice guide, it is possible to know in general if a student has used said services.

The third of the panels is *Search By Course* and is arguably the most important one for students and teachers. The function of this panel is to detect the percentage of a student’s progress in the practical laboratories. This educational panel is responsible for calculating whether a set of events is included in the set of events defined for each practical activity, in a given period of time. When students consult this panel, their username is selected by default and they must select the subject to which they belong and the period of time in which they were taking the course. By default, the date selected is that of the current academic semester (2019/2020). Figure 3.7 shows the percentage

DETAILS

Show entries Search:

#	User	Event	Timestamp (UTC)
0	alucloud20	CreateSecurityGroup	19:10:03 09-04-2020
1	alucloud20	RunInstances	19:15:59 09-04-2020
2	alucloud20	StopInstances	19:28:00 09-04-2020
3	alucloud20	TerminateInstances	19:30:27 09-04-2020
4	alucloud20	RunInstances	19:33:44 09-04-2020
5	alucloud20	TerminateInstances	19:37:55 09-04-2020
6	alucloud20	CreateBucket	19:51:47 09-04-2020
7	alucloud20	PutBucketPolicy	19:52:41 09-04-2020
8	alucloud20	PutBucketWebsite	19:55:11 09-04-2020
9	alucloud174	CreateSecurityGroup	20:46:00 09-04-2020

Showing 1 to 10 of 4,678 entries Previous 2 3 4 5 ... 468 Next

Figure 3.5: Details of services used by all users in the CloudTrail-Tracker *Dashboard*.

of completion of laboratory practices for a specific student. The teacher at a glance can determine if the student is progressing in carrying out the activities since the activity compliance bars have a traffic light classification system, that is, the bar can have three colors: red (<40% completion of practice), yellow (between 40% and 80% of completion of practice) and green (>80% completion of practice).

According to Butler and Wine et al. [27], student learning and understanding improves when a link is established between feedback and self-regulated learning. In [111] self-regulated learning is defined as a fundamental component of the teaching process, which implies active self-control of behavior and motivation in the performance of the academic tasks of an individual student. Freitas et al. [50], on the other hand, describes the use of gamified dashboards, learning analysis, and performance tracking to provide immediate educational feedback in higher education studies. This corresponds to the objective of this educational panel, to increase the effective action of feedback and self-regulated learning by providing access to the tool, so that students know the practical laboratories already carried out and those that are still pending [95].

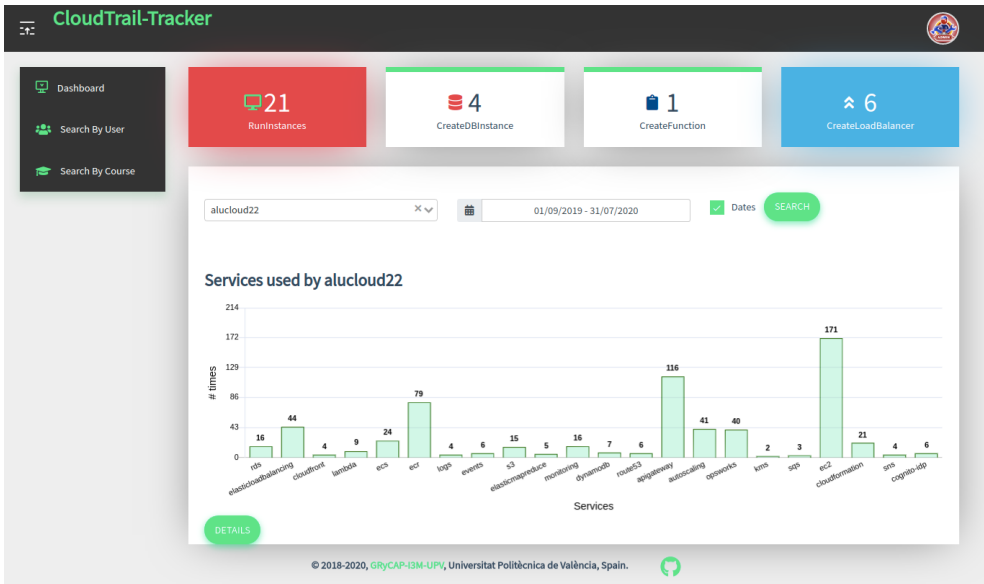


Figure 3.6: Services used by a student in a certain period of time (CloudTrail-Tracker Search by User panel).

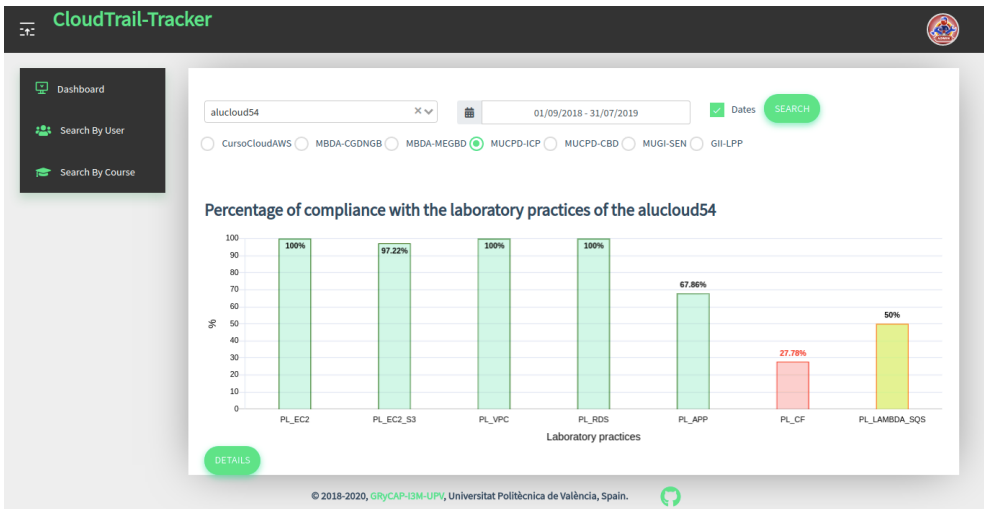
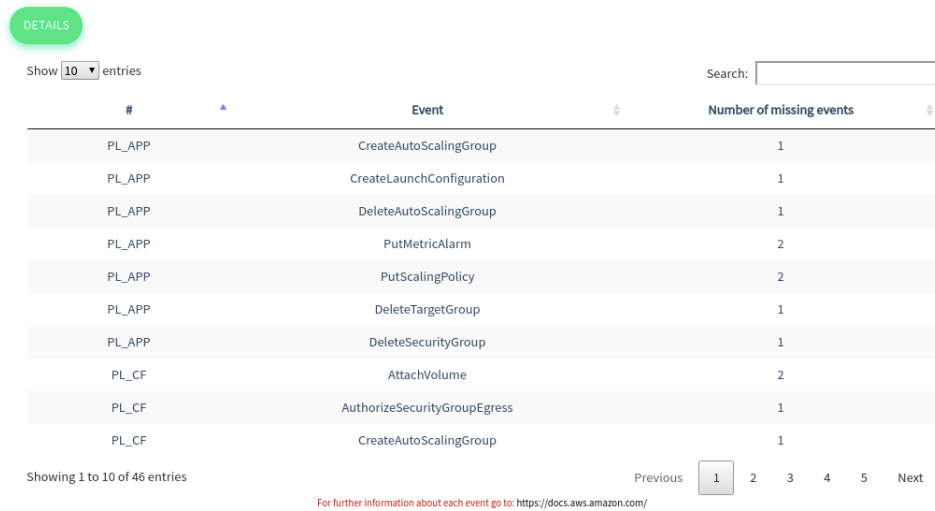


Figure 3.7: Percentage of completion for each hands-on lab for a specific student. Source:[95]

It is important to note that, in addition to knowing the percentage of compliance with practical activities, students can consult the actions that are still missing to complete the entire practice, as shown in Figure 3.8. By including the missing actions, a double objective is achieved. First, the students are aware of the missing activities and are given the opportunity to self-regulate. Second, many times the students forget to terminate the resources once the practical activity is finished, incurring in unnecessary expenses for the teacher.



DETAILS

Show entries

Search:

#	Event	Number of missing events
PL_APP	CreateAutoScalingGroup	1
PL_APP	CreateLaunchConfiguration	1
PL_APP	DeleteAutoScalingGroup	1
PL_APP	PutMetricAlarm	2
PL_APP	PutScalingPolicy	2
PL_APP	DeleteTargetGroup	1
PL_APP	DeleteSecurityGroup	1
PL_CF	AttachVolume	2
PL_CF	AuthorizeSecurityGroupEgress	1
PL_CF	CreateAutoScalingGroup	1

Showing 1 to 10 of 46 entries

Previous 2 3 4 5 Next

For further information about each event go to: <https://docs.aws.amazon.com/>

Figure 3.8: Missing actions for each hands-on lab for a specific student. **Source:**[95]

Taking into account the different panels developed in the web interface, it is possible to define three fundamental roles:

- **Students:** Users with this role can check the percentage of compliance with the practical activities carried out as well as be aware of the activities that still need to be completed. In this way, students benefit in terms of self-regulated learning.
- **Teacher:** Users with this role can monitor the progress of students in each course. Also, useful information is displayed that allows the teacher

to make decisions about how to conduct the learning of each student. The main advantages for users with this role lie in the supervision of the students, in addition to the automatic feedback for the student using metrics that influence the evaluation. This favors an early intervention by the teacher to reinforce the most lagging students and provide additional support if necessary.

- **System Administrator:** Users in this role can monitor the resources used in the AWS shared account in near real time (15 minute delay due to the CloudTrail internal log delivery process). From this point of view, irregularities in the use of resources can be detected, in addition to allowing a planning of the cost of the resources used throughout a month.

Another important aspect is to point out that this panel allows displaying information about the role of each user. Users with the student role can only access their own data. Users with the teacher role can access the information of all students. This prevents a student from feeling bad about seeing that his progress with respect to other students is inferior and, on the other hand, the teacher can differentiate who are the students who need more attention.

Table 3.1: Results of the satisfaction questionnaire with CloudTrail-Tracker (the percentage of students that answered in each interval, using a 10-item Likert scale, is shown).

Question	[0,4]		[5,7]		[8,10]	
	N	%	N	%	N	%
The tool was always accessible whenever I needed it	0	0,0	3	2,1	139	97,9
I knew how to use the tool without the instructor's guidance	1	0,7	7	4,9	134	94,4
I was able to properly understand the information given by the tool	2	1,4	12	8,5	128	90,1
The information shown by the tool helped me identify my progress in each lab activity	7	4,9	16	11,3	119	83,8
It can be considered an appropriate support tool for the technical education in AWS	4	2,8	12	8,5	126	88,7

Taking into account the impact on students of the developed tool, we decided to conduct an online survey where they could express their level of satisfaction

with CloudTrail-Tracker. For the questionnaire, the Likert scale of 10 items was used, where 0 means totally disagree and 10 means totally agree. Table 3.1 shows the results of a total of 142 students surveyed who have completed some of the subjects. The research carried out in [95] included the results of 46 students surveyed, which were those available at the time the article was presented. For this research, it seemed important to update the tables with the information collected so far. Table 3.2 shows the results of the satisfaction test taking into account each of the educational activities carried out.

The results indicate that students consider the use of the CloudTrail-Tracker tool beneficial as support material for the activities that take place in AWS. No differences were detected between the students who received the classes in person and those of the online course. The ability to provide almost real-time information on the progress of the practical activities carried out in the subjects was highlighted. For their part, the students indicate a series of recommendations that help improve the tool, for a better understanding of the data provided.

Table 3.2: Results of the satisfaction questionnaire with CloudTrail-Tracker in each of the educational activities (N stands for the number of students that filled in the questionnaire for each subject, AVG stands for average and STD stands for standard deviation. Subjects: CS (Cloud Services), PCI (Public Cloud Infrastructures), CBDMC (Computation and Big Data Management in the Cloud)).

	CursoCloudAWS (N=88)		SEN (N=17)		ICP (N=7)		CGDNBD (N=33)	
	AVG	STD	AVG	STD	AVG	STD	AVG	STD
Q1-Accessibility	9,62	0,85	10,00	0,00	9,71	0,76	9,85	0,44
Q2-Ease	9,42	1,12	9,65	0,86	9,71	0,49	9,03	1,19
Q3-Self explanatory	8,99	1,39	9,59	0,80	9,86	0,38	9,03	1,07
Q4-Relevant	8,20	2,27	8,20	0,87	9,57	1,13	9,18	1,18
Q5-Utility	8,85	1,85	9,82	0,39	10,00	0,00	9,36	1,03

The tools that provide information on the behavior of the students allow to reshape the courses taking into account the aspects in which the students present greater difficulties. The information collected with the use of CloudTrail-Tracker allowed a statistical analysis of the *Online Course in Cloud Computing with Amazon Web Services*. The study involves a total of

Table 3.3: Average percentage of progress of the students in each lab activity. Activities are shown in the table in chronological order of appearance in the course material from left to right.

Lab_Activity	Avg. Progress (%)
PL_EC2	72,12
PL_EC2_S3	54,58
PL_RDS	46,78
PL_APP	43,07
PL_CF	39,15
PL_VPC	35,54
PL_LAMBDA_SQS	19,74

427 students who participated in the academic year 2016/2017, 2017/2018, 2018/2019. All information related to the course is available at [93].

Table 3.3 displays the average percentage of student progress for each hands-on activity conducted on AWS. The practices are shown in chronological order, that is, PL_EC2 corresponds to the first laboratory to be carried out and PL_LAMBDA_SQS to the last one. The results indicate that, in general, the latest practices are the ones that students carry out the least frequently, this may mean that a re-planning of the practice time is necessary and that attention to students should be reinforced in the last part of the course.

3.2.4 Discussion

Cloud Computing helps solve some of the challenges facing education today. It can be said that Cloud Computing has driven the development of large online universities. The CloudTrail-Tracker tool constitutes a contribution in those courses where AWS services are used. Automated acquisition of student activities on the AWS platform results in a near-real-time learning analytics useful for students' self-awareness of their progress. Having a learning dashboard that gives the teacher an overview of the students' progress allows the development of different teaching methods.

Being able to share this information with students encourages self-regulation where the students know the activities they have carried out and those that still have to be completed. This is fundamentally the ability of a student to manage and plan their time, and to be aware of the academic skills that they may or may not achieve with the performance of the activities. In this way, the academic success of the students depends above all on themselves. The tool alone is an example of how to use and combine the diverse resources of Cloud providers such as AWS in applications that involve computing and data management.

In addition, the development of this type of applications, in educational areas, must be profitable and that it be carried out efficiently, compiling the information generated by each student in the development of laboratory activities. It is also important that it is accessible from anywhere at any time, which requires the system to be available at all times. The design of this event-driven serverless architecture encourages both, an all-day operating system that only generates cost when used. This depends on the use by students, but if they operate in the free tier of AWS services, the cost can be practically zero.

On the other hand, the use of the tool is not limited only to the educational sector, since anyone who has an AWS account can benefit. That is, through CloudTrail-Tracker you can view all the actions that are executed in AWS, which makes it possible to have more centralized control over the resources that have been used in a given period of time. The web portal developed in CloudTrail-Tracker allows to visually represent high-level aggregated information about the use of services on AWS by different users, based on the events recorded by CloudTrail. In addition, it allows querying personalized information for a particular user and obtaining summarized information across multiple users for the different services employed in a time frame. Despite the fact that in the use case it refers to courses, this concept can be replaced by a work group or project, since the grouping of users in different roles is done through AWS IAM. The *Dashboard* and the *Search by user* panel are general-purpose modules that provide an easy-to-use graphical view of the

services used in a multi-tenant AWS account. To the author's knowledge, there is no open-source serverless platform that offers high-level analytics of the usage of an AWS account. Therefore, outside of the educational field, the platform can be adopted by individuals, organizations or research groups that operate a multi-tenant AWS account.

After doing a study of the literature related to the implementation of educational dashboards, no research was found where reference was made to a serverless platform that automatically collects information about student actions in a shared AWS account. For other teachers or AWS account administrators to benefit from the implemented tool, CloudTrail-Tracker has been released as an open-source development under the Apache 2.0 license, available on GitHub¹⁷.

3.3 Chapter Conclusions

In the development of this chapter, the fundamental elements of the serverless architecture and CloudTrail-Tracker, the first strategy based on serverless computing addressed in this thesis, were introduced. All designed platforms build on the function-as-a-service model to provide event-based processing capabilities. In the FaaS model, the functions are executed in stateless compute containers, which are activated by an event, such as uploading a file to a file storage system such as Amazon S3, and which are completely managed by the provider of services.

The FaaS model allows the developer to focus more on the application logic and less on the configuration of the back-end infrastructure, which is managed by the service provider. The pay-as-you-go model, automatic scaling, and scale-to-zero are features that save costs and provide a flexible platform. This has led to this model being, in recent years, widely adopted by developers.

CloudTrail-Tracker is the first tool described, of the three strategies based on the FaaS model that are part of this research. CloudTrail-Tracker

¹⁷CloudTrail-Tracker GitHub repository - <https://github.com/grycap/cloudtrail-tracker>

processes usage records from the AWS public Cloud provider, with the aim of obtaining relevant information about the resources used in an AWS shared account. The system involves a completely serverless architecture based fundamentally on Lambda functions and a NoSQL database service, which allows minimizing the operational cost of the solution. It also operates within the AWS account itself, without requiring to provide external access to a third-party data processing tool. This tool has been implemented in the educational field in subjects related to computing in AWS. This has allowed the instructor to know the realization of the practical activities developed by the students in said infrastructure. In addition, an educational web portal has been implemented that includes graphic components to facilitate the interpretation of information. The use of the tool encourages the development of self-regulated learning. In this chapter, an analysis was made of the architecture on which CloudTrail-Tracker is based, as well as of the use cases where the tool was implemented. Finally, the results obtained were discussed and presented.

Chapter 4

Availability of GPUs on Serverless On-Premises Platforms

This chapter describes another serverless computing strategies developed as part of this research. As reflected in the state of the art (Chapter 2), the use of GPU-backed accelerated computing to optimize the processing times of the functions is a fundamental point that has not yet been properly addressed in serverless computing. We propose in this chapter a solution to this problem through a platform that supports virtualized computing of GPUs in on-premises serverless computing frameworks. Different GPU virtualization technologies are integrated with the benefits of the FaaS framework for scalable event-driven data processing.

This work extends a previous development from the GRyCAP research group: OSCAR¹ (Open Source Serverless Computing for Data-Processing Applications) [109], an open-source serverless platform that is based on Kubernetes and OpenFaaS for data processing applications. The platform supports parallel file processing through Docker container-based applications in response to uploading files to a data storage such like MinIO². The components used in the architecture design and some of the characteristics of the platform involved in the deployment of the platform are explained below.

4.1 Components used

Before addressing the general structure of the application, some of the underlying technologies used for the design and implementation of the proposed architecture are explained. Firstly, the structure of the OSCAR platform is exposed in a general way and then, the technologies used for the virtualization of the GPUs are described.

4.1.1 OSCAR

The serverless platforms of public Cloud providers have some limitations that prevent their benefits from being applied in the execution of general applications. These limitations are primarily focused on the maximum function execution time, the amount of allocable memory and the provided storage capacity. These challenges together with the advantages provided by the serverless paradigm have generated great interest in the development of open-source frameworks to define functions in orchestration platforms such as Kubernetes. These platforms do not have the aforementioned restrictions and can be deployed in public, private and federated Cloud infrastructures, as is the case of the EGI Federated Cloud³. Notice though that deploying an open-source FaaS framework on a public Cloud lacks the benefits of

¹OSCAR - <https://github.com/grycap/oscar>

²MinIO - <https://min.io/>

³EGI Federated Cloud - <https://www.egi.eu/federation/egi-federated-cloud/>

scale-to-zero. Thus, the main targets should be both on-premised and federated Clouds.

OSCAR provides users with the ability to self-deploy a scalable integrated platform accessed through a simple Graphical User Interface (GUI) to define and manage the full life cycle of functions that are triggered when users upload a file to a storage system data [109].

From the web interface the user can view the functions displayed on the platform and access the system that stores the files used with the functions. In the process of defining a function, the user has to specify three required elements and several optional elements (such as environment variables and CPU or memory limits). The required elements are: the name of the function, the Docker image that contains the code of the application, the environment with the software dependencies and a shell-script to be executed when the function is activated. The function is triggered when a file is uploaded to the input storage system. From this moment on, the file is processed in an ephemeral container that contains the application code and the settings specified in the function definition. After processing is complete, the output is displayed on the output storage system.

The components that make up the OSCAR architecture can be divided into two main groups: those that allow elasticity at the virtual machine level and those that provide functions as a service. For now we will focus on those that provide FaaS platform capability and later in Section 4.2.1 we will refer to the other components.

OpenFaaS, MinIO and Kaniko⁴ are the fundamental elements of the OSCAR architecture that provide the capability of functions as a service. OpenFaaS is a framework that allows the creation of serverless functions with Docker and Kubernetes. MinIO is an open-source object storage server with features similar to Amazon S3, was one of the first to adopt the API and provides S3

⁴Kaniko - <https://github.com/GoogleContainerTools/kaniko>

compatibility as the de facto standard for Cloud data storage [92]. Kaniko is a tool to create container images in Kubernetes from a Dockerfile⁵.

OSCAR Manager is the service that provides the orchestration of the services. Users can invoke and initialize functions through a REST API offered by this service [109]. The process of creating the function is transparent to the user and part of the request for a new function from the web interface. This request is received by the *OSCAR Manager* service and Kaniko creates the Docker image corresponding to the specified function. The image built by Kaniko is stored in a Docker Registry running inside the Kubernetes cluster, the input/output *buckets* are created in MinIO, and the OpenFaaS function is created using the image created by Kaniko. Once the process detailed above is complete, the function is ready for file processing. The function is invoked when a file is loaded into the MinIO input *bucket*, and then the result of the function is received into the output *bucket*.

The OSCAR user interface is developed with Vue.js and Vuetify⁶. Both are versatile JavaScript frameworks that allow for a better experience in web development. The web interface is deployed within the Kubernetes cluster so it is necessary to expose this service to make it accessible from a web browser. The web interface is implemented within the Kubernetes cluster, so it is necessary to expose this service to make it accessible from a web browser. To expose the services, a NodeJS server was used that allows interaction with other services such as *OSCAR Manager*, MinIO and OpenFaaS [109].

OSCAR is a platform developed by several researchers at GRyCAP, the space where the thesis was developed. In this sense, the collaboration with Alfonso Pérez and Sebastián Risco, researchers at the GRyCAP, stands out. The final results of this research are included in the Master's Thesis of Sebastián Risco [51]. The contribution to this platform was mainly oriented to the development of a web interface that allowed users to deploy and configure functions in a visual and simple way. In addition, to interact with the underlying services

⁵Dockerfile reference - <https://docs.docker.com/engine/reference/builder/>

⁶Vuetify - <https://vuetifyjs.com/>

of the platform such as OpenFaaS (creation of functions) and MinIO (file storage), a NodeJS server was implemented that, first, served the static web page and, second, allowed communication between the internal services of the Kubernetes cluster where OSCAR is implemented and the elements displayed in the interface. In addition, the author also introduced the GPU support for accelerating the execution of the function invocations in the OSCAR platform and supported the different use cases shown in the thesis.

4.1.2 *rCUDA*

Virtualization is the fundamental technology in public and private clouds capable of offering greater flexibility and simplicity. In virtualized and specifically containerized environments, access to the GPU is increasingly necessary, hence new challenges have emerged in terms of compatibility with these devices. In environments where high-performance computing workloads are required to be run, virtualization of GPUs can reduce acquisition costs and consumption by increasing the use of the same GPU.

In the section 2.5, the best-known solutions for GPU virtualization were mentioned. In the development of this research it was decided to use rCUDA⁷.

rCUDA is a framework developed by researchers from the Universitat Politècnica de València that allows the virtualization of remote GPUs compatible with CUDA. rCUDA is based on creating virtual GPUs on machines without a local GPU. These virtual devices are physically located on a remote host that offer GPGPU services [118]. In HPC centers, rCUDA increases the flexibility, sharing a GPU between multiple applications, thus promoting the development of a multitenant environment.

rCUDA is structured as a client-server application as shown in Figure 4.1. The rCUDA client is presented as an NVIDIA CUDA Runtime and Driver API and is installed on the node where the application that requests GPGPU services is located. On the other hand, the server must be installed on the node where

⁷rCUDA - <http://www.rcuda.net/>

the physical GPUs are located. When the rCUDA client receives a request from the application, it is processed and forwarded to the server, where it is interpreted and executed on the real GPU. Once the processing on the GPU is finished, the rCUDA server sends the result to the client, where they are finally sent to the application.

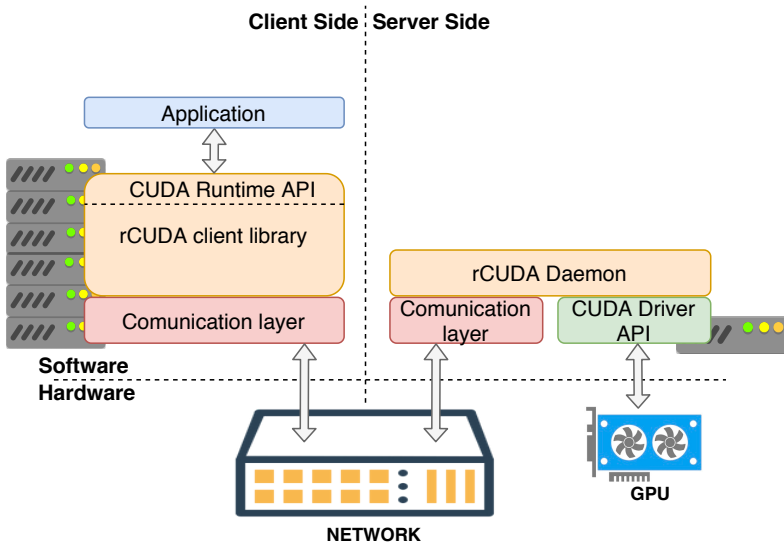


Figure 4.1: Overview of the rCUDA virtualization strategy.

As all rCUDA client-server applications require the network to establish communication between the client and the server, this may introduce delays in receiving information due to the characteristics of the network. To optimize this information exchange rCUDA provides support for various underlying network technologies such as InfiniBand [110], the RoCE network (RDMA over converged Ethernet) [120] for efficient data transport and the stack general of TCP/IP protocols.

Unlike other tools that allow you to virtualize GPUs with rCUDA, it is possible to share the same GPU by several applications. Furthermore, these virtualization framework separates the GPUs from the nodes, that is, it allows applications to access virtualized GPUs that are not even installed

on the physical machine where they are run. Another very important functionality of rCUDA is the possibility of providing support to different interconnection technologies such as the high performance network InfiniBand. These characteristics were taken into account when choosing rCUDA as GPGPU virtualization technology for integration with OSCAR.

4.1.3 NVIDIA Container runtime for Docker

The advantages of containers for application delivery by encapsulating their dependencies facilitate the reliable execution of applications, which has led to an increase of this technology in data centers. In data intensive applications as in the case of deep learning applications, it is nearly mandatory to use acceleration devices to reduce processing times. Docker containers do not natively support the use of specialized hardware such as GPUs, because both, Docker containers and hardware, are platform independent.

To allow the use of NVIDIA GPUs in Docker images, it is necessary to install NVIDIA-Docker⁸ on the host where the containers are running. NVIDIA-Docker is a runtime developed by NVIDIA that simplifies the use of applications that require GPUs within Docker containers. Once the runtime is installed, it is only necessary to specify the `--gpus` option when executing the `docker run` command, thus providing the container with all the elements necessary to execute the code on the GPU. One of the limitations of using NVIDIA-Docker is that it restricts the use of the GPU concurrently by another container.

4.2 Architecture

Figure 4.2 shows the proposed architecture for accessing a remote GPU from OSCAR through rCUDA. The rCUDA client is installed on the Kubernetes jobs where the functions are processed and the rCUDA server is located in another cluster where the physical GPUs are located.

⁸NVIDIA-Docker - <https://github.com/NVIDIA/nvidia-docker>

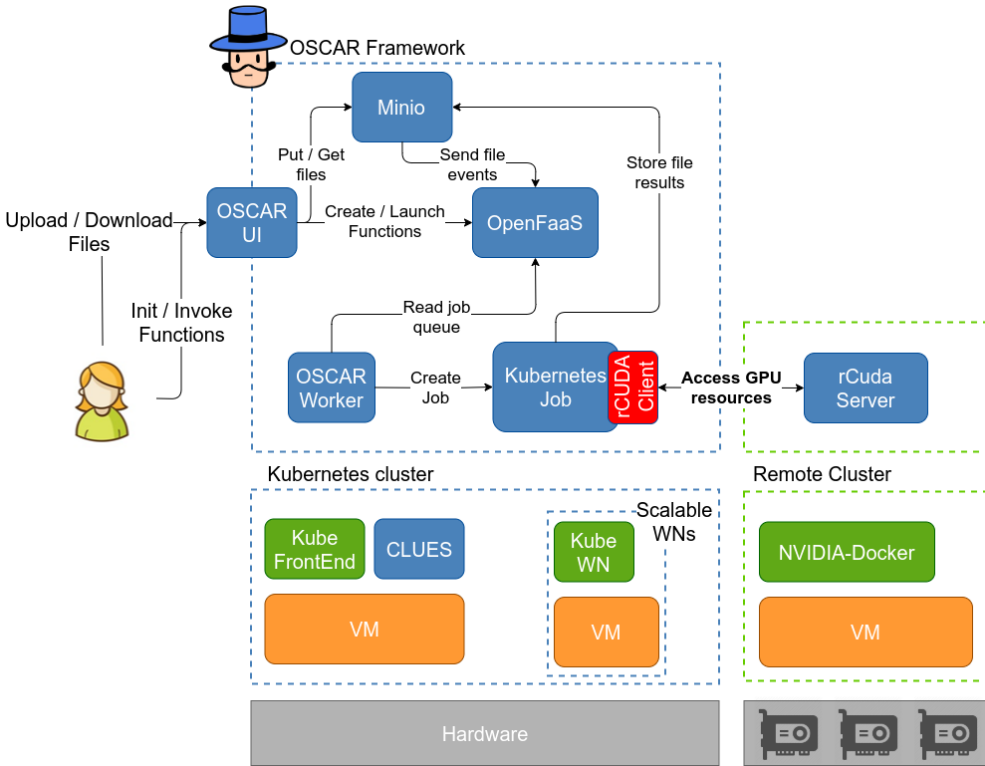


Figure 4.2: OSCAR architecture to access external GPU resources through rCUDA. Source:[96]

As explained previously, the proposed architecture is based on the OSCAR framework that allows developers to run applications packaged in Docker containers as functions that are triggered in response to certain events, in this case when uploading a file to the MinIO input *bucket*. This event-based architecture simplifies data and infrastructure management by abstracting all end-user configurations [96].

Figure 4.3 shows the integration of rCUDA into the OSCAR architecture with a higher level of depth.

On the one hand, there is a Kubernetes cluster with the OSCAR framework that supports the implementation and execution of functions that run as

Kubernetes jobs. The user from the web interface creates a function defining the Docker image with the application code, the name of the function and the script to execute inside the container. The *OSCAR Manager* uses Kaniko to build the Docker image including, in addition to the components for the serverless execution of the function (OpenFaaS Watchdog and FaaS supervisor), the rCUDA client libraries. This allows to have a ready image with the necessary environment settings for the execution of the function once an event is received. In this way, the image is reused at each invocation of the function, which makes the process of generating the final result faster. If the user application supports the use of GPU, requests to the CUDA API are made through the rCUDA client library to the server. On the other hand, in a Docker container running on a remote cluster where the physical GPUs are located, the CUDA libraries are installed and the rCUDA server running as a daemon. To access the GPUs from the rCUDA server, NVIDIA-Docker has been installed to virtualize the GPUs inside the container. It is possible to install the rCUDA server on the underlying operating system without using Docker, but by having it in a container eases the ability to add new GPGPU rCUDA node servers.

Requests made from the application to the CUDA API are detected by the rCUDA client library and over the network are sent to the rCUDA server and executed on the physical GPU. Once the result of the execution is obtained, it is sent again to the rCUDA client that sends the response to the application.

Including the rCUDA server in a Docker container is not trivial and was not supported by the rCUDA code. During the investigation, bugs were detected and reported to rCUDA developers who were responsible for correcting them in the code.

In the cluster where OSCAR is running, the user creates a new function with the application code from the web interface. It is important to note that in order to use the acceleration devices the user needs to configure their application with GPU support. Once *OSCAR Manager* receives the request for a new function, Kaniko is executed, where in addition to the user's application,

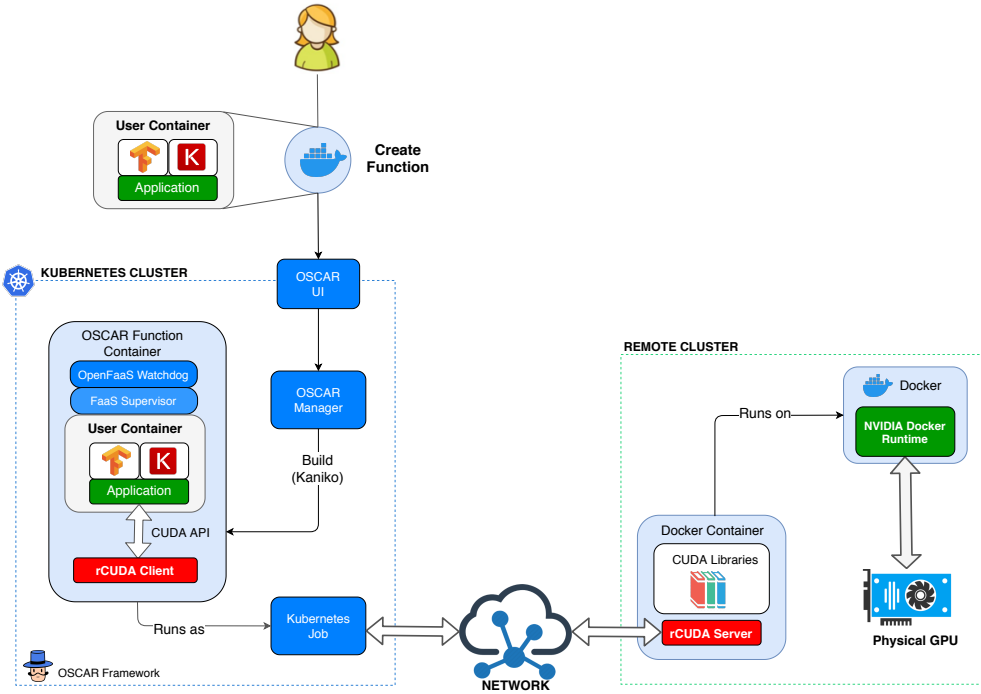


Figure 4.3: Integration of rCUDA into the OSCAR architecture. Source:[96]

the rCUDA client libraries and the environment variables required for the configuration, such as the IP of the server and the number of GPUs to be used, are included in the construction of the image. This is executed as Kubernetes jobs and the communication with the server is through the network.

All the components of the architecture exist independently, so the main contribution to the proposed solution is the integration of GPUs as background resources to serverless functions that need to be executed quickly and with resources immediately provisioned. This serverless execution strategy allows optimizing the use of GPUs, as well as reducing the cost of implementation, since rCUDA allows the use of a GPU by multiple applications. This architecture extends the support of the OSCAR serverless platform into mixed workloads that include standard CPU and GPU requirements [96].

The integration of an application in a Docker container is relatively easy, despite this, the integration of rCUDA had configuration problems because it had never been tested in these environments. These bugs were resolved in collaboration with the rCUDA developers. The work of the thesis has concentrated on the integration of these components into the OSCAR architecture to enable the efficient use of accelerator devices such as GPUs by the functions. This is not trivial and required to address issues such as: integrating rCUDA in the container environment, optimizing the execution of functions taking into account the best possible scenario, and accessing GPUs from a container in a Kubernetes cluster.

4.2.1 Description of underlying technologies

OSCAR services are implemented in a horizontally elastic Kubernetes cluster. The OSCAR architecture components that enable this feature are the *Infrastructure Manager* (IM) [29] and *CLuster Elasticity System* (CLUES) [3]. IM is an open-source tool for deploying complex Cloud infrastructures using high-level declarative languages like RADL⁹ (Resource Application Description Language) and TOSCA¹⁰. Thanks to the multi-cloud ability of IM, OSCAR can be deployed on multiple public, on-premises and federated Clouds.

CLUES is an open-source modular elasticity system that allows the introduction of horizontal elasticity capabilities (greater number of compute nodes) for cluster-based computing. Elastic Cloud Computing Cluster (EC3) [28] [31] was used for the deployment of these components. EC3 is an open-source tool that allows the deployment through IM of clusters defining elasticity rules with CLUES [109]. All these tools were developed by GRyCAP researchers and are currently used in production in EOSC marketplace¹¹ and EGI Federated Cloud¹², an IaaS-type Cloud, composed of private academic

⁹RADL - <https://imdocs.readthedocs.io/en/latest/radl.html>

¹⁰Tosca - <https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/os/TOSCA-Simple-Profile-YAML-v1.3-os.html>

¹¹EOSC marketplace - <https://marketplace.eosc-portal.eu/>

¹²EGI Federated Cloud - <https://www.egi.eu/federation/egi-federated-cloud/>

Clouds and open standards that offers computing resources to the European research community [35].

Firstly, as a proof of concept with OSCAR, the *Ramses* and *OneCloud* platforms were used. Later, to implement the proposed architecture with the tools mentioned above, the on-premises Cloud *Horsemen* was used. All this platforms are part of the infrastructure that the research group GRyCAP has and the characteristics were described in Section 1.3.

4.3 Use Case: Availability of GPUs on Serverless On-Premises Platforms

This section covers the use case implemented in the platform designed for GPU integration in OSCAR. The experiments carried out allow demonstrating the functionalities and showing the times obtained when using different scenarios involving executions in CPUs and virtual and native GPUs. Initially, it is explained what it consists of and the objectives of the case study. Then, the results of the experiments performed will be shown and finally a discussion is made taking into account the results obtained.

4.3.1 Objectives of the experiments

To cover the different GPU virtualization strategies, several scenarios have been designed to evaluate the benefits of the proposed solution in a real use case for the classification of echocardiography movies in pathological, borderline and sound cases. The classification model applied to this case study is developed by QUIBIM¹³ a company dedicated to applying image processing techniques to improve human health and that maintains a close collaboration with the GRyCAP research group. The model allows to classify an echocardiography image into three categories taking into account the acquisition view: four-chamber, long-axis and short-axis [71] [33]. To apply other techniques to extract information from the images, it is necessary to

¹³QUIBIM - <http://quibim.com/>

apply this classification since they depend fundamentally on the acquisition view.

The echocardiography movies used to test the functionalities of the architecture were obtained from the PROVAR [97] study. From each patient, 10-20 movies are obtained using different techniques, morphological and doppler, from three different viewing angles. The movies have a duration of 2 to 4 seconds and are made up of a few dozen frames. The resolution of the movies is 240 by 340 pixels.

The model for the classification of echocardiographic movies was developed using Keras¹⁴ and Tensorflow¹⁵. In the training, validation and estimation phases it is possible to use GPU if they are available. As indicated earlier, all echocardiographic movies are made up of several black frames, some of them contain noise, which makes it difficult to infer the view, so it is necessary to apply the model to each frame and thus obtain a consensus. Figure 4.4 shows the processing of an echocardiography movie. Firstly, it is necessary to extract the frames from the movie. Then, to each one of these frames the estimator is applied to obtain the classification of the view. Finally, the result obtained from each frame is analyzed and a consensus of the view is obtained.

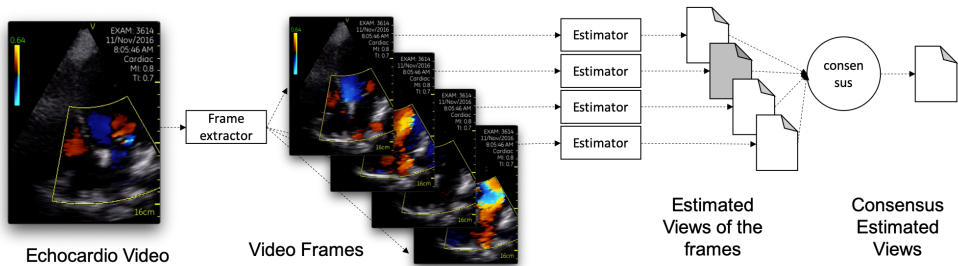


Figure 4.4: Processing pipeline for the use case. **Source:**[96]

In this particular use case, the model has been acquired in collaboration with QUIBIM, so the training was carried out previously. Therefore, we have focused on the segmentation part of the echocardiographic movies and on the

¹⁴Keras - <https://keras.io/>

¹⁵Tensorflow - <https://www.tensorflow.org/>

classification of the images since the executions are of short duration. The fundamental objective with this case study is to test the capabilities of the architecture regarding the use of virtualized GPUs. However, it is also possible to train the models since it supports the execution of long-running jobs.

Requirements

In general, the training process of ML models is the ones that need the most computing resources. However, the cost of the inference process is not negligible either. Serverless architectures appear to be a viable solution to utilize ML models in production by implementing parallelization techniques and acceleration devices. Several videos are obtained from the same patient, which are uploaded on the platform, this triggers the process, first, of segmentation of the video into frames and then of inference according to the different views. This allows the simultaneous processing of multiple videos.

In case studies that use unsupervised classification models, where the model conforms to observations, certain requirements need to be considered. These models are built on a previous phase where they are trained and validated, followed by an error evaluation phase and finally the production phase in which the models are used for classifying new objects. The following requirements are defined in this use case:

- The process of extracting the frames of each video and the simultaneous execution of the inference phase throughout the processing nodes must be able to be carried out without the application developer explicitly implementing it, allowing the parallelism capabilities to be exploited.
- GPUs should be used transparently by functions if available on the underlying computing node.
- Avoid blocking a GPU by a single process in order to maximize the use of resources.

- Easily deploy in the production environment, the applications already validated in the development environment.

Data is updated every day, so model retraining is critical, as continuous acquisition of information will optimize model accuracy, improve performance, and save time by customizing models automatic. In this sense, the use of GPUs can alleviate the problem to speed up continuous training and the deployment of algorithms. Using the function as a service model and GPUs seem to be very convenient for scenarios that use unsupervised classifiers.

Scenarios

Serverless computing has enabled a new way of designing architectures, where services are developed as functions largely based on CPU. In order to analyze the feasibility of integrating acceleration devices, such as GPUs, into a serverless on-premises platform (OSCAR), the following scenarios are proposed:

- **Scenario 1:** Execution from the Python console. Echocardiography movies are processed from the Python console in a Docker container that has access to a Tesla V100 GPU through NVIDIA-Docker. In this scenario, the execution times that are obtained are the most similar to those that would be obtained when accessing a GPU natively. The main disadvantage of this approach is that it is not possible to automate the movie processing workflow. The user would have to manually run the script to segment the videos and to classify each of the frames extracted from the movies. From the point of view of the development of computing today, this implies a manual execution that makes it difficult to use it in production, permanently blocks GPU resources and limits its efficient use since in case the process workflow had some stages that could be executed in CPUs, since they are not decoupled, they would continue consuming GPU resources. Furthermore, it requires manual implementation of parallelism. However, allows to determine the baseline processing times when using a lightweight virtualized approach to access a GPU.

- **Scenario 2:** OSCAR+CPU. In this scenario, the processing of the movies in the functions that run in OSCAR is done automatically, but based exclusively on the available CPU. These settings will serve as a basis for comparing execution times in scenarios where acceleration devices are used.
- **Scenario 3:** OSCAR+Remote GPU (rCUDA). This approach provides remote GPU access to serverless functions that run on the OSCAR platform. These functions are executed in Docker containers in an elastic Kubernetes cluster. Using rCUDA makes it possible to share a GPU across multiple applications at the same time, enabling the development of a multitenant environment and the efficient use of GPUs.
- **Scenario 4:** OSCAR+Native GPU. In this configuration the physical GPUs are located on the machine where the OSCAR platform is deployed. Accessing GPUs from within the Kubernetes cluster is via the NVIDIA device plugin for Kubernetes¹⁶. In addition, to access GPUs in the processing of the movies, it is necessary to install NVIDIA-Docker in the containers within the Kubernetes cluster, which are those that run as serverless functions.

The computer infrastructure used to carry out the test was *Horsemen*, whose characteristics were explained in Section 1.3. It is important to note that in all scenarios that use the GPU, virtualization techniques are used, from light virtualizations as is the case of scenario 1, to more complex visualizations as is the case of scenario 4 where it is necessary to virtualize devices at more than one level. The possibility of native execution on GPUs has been ruled out, since the OSCAR platform is a multitenant Cloud and therefore users are not expected to execute their code natively. The implemented scenarios allow to take advantage of GPU virtualization efficiently and profitably.

One of the main advantages provided by FaaS environments is the automated horizontal elasticity managed by the platform. This translates into the ability

¹⁶NVIDIA device plugin for Kubernetes - <https://github.com/NVIDIA/k8s-device-plugin>

to process multiple functions in parallel, which can be implemented in OSCAR. In this sense, two variants are defined in scenarios 2, 3 and 4 in order to achieve optimal parallelization levels. The implementation of the model to classify the images extracted from the echocardiography movies requires the use of the Tensorflow and Keras libraries, which is an important element when designing the workflow for the execution of the functions.

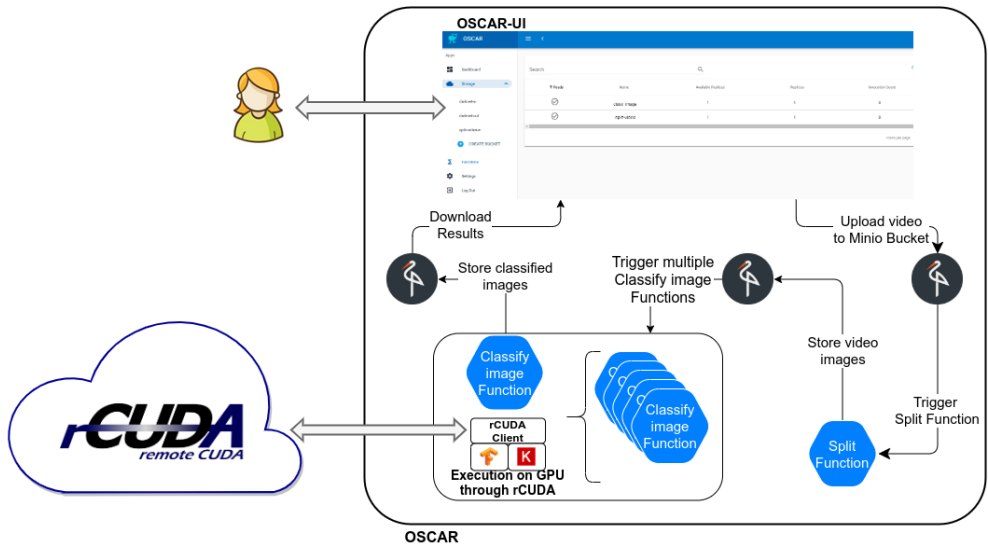


Figure 4.5: Workflow of the selected use case on the OSCAR platform using remote GPUs with two different functions. **Source:**[96]

Figure 4.5 shows the first variant, where two functions are defined: one to segment the echocardiography movies (*Split Function*) and the other (*Classify Image Function*) to classify the frames that are obtained from the segmentation. In this approach, the Keras and Tensorflow libraries are imported each time the segmentation function generates an image to classify. This solution achieves the execution of multiple classification functions in parallel but increases the execution time by having to import the libraries each time an image is classified.

In the second variant, Figure 4.6, all the processing is done in a single function. Firstly, the videos are segmented into images that are later classified. In this

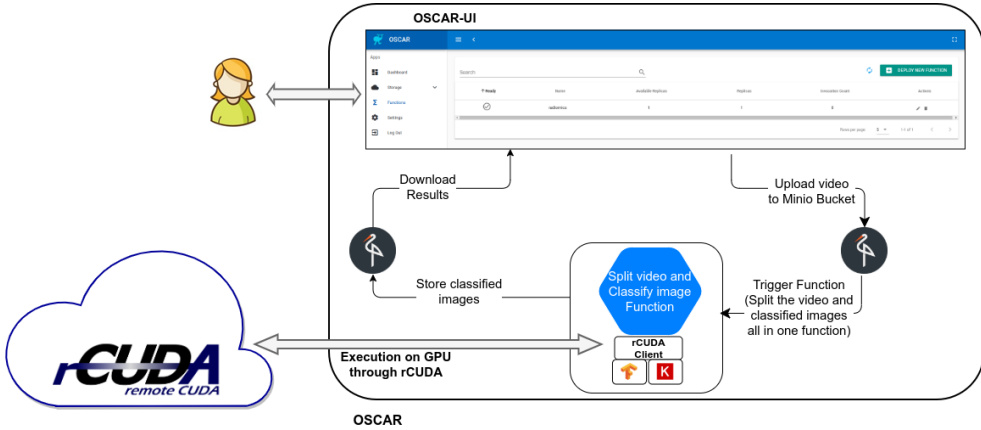


Figure 4.6: Workflow of the selected use case on the OSCAR platform using remote GPUs with all the code in one function. **Source:**[96]

case the Tensorflow and Keras libraries are imported only once. The use of this variant means that the parallelization will be done in the execution of multiple videos and not in the classification of the images as proposed in the first variant.

In Scenario 3, rCUDA is used as the virtualization technology for GPUs. In this case the use of the first variant can have a significant impact on performance since the libraries are imported from the client to the server through the network. Therefore, based on how it can affect the performance of the application according to the workflow described in Figure 4.5, it was decided to use the variant shown in Figure 4.6 to integrate rCUDA in the OSCAR platform. However, it is important to note that in other case studies, where the cost of processing the functions is of a higher order and it is not necessary to import the rCUDA libraries a large number of times, it could be relevant to divide the functions.

The objective of defining these scenarios is to be able to determine the advantages and limitations of each one, comparing the execution time of the echocardiography movies. With the result of this comparison, it will be possible to determine which of the scenarios is the most suitable for real use cases similar

to the one we have analyzed in this section, which constitutes a novel element in the integration of GPUs in serverless architectures.

4.3.2 Experiment Results

In each of the proposed scenarios, the processing time of the echocardiographic movies of a patient was analyzed. For this, the segmentation time of the video into images and its classification using the pre-trained model of Tensorflow and Keras were taken into account. Figure 4.7 shows the average times measured to load the libraries and the training model.

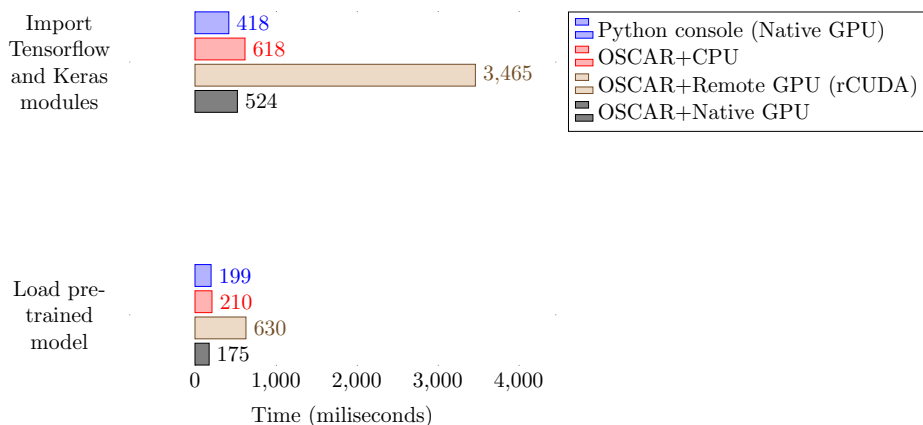


Figure 4.7: Average time importing the required modules and loading the pre-trained model in the analyzed scenarios. **Source:** [96]

As can be seen in the graph, the shortest time is for scenario 1 where execution from the Python console is using a lightweight virtualization environment. On the other hand, the most considerable time when importing the libraries and loading the model corresponds to scenario 3, where rCUDA is used to virtualize the GPUs within the containers that execute the processing. In fact, the rCUDA server loads the data and the application kernel (Tensorflow and Keras libraries) obtained through the rCUDA client onto the GPU (Tesla V100 32GB), using the network. The rCUDA client runs on Kubernetes jobs

on a virtual machine where access to GPUs is not available. The network connection established between the client and the rCUDA server is through a 10GbE card. It was precisely to avoid this overload that the second variant of packaging all the processing in a single function was selected.

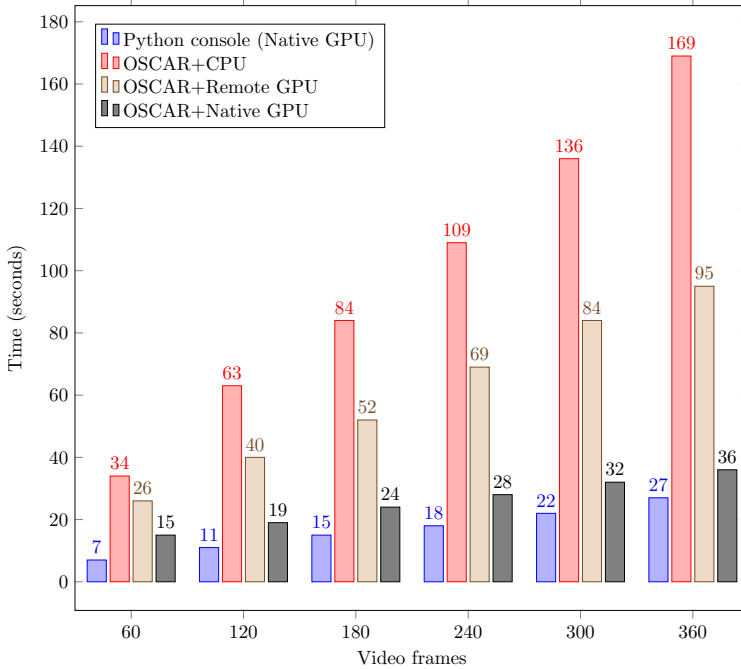


Figure 4.8: Comparison classifying different length videos in the analyzed scenarios. **Source:**[96]

Several executions were implemented using the on-premises serverless platform, to analyze the total processing time in each of the proposed scenarios. In the executions, videos with different lengths were used, which could be decomposed into 60, 120, 180, 240, 300 and 360 frames. Figure 4.8 shows the times measured for the different runs. It is important to note that the processing times in this graph include the time to import the Tensorflow and Keras libraries and the pre-trained model shown in Figure 4.7.

For each of the tests carried out, the results obtained in scenario 1 constitute the base reference for the other scenarios. In this case the GPU virtualization environment is the lightest and a serverless platform is not used. With this approach it is not possible to take advantage of serverless architectures when it comes to multi-run processing along with automated elasticity based on the number of virtual machines in the Kubernetes cluster. Using the serverless platform and processing on the CPUs, the processing times are higher than using the GPU, mainly in those videos that are divided into more frames and need more computing resources.

Scenarios 1 and 4 are the ones that use light virtualization environments for the GPU, since the GPUs are located on the physical machine where the execution takes place. If we compare these two scenarios, the execution time from the Python console are slightly less than the execution time obtained with OSCAR. This behavior is due to the fact that once a video is uploaded to OSCAR, the environment creation is triggered to execute the functions that implement the classification code, and in the case of the Python console, this environment is already created and the classification code execution is done by the user manually. An alternative to improve this difference would be to prevent the OSCAR platform from scaling to zero, which translates to keeping a function warm while waiting to execute the processing. Since the difference in the execution time in this case is not significant, we do not consider it necessary to implement this alternative.

In scenarios 3 and 4 the use of the GPU in OSCAR has been integrated. In this case, the best performance approach to process a single video, that can have different durations, is scenario 4, where the physical GPU is on the machine where the platform has been deployed. In the processing of a video on the serverless platform we define different phases: *Initialize container*, *Input/Output*, *Import Modules*, *Load pre-trained model* and *Processing*. The first two phases are executed by the OSCAR framework for creating the processing environment and to upload and download the input and output files respectively. The other phases correspond to the GPU virtualization overhead and depend essentially on the performance of the network. In the Figure 4.9

the processing times are shown, indicating each one of the phases mentioned above.

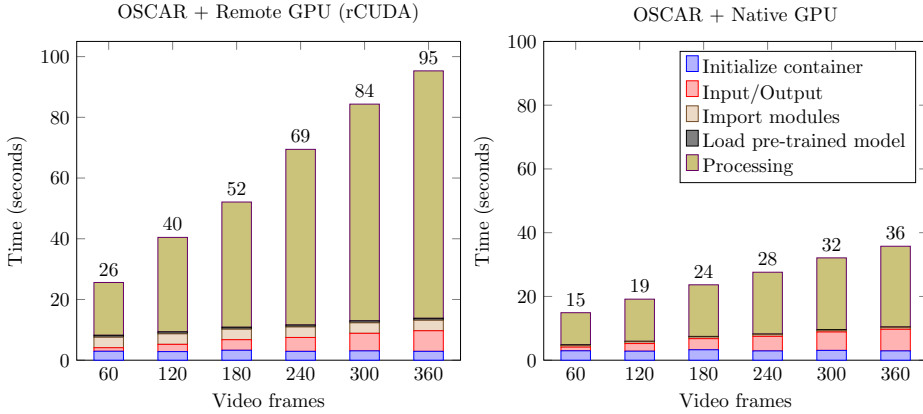


Figure 4.9: Time segmentation according to the main execution phases in scenarios 3 and 4. **Source:** [96]

It is evident that in the case of scenario 4 where access to the GPU is local, the processing is faster since the access is practically native. In the case of OSCAR with rCUDA, processing is significantly slower because rCUDA is a client-server application that makes extensive use of the network.

A last experiment has been carried out to evaluate the behavior of scenarios 3 and 4 when several videos are processed simultaneously. Figure 4.10 shows the execution times of up to 4 videos processed in parallel. In the case of OSCAR + Native GPU, as the processing load increases, a bottleneck is created due to the fact that the NVIDIA device plugin for Kubernetes does not allow sharing a GPU between multiple jobs, so the processing must be done one by one. In the case of OSCAR with the virtualized GPU through rCUDA, better results are obtained in the parallelization since rCUDA allows a better use of the GPUs by being able to share them with multiple applications.

In order to assess the elasticity of the architecture, the deployment of a new Kubernetes cluster node was activated to determine the time required for

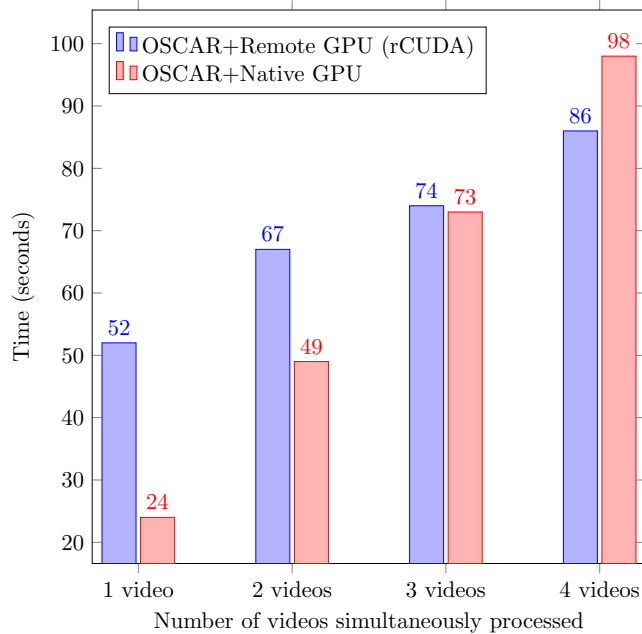


Figure 4.10: Execution times for processing up to 4 videos (180 frames) in scenario 3 and 4. **Source:**[96]

provisioning and configuration in the on-premises platform. The calculated time to provision one node was approximately 5 minutes, which can be significantly reduced by using preconfigured virtual machine images. In addition CLUES could be extended with proactive policies that anticipate peak loads. In the case of on-premises Clouds, there are other alternatives such as the suspension and reactivation of resources that significantly reduce the cost. In [24] all these alternatives are studied. This time was not considered when measuring the execution time of the implemented use cases, since the objective was not to measure the elasticity of the platform but the ability to integrate GPU support for the execution of serverless functions.

4.3.3 Discussion

In today's computing, graphics cards are a basic part of scientific applications, with particular importance in providing essential GPGPU services. The performance of an application increases considerably when acceleration devices are used. GPUs allow to solve problems of a higher order and complexity in a reasonable time that cannot be solved in CPUs. The code runs faster and allows multiple instructions to run simultaneously. The use of CPUs in this case study showed that the use of GPUs improves performance considerably in the intensive processing of complex applications.

Integrating GPUs into a serverless on-premises platform that provides event-based computing for invoking functions that run as jobs in an elastic Kubernetes cluster enables efficient processing of applications that make intensive use of computing resources. In this case study, different virtualization techniques of GPUs have been used for the segmentation of echocardiography movies and classification of the images obtained from the segmentation process.

Scenario 1 is a reference to analyze the processing times when using a GPU in a light virtualized environment and there is no automation in the execution of the video processing. In scenario 2, using OSCAR, automates the processing of videos in serverless functions and allows analysis of processing times when GPUs are not used. Scenarios 3 and 4 that use different GPU virtualization technologies in OSCAR have limitations and advantages, and it is the user who must decide which option best suits the needs of their application.

According to the results obtained in the previous experiments, where a single video that can have different duration is processed, the most effective solution is that of scenario 4, where the GPU is located on the physical machine where the serverless platform is implemented. Now, the results are very different when multiple videos are run simultaneously. In this sense, the use of rCUDA in OSCAR allows the development of a multitenant environment by executing multiple functions that simultaneously access the same GPU.

GPU support in container orchestration systems is managed by reserving full GPUs to containers by means of PCI-passthrough or similar technologies. The resource contention and management is simple and managed by the orchestrator. There are still unexplored areas regarding the integration of GPU-based computing into orchestration systems through remote virtualization techniques for accelerated computing. Concurrent GPU access cannot be implemented so far in a Kubernetes cluster with native access to GPUs where a function (application) reserve these types of devices completely and are only released when processing is complete.

We pursue the use of GPU virtualization or GPU sharing techniques, such as the use of rCUDA, which increases the complexity as the GPU is shared among containers. Time sharing is provided by the rCUDA server, but we could still limit the number of virtual GPUs through the resource annotation in the container orchestration. The resource contention is much more complex, and it is mainly performed at the rCUDA server side. However, we have the risk of over-committment in the client side (e.g. scheduling so many virtual GPUs that physical resources are exhausted). Although this may happen with rCUDA, this issue will be solved in future releases of rCUDA. In the meantime, our solution will take care of the limitations of resources by limiting the number of virtual GPUs.

In the Chapter 2 an analysis was made of some research that allowed the virtualization of GPUs. Such is the case of [77] where the authors propose to integrate an open-source serverless framework (IronFunctions¹⁷) with NVIDIA-Docker to allow containers to access GPUs. This solution has the disadvantage that only one function can access the GPU either virtualized or native, while the architecture designed in this research using rCUDA allows simultaneous access of several functions to the same GPU.

One of the most promising areas of research today is closely related to serverless computing and fog/edge computing. This paradigm aims to solve the problems related to the emergence of data-intensive and real-time applications powered

¹⁷IronFunctions - <https://open.iron.io/>

by mobile computing and the IoT. A promising application of serverless computing based on event-driven execution is workload distribution among hundreds of end devices for IoT applications.

The integration of acceleration devices in serverless platforms, allows the introduction of support for event-driven execution applications with dynamic cluster provisioning and scale-to-zero capabilities. OSCAR is the perfect paradigm for event-driven applications in highly distributed heterogeneous environments, such as the IoT. This type of platform fits very well in any application that conforms to a model where data is captured in a distributed way and in which its process requires non-trivial resources. For example, image analysis, characterization of medical data, dynamic re-training of machine learning models, to name a few.

Using rCUDA, as a GPU virtualization tool, allows GPUs to be multiplexed and made available where no physical GPUs are available. The integration of rCUDA in OSCAR, fits very well in an environment combined with a backend in the cloud where GPU resources cannot be exclusively dedicated or due to their nature it is more efficient to concentrate them on a specific equipment, such as having a server with NVLink¹⁸ and several cards, to which several conventional servers are connected.

The application fields of serverless computing are very broad and the power of using accelerated devices in the processing of scientific applications further widens the path, towards the application of other areas of computing, where the use of accelerated serverless computing to scientific applications can be very beneficial.

Developing this architecture extends the current trend of serverless architectures that rely only on CPUs. With this study it has been shown that the integration of GPUs in a serverless on-premises platform is possible and also allows improving performance in scientific applications that require the use of specialized devices.

¹⁸NVLink - <https://www.nvidia.com/en-us/data-center/nvlink/>

4.4 Chapter Conclusions

In the development of this chapter, the second serverless computing strategy that is part of the research was addressed. In this case we adopted OSCAR, an open-source platform for data processing applications based on the function-as-a-service model. The analysis mainly focused on the integration of virtualization techniques for GPUs in on-premises serverless computing platforms based on containers within OSCAR. Specifically, remote GPU virtualization techniques through rCUDA and light virtualization techniques through NVIDIA-Docker were addressed. These virtualization techniques were analyzed through 4 scenarios that include the “practically native” execution (light virtualization with NVIDIA-Docker) of the application on the GPU, the execution using the OSCAR platform but using the CPU, running with OSCAR using a GPU that was in a remote cluster and running with OSCAR in a cluster that had access to the physical GPU.

In addition, a use case of transthoracic echocardiography imaging that uses machine learning techniques to perform segmentation and classification of images was presented. For this case study, the 4 proposed scenarios were analyzed and the results obtained in each case were compared with the objective of analyzing with which approach the best results were obtained. The results obtained showed the feasibility of integrating GPUs and serverless computing through rCUDA, to provide accelerated support for the functions.

Serverless services for machine learning model inference

Using ML models and artificial intelligence applications on local servers is challenging due to the lack of high-end local computing power, what introduces significant delays in inference and training processes. Indeed, the continuous maintenance of hardware by system administrators is becoming increasingly complex. The difficulty in the use of these models is another challenge that machine learning users face. In addition, the problem of scaling according to demand is not easy to implement without downtime. Under these circumstances, serverless computing arises as an alternative that allows managing scalable computing resources in a transparent way for the user.

To this aim, the third serverless computing strategy presented in this thesis is to design a serverless service that enables the user to efficiently execute (in terms of cost and performance) predictions based on ML models on the AWS Cloud provider and on a serverless on-premises platform based on the

OSCAR framework. This architecture can automatically scale-to-zero to reduce deployment costs when the platform is not in use. An existing catalog of already trained and publicly available ML models is integrated with the serverless platform. The models used are from the DEEP Open Catalog¹ that was created in the DEEP Hybrid-DataCloud² [86] European project.

5.1 Components used

This section introduces the main technologies used before describing the overall architecture of the application. Firstly, the DEEP as a Service API³ (DEEPaaS API) [85] is described. DEEPaaS is a tool developed in the frame of the DEEP Hybrid-DataCloud⁴ H2020 project that facilitates access to machine learning, deep learning and artificial intelligence models through a REST API. Secondly, the frameworks, SCAR and OSCAR, that allow the deployment of the serverless platform on AWS and on-premises are described. These frameworks are products developed in the research group (GRyCAP) in which the thesis is developed and DEEPaaS is a development within the European project in which the group has participated intensively. Lastly, the web interface developed to facilitate access by users to the models implemented in the proposed solution is described.

5.1.1 DEEPaaS API

DEEPaaS API is a Python-based tool built on *aiohttp*⁵ that provides access to machine learning models. With DEEPaaS, users can integrate their machine learning model or a neural network with a REST API and thus access its functionalities through HTTP calls. For the integration of applications with the DEEPaaS API the requirements and changes are minimal, allowing easier interaction with the training and model validation functionalities.

¹DEEP Open Catalog - <https://marketplace.deep-hybrid-datacloud.eu/>

²DEEP Hybrid-DataCloud - <https://deep-hybrid-datacloud.eu/>

³DEEPaaS API - <https://github.com/indigo-dc/DEEPaaS>

⁴DEEP Hybrid-DataCloud - <https://deep-hybrid-datacloud.eu/>

⁵AIOHTTP - <https://docs.aiohttp.org/en/stable/>

DEEPaaS API allowed the prediction and training of a machine learning model through the integrated REST API. In order to obtain the prediction of a model in the proposed solution, it was necessary to develop in the DEEPaaS API a functionality to obtain the result of the prediction through a command-line interface. For this, a command written in Python was implemented, that allows specifying certain input values, to obtain the prediction result through the command line. This functionality provides support for batch-based execution of ML models packaged with the DEEPaaS API to execute on both high-end HPC supercomputers and batch-based computing installations such as virtual clusters.

The command to be executed is *deepaas-predict*. It requires options to specify the input file and the output directory, which are `--input-file` and `--output-file` respectively. The option `--url` allows you to specify a URL instead of a local file as an input element. The option `--content-type` allows you to specify the type of file you want to obtain at the output. By default it is a JSON file with the result of the prediction, but there are several models that allow, for example, to obtain JPG or ZIP files. Another of the available options (`--model-name`) is mainly used when there is an environment with several models installed and we must specify from which one we want to obtain the prediction. This implemented functionality allows to extend the field of use of the DEEPaaS API in scenarios where the REST API cannot be used and the command line is possible. The development of CLI tools that connect to a REST API that implements the service is the habitual approach in distributed systems.

5.1.2 *Serverless Frameworks*

This section refers to the frameworks that allow the serverless execution of applications on AWS and on-premises Cloud. For the deployment of the models in a private Cloud, OSCAR is used, an open-source platform that supports the FaaS computing model, which was described in Chapter 4 in Section 4.1.1.

Next subsection describes SCAR [108], a tool to run containers out of Docker images in AWS Lambda.

SCAR

As described in previous sections of this thesis, Lambda is the serverless computing service provided by AWS. One of the main limitations of this service and most all FaaS platforms is that it is not possible to install external packages at run time because functions do not run under root privileges. This prevents using generic applications or applications that require runtimes that are not supported by the serverless computing framework. Despite Docker could be a solution to encapsulate such dependencies, installing Docker requires root access. To resolve this limitation, a mechanism is required to run Docker images in user space without the need for prior installation. This is where tools like *udocker*, *Singularity*, *CharlieCloud*, *Shifter* or *Podman* are important because they precisely allow the execution of Docker images in the user space.

Despite these tools have similar characteristics, it is important to analyze some metrics like interaction with Docker images as the most widely used container technology, GPU and MPI support, security and portability, privilege model, among others, to determine the advantages and limitations of each one [94]. Table 5.1 shows a summary of the comparison of these tools. As a result of this analysis, it was concluded that:

- All container-based visualization tools have native support for MPI and Infiniband. They also have GPU through commands in the CLI.
- *Singularity* is the most used technology today in HPC centers, but it requires that system administrators have installed previously the Singularity framework in the resources.
- *CharlieCloud* and *Shifter*, also need privileged access to install some of their dependencies.

Table 5.1: Comparison of container-based virtualization tools

	Interaction with Docker	Deployment
udocker	Does not use Docker at all. It has the ability to import Docker images.	Just download and execute udocker and the installation will be performed automatically. (no root access required)
Singularity	Works completely independent of Docker. It has the ability to import Docker images, convert them to singularity images, or run Docker containers directly.	Root access required to install singularity, setuid need root to work correctly. Needs Go installed in the host.
Charlie Cloud	Needs Docker installed to run most of the commands. It can import Docker images.	Download the GitHub repo and build. Most commands require docker installed, so it requires root access.
Shifter	Docker does not need to be installed. Primary workflow is to pull and convert Docker images into Shifter images.	Needs an Image Gateway (depends on MongoDB server, squashfs-tools, virtualenv and Python 2.7). Requires root access to build and compile.
Podman	Does not need Docker installed. Has the ability to import images from Docker hub and Quay.io.	Available through installation packages on most Linux distributions. Requires root access for installation.

- *Podman*, of the technologies analyzed, is the most recent. For those users with experience in Docker, the use of *Podman* is reduced to replacing *docker* for *podman*. It has GPU support, however the configurations and settings are more complex than in the case of the previous tools. One of the characteristics of Podman that has most interested the development community is the ability to transition from the traditional world of containers to Kubernetes. The developments implemented look promising, but it will probably take a few more years for most HPC use cases.
- *udocker* has very good characteristics in terms of the metrics analyzed: it does not require prior installation or privileges and is a more user-oriented tool due to its simple way of use, which mimics the Docker CLI to some extent.

The previously discussed solutions that require privileges for installation are not operational as the executions are implemented in Lambda functions where root privileges are lacking. Indeed, SCAR is a framework that uses *udocker* to transparently allow containers out of Docker images to be run on AWS Lambda as event-driven applications such as in response to uploading a file in an S3 *bucket*. This is possible because SCAR uses *udocker* to extract container images from Docker Hub⁶ and run such containers on non-privileged user space [55].

The SCAR architecture is divided into two fundamental parts: *client* and *supervisor*. The SCAR client is a Python script that provides a CLI (Command Line Interface) and is responsible, among other tasks, for managing the life cycle of the Lambda function. On the other hand, the SCAR Supervisor represents the code of the Lambda function and is responsible for downloading the image from Docker Hub and creating the container using *udocker*. In case of having a triggering event source, it manages the input data and output and allows the execution of a script inside the container for greater versatility [108].

To create a Lambda function with SCAR a YAML⁷ file is needed to describe the job to be executed. It is necessary to specify the Docker image with the application is specified, the script that you want to execute within the container, the input and output *bucket* and the execution mode. SCAR has three execution modes: *lambda* (default), *batch* and *lambda-batch* that determines the service that will execute based on computational requirements.

In the *lambda* execution mode, all executions will be performed as Lambda functions. In the *batch* execution mode, the execution is carried out in AWS Batch⁸ a service that runs jobs based on Docker containers on a set of self-provisioned virtual machines, even with GPU support, that grow and shrink depending on the execution needs. In *lambda-batch* execution mode, executions are performed in Lambda and if the predetermined timeout is reached the execution is delegated to AWS Batch. In this way, AWS Lambda

⁶Docker Hub- <https://hub.docker.com/>

⁷YAML - <https://yaml.org/>

⁸AWS Batch - <https://aws.amazon.com/batch>

can be used for short jobs, while longer runs can be delegated to AWS Batch [14].

The AWS Lambda development environment has some restrictions such as: restricted computing capacity currently limited by 3008 MB of RAM, where CPU performance is correlated to the amount of memory chosen, maximum execution time of 15 minutes, read-only file system based on Amazon Linux, with a disk storage capacity of 512MB in `/tmp` that may be shared among several invocations, and a default limit of 1000 concurrent execution of the same function. These limitations mean that certain applications cannot run on Lambda, so SCAR has expanded its computer back-end by integrating AWS Batch into its execution modes. In this way, SCAR automatically delegates the job resulting from the invocation to the function as an AWS Batch job, without the user having to intervene.

AWS Batch

Typically, jobs that involve analyzing large amounts of data are long-running and require continuous use of available resources. AWS Batch directly provides a service and an endpoint for batch job execution, without having to provision and configure the resources required to install a job queue, and without worrying about elasticity management. In this sense, Cloud providers have developed services for batch job processing, such as AWS Batch.

AWS Batch dynamically executes Docker container-based jobs on a set of virtual machines that automatically scales up and down based on execution needs, allowing you to focus on analyzing results. AWS Batch accelerates the execution of jobs by provisioning GPU-based virtual machines in applications that have these devices integrated. The four components that help organize work in AWS Batch are [142]:

- *Jobs*: It is the unit of work that is sent to AWS Batch, which can be implemented as a shell script, executable or a Docker image. Jobs are run transparently as containerized applications in an EC2 instance of a

compute environment. The parameters of the job definition are used for execution.

- *Job Definition*: Describes how a job will run. Resources such as IAM roles are provided to access other AWS services, memory and CPU requirements are specified. The job definition can also control container properties like environment variables and persistent storage.
- *Job Queues*: Where the job remains until it is processed in a compute environment. It is possible to associate one or more environments to the same queue and assign priority values even between work queues.
- *Compute Environment*: Managed computing resources (AWS efficiently manages instance scaling and configuration) or unmanaged (user manages their own environment) that are used to run jobs. It is possible to specify the type of instance and the maximum, minimum and desired number of virtual CPUs.

A typical architecture for processing a job in AWS Batch is shown in the Figure 5.1. Through the AWS Batch console it is possible to send jobs to process. However, in the designed architecture, the Lambda function created with SCAR, which is activated when uploading a file to the S3 *bucket*, is responsible for sending the jobs to AWS Batch. The use of AWS Batch can be summarized in the following four steps:

- **Step 1**: Create a compute environment defining among other characteristics the type of environment (managed or unmanaged), name of the environment, role of the service and role of the instance to delimit the services associated with the jobs that will be executed in the EC2 instances.
- **Step 2**: Create a job queue to store the jobs to process. The *AWS Batch Scheduler* is in charge of deciding when, where, and which job will be processed in the compute environments. The job definition specifies the priorities for processing the jobs.

- **Step 3:** A job definition is created that specifies the Docker image to use, the memory and virtual CPU requirements that must be less than the maximum values of the compute environment stated before. You can also specify a maximum run time for the job execution to detect failures, as well as the number of times you want AWS Batch to retry job execution in case of failure.
- **Step 4:** Create a job from the job definition and submit it for its processing.

Jobs that are submitted to AWS Batch go through several states to reflect the different stages of the processing:

- *SUBMITTED*: A job has been submitted to the job queue but has not been processed by the scheduler. It is evaluated to see if any dependencies are necessary and, if so, it reaches the *PENDING* state, otherwise it goes to the *RUNNABLE* state.
- *PENDING*: A job that is queued but could not be processed yet due to a dependency on another job or resource.
- *RUNNABLE*: A job that has no dependency and is ready to be processed on a host as soon as resources are available.
- *STARTING*: The job has been assigned to a host and the container creation process is starting. After the container is running it goes to the *RUNNING* state.
- *RUNNING*: The job is running on an Amazon EC2⁹ instance as a Docker container. If the job has any failed attempts and you still have retries in your configuration the job goes to *RUNNABLE* state.
- *SUCCEEDED*: The job completed successfully and remains visible in the AWS Batch console for 24 hours.

⁹Amazon ECS - <https://aws.amazon.com/ecs/>

- **FAILED:** The job has failed on all the attempts. Jobs persist on the AWS Batch console for 24 hours.

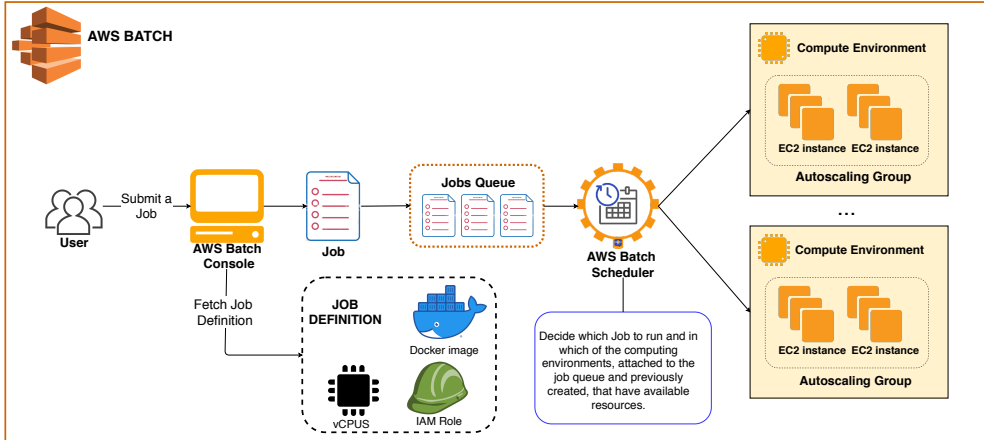


Figure 5.1: Typical AWS Batch architecture for job processing.

5.1.3 Web Interface

The implementation of a graphical interface contributes to improve the user experience, reducing the learning curve of the tool and its efficient usage. As part of this solution, a web interface was implemented so that users could interact more easily with machine learning models. For development, Vue.js and Vuetify were used, two JavaScript frameworks that, as explained in previous Sections (3.2.1, 4.1.1), allow the development of user interfaces in an intuitive and simple way. The web interface is compiled as a static web that is served from an S3 *bucket* and registered under the *grycap.net*¹⁰ domain.

Web authentication is done through Amazon Cognito, a service offered by AWS that allows user sign-up, sign-in, and access control in web and mobile apps quickly and easily. In this architecture, authentication is used in two ways, the first through an Amazon Cognito *User Pools*¹¹ where a group of users is

¹⁰Web Interface - <https://scar-deepaas-ui.grycap.net/>

¹¹User Pools - <https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-identity-pools.html>

created and assigned access credentials. The second form of authentication is through an OpenID Connect provider. In this case, it was integrated with DEEP IAM¹² to allow existing users from that community to log-in to our service. Both forms of authentication are integrated with Amazon Cognito *Identity Pools*¹³ (Federated Identities) to obtain temporary AWS credentials with limited privileges that allow access to other AWS services such as AWS Lambda, Amazon S3, AWS CloudWatch among others.

The user can obtain easily the prediction of a model from the web interface, for this it is possible to upload, download, delete and list the files that you want to process, also it is possible to check the status of the job being processed. All this is possible through the OSCAR API and AWS services such as Amazon S3, AWS Lambda and AWS Batch, so it was necessary to use the *AWS SDK for JavaScript* which allows access to these services through a web application.

5.2 Architecture

Figure 5.2 shows the proposed architecture for the integration of ML models as serverless services on public and on-premises Clouds. For integration in AWS, the architecture is based on SCAR that allows Docker images to run, as functions that are triggered in response to events, in this case, when uploading a file to an S3 *bucket*. In the function creation process, the user specifies the execution mode taking into account if they are long duration jobs (*batch* execution mode executed in AWS Batch) or if they are short duration jobs (*lambda* execution mode executed in AWS Lambda). In addition, if the duration of the work is unknown, the user can choose the *lambda-batch* execution mode where it is first executed in AWS Lambda and before reaching the default timeout, the process is automatically delegated to AWS Batch.

In both AWS and OSCAR deployment methods, you can define three storage providers (Amazon S3, MinIO and EGI DataHub) for the input and output

¹²DEEP IAM - <https://iam.deep-hybrid-datacloud.eu/>

¹³Identity Pools - <https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-identity.html>

files. In the case of deployment on AWS, Amazon S3 was selected as the storage provider for the input and output files, with the aim of fully implementing the platform in a public Cloud provider.

The integration of the models in a private Cloud is done through the OSCAR framework, where users upload the files to a shared storage back-end, which automatically activates the execution of parallel invocations to the function responsible for processing each file. In this deployment method, to evaluate the versatility of the platform, MinIO and EGI DataHub were used as storage providers. The input files, accessible from the web interface, are stored in MinIO, so the user must include the credentials to access MinIO from the web interface. The results of the prediction are stored in the user's space in EGI DataHub, which is accessible via a link in the graphical interface.

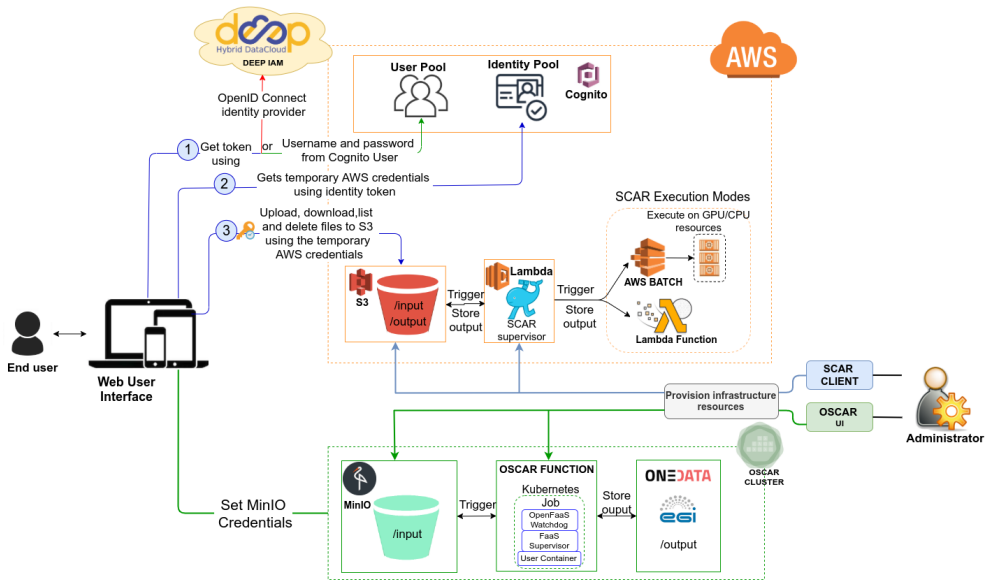


Figure 5.2: Architecture for the integration of Machine Learning models in AWS.

Previously, it is necessary for the administrator to create the functions that will make the model available through the SCAR *client* in the case of the deployment method in AWS and through the OSCAR user interface in the

on-premises deployment. In the case of SCAR it is necessary to specify the Docker image, execution mode, input and output folder and the script to execute to carry out the processing. In the case of OSCAR, the same characteristics are specified as in SCAR, except for the execution mode that is only applicable in the case of AWS.

As a result, a function is created that will be executed every time a file is uploaded to the input folder. The users can authenticate with their Amazon Cognito credentials or if they are DEEP users, they can do it through DEEP IAM identity provider. Once the user authenticates on the web, an identity token is obtained that allows access to the files that are stored in the S3 *bucket* in case of selecting the deployment method in AWS. In case of selecting OSCAR, it is necessary to enter the MinIO access credentials as explained above.

Developing this architecture enables machine learning and deep learning models to be integrated into a serverless platform that enables execution in public and on-premises Clouds. The open-source tools SCAR and OSCAR were used to create highly parallel event-driven file processing serverless applications in environments such as AWS Lambda, AWS Batch and on-premises Cloud. This implementation has enabled the creation of a serverless service on the AWS public platform and in a private Cloud, that grows elastically based on execution needs, and terminates the provisioned resources when they are no longer needed.

5.2.1 Computing platform and price of services

The implemented architecture has two fundamental branches, one fully developed in the AWS Cloud and the other in a local Cloud. It can automatically scale-to-zero to reduce implementation costs when the platform is not used. In the case of AWS the costs are generated when using Amazon S3, AWS Lambda and AWS Batch. Using AWS Batch does not incur in an extra cost apart from the cost of the EC2 instances created to run the application.

In the case of Amazon S3, two *buckets* were used, one to store the files to obtain the predictions of the models, which in turn works as a trigger for the Lambda function, and the other *bucket* is used to host the web interface. In this use case, Lambda functions have a memory of 1024MB and there is one function deployed for each integrated model. In the case of the models implemented in AWS Batch, jobs run on general-purpose m3.medium EC2 instances with 1 vCPU and 3.75 GBi of memory priced at \$0.067 per hour.

For the deployment process of the platform designed in multi-cloud, Infrastructure as Code (IaC) tools that rely on DevOps approaches are used. These are in charge of automating the infrastructure provision and delivery and configuration of the software for the deployment of the system. For example, to deploy OSCAR, IaC tools are used, such as EC3, Ansible and IM. EC3 provisions elastic virtual clusters through the IM on public (AWS, Google Cloud or Microsoft Azure) and on-premises (OpenNebula and OpenStack) infrastructure providers. The IM allows the deployment of complex and custom infrastructures defined in recipes for Ansible Galaxy and described using the TOSCA specification in order to manage the configuration of the underlying infrastructure. The implementation effort in the thesis has focused on supporting the availability of different accelerator resources in the framework of serverless computing, taking advantage of the infrastructure created through tools such as EC3, IM or Ansible Galaxy. The use of these tools facilitates the development of high-performance infrastructures at a higher level of abstraction.

The OSCAR framework was deployed on the Ramses platform, a cluster that was configured with characteristics similar to the compute environments (2 m3.medium instances) defined in AWS Batch. That is, a maximum of two nodes were configured with 1vCPU and 4GB of memory.

5.3 Use Case: Serverless services for machine learning model inference

This section evaluates the use of the proposed architecture for the execution of several case studies of ML models on the platform. In this sense, and with the aim of being able to integrate other available models, a detailed study of the use of the platform in obtaining the inference of ML models that are publicly accessible is provided. The objective is to be able to evaluate the advantages of the developed system, in terms of jobs processed/time unit.

5.3.1 Objectives of the experiments

The proposed event-based architecture integrates machine learning models with different execution strategies and methods. Specifically, three models from the DEEP catalog are used and the Darknet example available in the SCAR repository. As discussed in Section 5.1, one of the elements on which this platform is based is the DEEPaaS API, a tool developed in the DEEP project, which is why we have decided to use models from the DEEP catalog as use cases. In the deployment on AWS these models run in AWS Batch because they do not meet AWS Lambda's limitations. In order to test the *lambda* execution mode, the Darknet example was integrated into the platform. These models are described below:

- Audio Classifier¹⁴: This model is a tool for performing audio classification, which is previously trained in the 527 high-level classes in the AudioSet¹⁵. The input file is an audio file (compatible with most formats) and returns a JSON file with the top 5 predictions.
- Plant Species Classifier¹⁶: The plant species classifier uses an optimized neural network to identify plants using RGB images as input. As a result

¹⁴Audio Classifier - <https://marketplace.deep-hybrid-datacloud.eu/modules/deep-oc-audio-classification-tf.html>

¹⁵AudioSet - <https://research.google.com/audioset/dataset>.

¹⁶Plant Species Classifier - <https://marketplace.deep-hybrid-datacloud.eu/modules/deep-oc-plants-classification-tf.html>

a JSON file is obtained with the 5 main predictions. The previously trained model is based on plant images from iNaturalist¹⁷.

- Body Pose Detection¹⁸: The body pose detection model was originally created by Google¹⁹ and allows using the deep neural networks pose estimation in real time. With this model it is possible to detect the pose of one person or multiple people. As input it expects an RGB image and it is possible to obtain three types of output as specified by the developer. For example, it is possible to obtain a JSON file with the key points, an image indicating these points or a ZIP file with the JSON file and the image indicating the points of the body. For this case study, a ZIP file is obtained with the image and the JSON file resulting from the classification.
- Darknet example²⁰: Darknet²¹ is an open-source neural network written in C and CUDA. This example uses the YOLO (you only look once) library for real-time object detection, such as people, cars, animals, among others.

Figure 5.4 shows the workflow for processing a file. The following points explain how this process occurs from the time a file is uploaded to the platform until the classification result is obtained.

- The users authenticate on the web to access the classification services of the different models available. To authenticate, credentials are required from either the Amazon Cognito user pool or the DEEP IAM which is the external identity provider included in this application.
- Once the users have authenticated on the web, they have to select a method of deployment. If OSCAR is selected, it is necessary to configure

¹⁷iNaturalist - <https://www.inaturalist.org/>

¹⁸Body Pose Detection - <https://marketplace.deep-hybrid-datacloud.eu/modules/deep-oc-posenet-tf.html>

¹⁹Google body pose detection - <https://github.com/tensorflow/tfjs-models/tree/master/posenet>

²⁰Darknet example - <https://github.com/grycap/scar/tree/master/examples/darknet>

²¹Darknet - <https://pjreddie.com/darknet/>

the MinIO credentials in the SETTINGS tab as shown in Figure 5.3. If the deployment is on AWS, this step can be skipped.



Figure 5.3: Settings tab to configure access to MinIO.

- Then, one of the available ML models must be selected. When a model is selected, two links of interest are displayed, *Input example for models* and *Link to the model in the Catalog*, to access examples of input files and the catalog repository respectively. With this information, the user has more specialized information on the use of each of the models.
- Once a model has been selected, it is possible to upload the files to obtain the prediction result.
- The input and output files are stored in the storage providers indicated in the deployment of the functions. In the case of AWS, an S3 *bucket* for the input and output files. In the case of OSCAR, a MinIO *bucket* for the input files and EGI DataHub for the output files. This *bucket* has a directory structure that is important to understand to facilitate the integration of a model with the architecture. When deploying the function, a folder is created in the *bucket* with the name of each model (this is defined in the SCAR YAML and OSCAR user interface). Later,

within each folder there are two folders, *input* and *output*, and inside these folders a new folder is created (from the web interface) with the user's name. This structure allows users to only access their own data because the information displayed on the web interface is limited according to the authenticated user. This allows the development of a multi-tenant environment and add a trigger event for each model independently, making use of the same input *bucket*.

- Once the function finishes processing the input file, the result is stored in the *output* folder corresponding to the user and the selected model.
- The result of the prediction generated by the selected model can be downloaded from the user interface or accessed via a link to the user's space in EGI DataHub, if the OSCAR deployment method was selected.

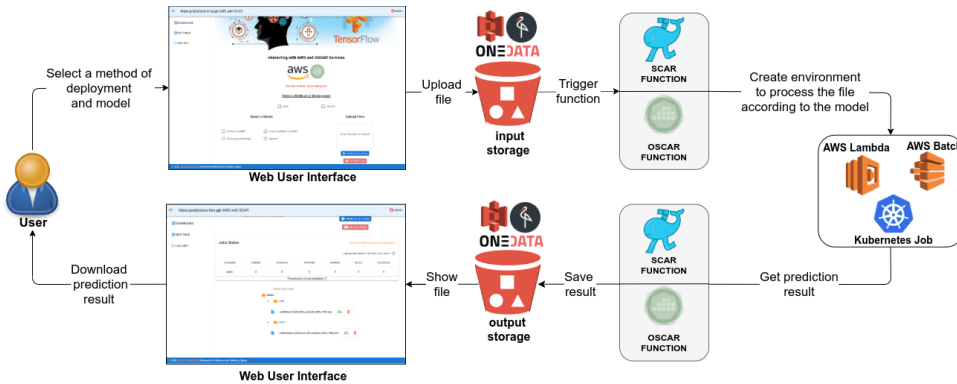


Figure 5.4: Simplified file processing workflow. Functions with different deployment methods selected, either on AWS or in a local cluster with OSCAR.

5.3.2 Experimental results

In order to test different execution methods and the integration of the models as services in the AWS Cloud provider and in a private Cloud, different experiments were developed. This section analyzes the results obtained by doing similar tests for the different deployment methods and execution modes in the case of AWS. The proposed architecture is fundamentally based on

SCAR and OSCAR for the execution of serverless functions in AWS (Lambda or AWS Batch) and in a private Cloud respectively.

The models belonging to the DEEP catalog use Tensorflow and the size of the Docker images are bigger than the limit allowed by Lambda (512MB), so the inference process for these jobs must be run in *batch* execution mode. In the Darknet example, the Docker image meets this limit, so execution in *lambda* mode was possible. Functions in OSCAR that run as Kubernetes jobs do not have any of these limitations.

Figure 5.5 shows the web interface once the user has authenticated using one of the available ways (Amazon Cognito user pool or DEEP IAM credentials). The panels that remain visible are *Select a Method of Deployment*, *Select a Model* and *Upload files*. To generate an event that will automatically trigger the function the files are uploaded in the following path `<bucket_name>/<model_name>/input/<username>/<input_file>`. This storage process is done automatically by the web interface taking into account the method of deployment (AWS or OSCAR), the user and the selected model.

The AWS Batch compute environment deployment process begins once the file has been uploaded to the S3 *bucket* and the event that triggers the SCAR function is generated. Deploying the AWS Batch compute environment can take several minutes because the EC2 instance needs to be provisioned and configured.

From the web interface it is possible to see the status of the jobs deployed in AWS Batch and in OSCAR, Figure 5.6. In the case of AWS, for each model that you want to integrate into the platform, a Lambda function is created. In addition to this, another Lambda function is created to check the status of the jobs through the AWS Batch API²², as shown in Figure 5.8. This function is activated when the user updates the status of the jobs from the web interface. Through the OSCAR API it is possible to check the status of the deployed jobs. This allows you to track the lifecycle of jobs that are processed in AWS

²²AWS Batch API - <https://docs.aws.amazon.com/batch/latest/APIReference/batch-api.pdf>

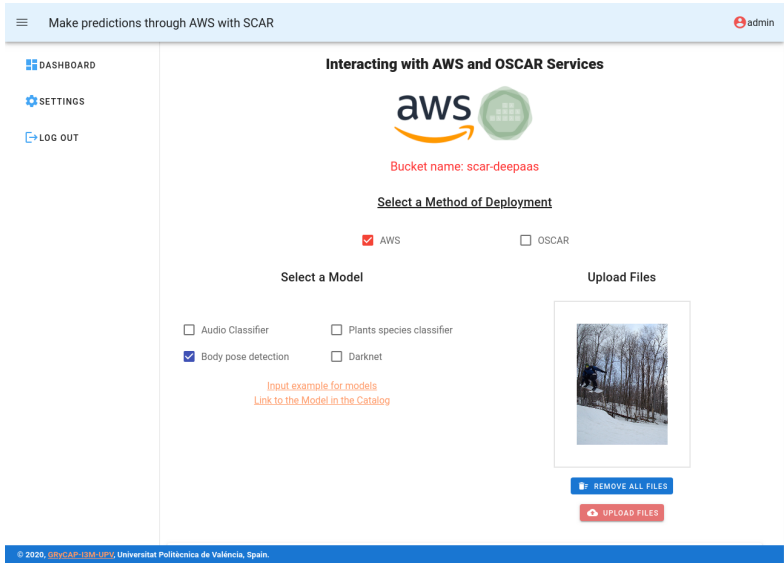


Figure 5.5: *Select Method of Deployment, Select Model and Upload Files panels of the Web Interface.*

Batch, which are long-running. These jobs constitute the inference phase of a job that was triggered by an event, in this case, uploading the file to be processed in an storage provider like S3 or MinIO.

As previously discussed, accessing AWS resources from the web interface requires the use of the AWS SDK. However, there are certain resources that are not yet available from the AWS SDK for Javascript due to cross-origin resource sharing (CORS) [47] policies, a mechanism that uses additional HTTP headers to allow a user agent (a browser in a Web context) to obtain permissions to access selected resources from a server, in an origin (domain) other than the one it belongs to.

The AWS Batch API is one of these services that CORS does not currently support. Therefore, it was necessary to implement the query from a Lambda function that does allow CORS policies to be activated from the web interface and makes the request from an environment where CORS does not apply, so it is possible to access the AWS Batch API. Figure 5.7 shows the code of the

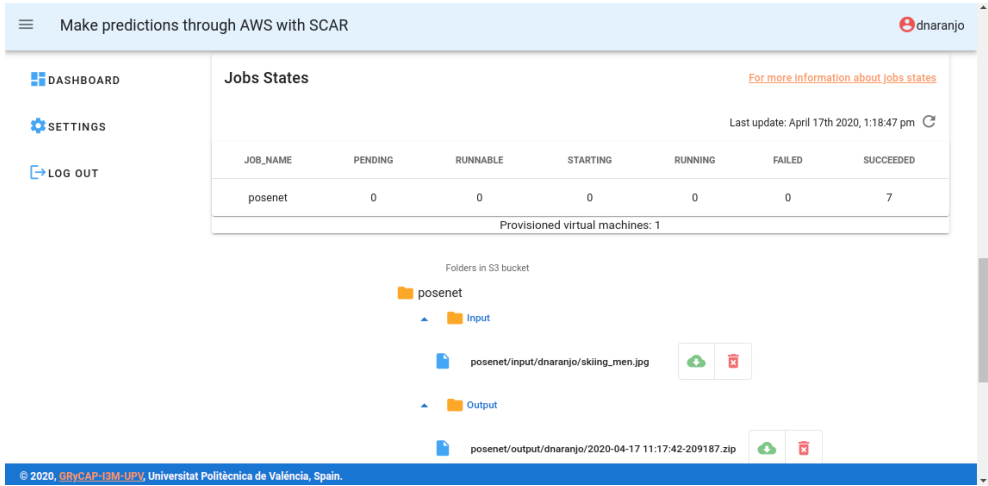


Figure 5.6: Web interface for the status of jobs and user files.

Lambda function implemented to query the status of the jobs. The code is written in Node.js²³ and only jobs that are in the PENDING, RUNNABLE, RUNNING, FAILED and SUCCEEDED states are checked. The information obtained is returned to the web interface and is displayed in the *Jobs States* panel.

The result of the prediction is stored in the path `<bucket_name>/<model_name>/output/<username>/<output_file>` in the storage provider defined in the creation of the function. The Figure 5.6 also shows the panel of the web interface where the files belonging to each user are displayed according to the specified model. From this panel it is possible to download the files or delete them if they are not necessary. In the case of OSCAR, the output files are stored in the user's space in EGI DataHub, which is accessible through a link.

Having the file system on this storage providers (Amazon S3, MinIO and ONE DATA) we achieve high availability, long-term preservation and remote accessibility from anywhere to the files used in the prediction.

²³Node.js - <https://nodejs.org>

```

1  const AWS = require('aws-sdk');
2
3  console.log('Loading function');
4
5  exports.handler = async (event, context) => {
6    // Log the received event
7    console.log('Received event: ', event);
8    // Get jobId from the event and add to params object
9    var states = ['PENDING', 'RUNNABLE', 'STARTING', 'RUNNING', 'FAILED', 'SUCCEEDED']
10   //var states = ["STARTING"]
11   var response = {
12     "PENDING": [],
13     "RUNNABLE": [],
14     "STARTING": [],
15     "RUNNING": [],
16     "FAILED": [],
17     "SUCCEEDED": [],
18     "computeEnv": []
19   }
20   for (let i = 0; i < states.length; i++){
21     const params = {
22       jobQueue: event.jobQueue,
23       jobStatus: states[i]
24     };
25     console.log(params)
26     // Get the Batch Job status
27
28     try {
29       var data = await new AWS.Batch().listJobs(params).promise();
30       response[states[i]].push(data)
31       console.log(response)
32     }
33     catch (err) {
34       console.error(err);
35       const message = `Error calling listJobs for: ${event.jobQueue}`;
36       console.error(message);
37       throw new Error(message);
38     }
39   }
40   var params_env = {
41     computeEnvironments: [
42       event.jobQueue
43     ]
44   }
45
46   var data_compute_env = await new AWS.Batch().describeComputeEnvironments(params_env).promise();
47   response["computeEnv"] = data_compute_env.computeEnvironments[0].computeResources.destRedvdCpus
48
49   return response;
50
51
52

```

Figure 5.7: Lambda function code to check the state of jobs through the API of AWS Batch.

As an example, Figure 5.9 shows the prediction result of the body pose detection model. On the left (a) the original image is shown, on the right (b) the image with the points of the body detected by the model is observed and in the bottom part (c) the result of the prediction is shown in JSON format. These files are stored in a ZIP file accessible from the web interface that generates the model during the processing of the input image.

As it was well addressed in other sections of the thesis in the case of serverless platforms with scale-to-zero, it is important to determine the *cold start* of the functions. On this platform that AWS Batch is used for the processing of functions, it is also important to analyze the startup time of the instances.

Figure 5.10 shows the processing times for 10 sample images used in the Darknet model, deployed on AWS Lambda. The graph shows that for the first execution the time is considerably longer because the function needs to first

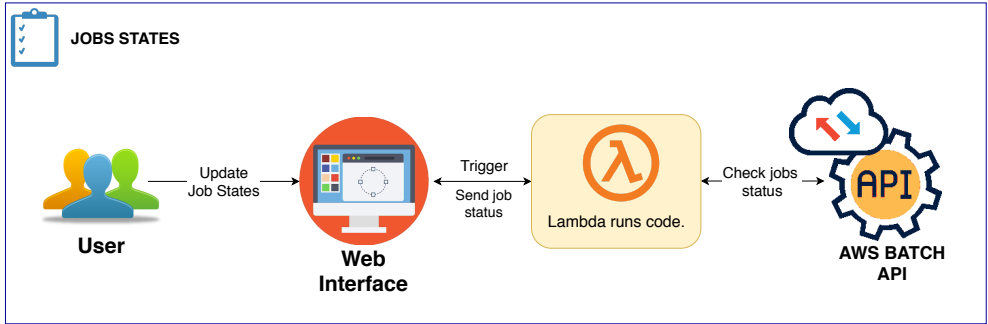


Figure 5.8: Process of querying, from the web interface, the status of the jobs in AWS Batch.



Figure 5.9: Example of the result obtained using the Body Pose Detection model.

carry out the configuration process, which involves downloading the Docker image containing the application code, starting a new execution environment, run the initialization code and run the main function. From this moment on, the other execution times are practically the same. After the first invocation where the function is initialized and affected by the cold start, the following invocations have a similar execution time of approximately 16 seconds. Notice that subsequent invocations will mostly reuse the aforementioned configuration, thus executing much faster.

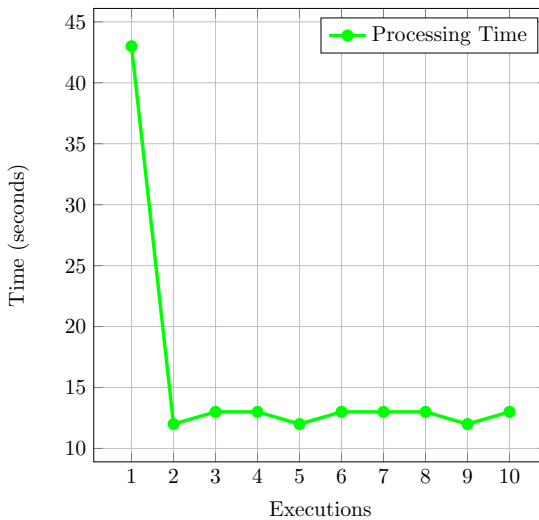


Figure 5.10: Execution times for 10 images in AWS Lambda.

Figure 5.11 shows the same experiment performed on AWS Lambda but in this case the 10 runs are performed with images of the Body Pose Detection model on AWS Batch. In the graph, the blue bar shows the time since the job is sent to the compute environment to be processed until a result is obtained. This required approximately 14 seconds, which was very consistent across the different executions.

On the other hand, the total execution time (yellow bar + blue bar) that is measured from the moment the job is created to the end of the processing is

very different. This behavior is due to the fact that in the first execution it is necessary to provision the resources of the instance that will process the jobs, hence it takes a little longer. The compute environment in AWS Batch is configured in this case for the maximum execution of 2 EC2 instances to service the requests. The 10 jobs are sent simultaneously and queued, the AWS Batch scheduler delegates them to the instances whenever there are resources for processing. Therefore, the processing time is divided into: waiting time for available resources and the time it takes to be processed once the work is delegated to the instance. Once there are available resources, the job changes its state and is processed in a few seconds as shown in the graph. Because these jobs are precisely those of long duration it was decided that the web interface should show a panel where the user could track the status of the jobs.

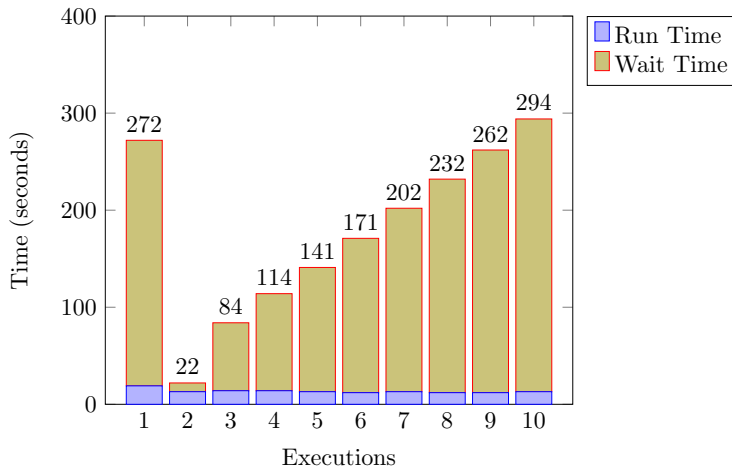


Figure 5.11: Execution times for 10 images in AWS Batch.

The information provided by AWS Batch allows knowing the total execution time (yellow bar + blue bar), which includes the waiting time in the job queue (yellow bar) and the file execution time (blue bar) once it has been delegated to the compute environment. In the case of AWS Lambda, it is only possible to know the total execution time (green line).

Figure 5.12 shows the times obtained in the same experiment performed in AWS Lambda and AWS Batch, but in this case the execution is carried out in a private cluster with OSCAR. OSCAR is deployed in an elastic Kubernetes cluster that grows or shrinks, based on the number of nodes, depending on the workload. To recreate a compute environment similar to AWS Batch, the cluster was configured with a maximum of 2 nodes with 1 vCPU and 4 GB of memory (equivalent to the two instances defined in AWS Batch).

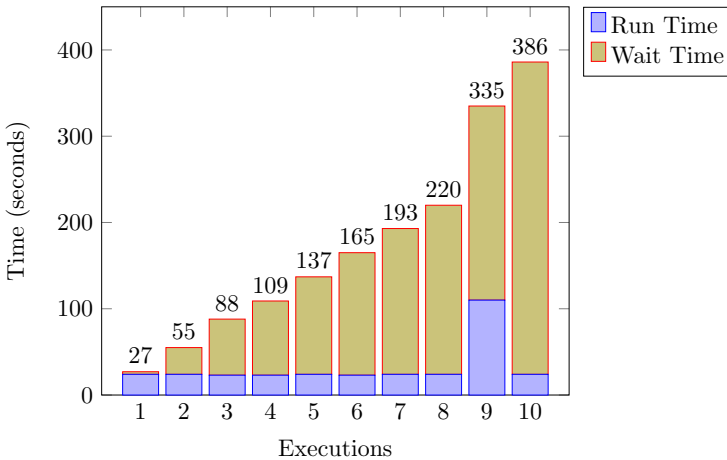


Figure 5.12: Execution times for 10 images in OSCAR cluster.

As in AWS Batch, the yellow bar shows the waiting time of the jobs in the queue and the blue bar the execution time. The execution time, as in AWS Batch, includes the download time of the input file, the processing time of that file and the upload time of the output file. In all cases, except execution number 4, the processing time is approximately equal to the processing times obtained in AWS Batch. It is important to note that the order of execution of the work depends on the configuration of Kubernetes scheduler. The total execution time of the 10 jobs is similar to those obtained in Batch (300 seconds in AWS Batch for 360 seconds in OSCAR approximately).

In execution number 4, the increase in execution time (blue bar) is striking. Since files with similar characteristics are used, this difference should not exist. It is important to note that the 10 simultaneous executions cause the OSCAR platform to deploy a new node, to cover the workload that has been generated. After several tests carried out, we were able to conclude that the increase in execution time, in this job, is due to an internal network problem when a new node is deployed, which causes the function to take longer to download the input file from the MinIO *bucket*, although the input file processing time remains the same. This problem is possibly due to the fact that when a new node is deployed a reconfiguration of the nodes is carried out, which causes the network connection to be lost for a few seconds. To verify this, the 10 jobs were submitted with both nodes active and the results showed that this increase in execution time did not occur. The different tests carried out allowed us to detect this problem, which is being worked on for its subsequent solution.

The deployment of a new node is done using CLUES, a tool presented in Section 4.2.1. The maximum number of nodes to deploy with CLUES has been set to two to simulate the same compute environment as in AWS Batch. After a few seconds of uploading the files to the input *bucket*, CLUES detects the need to deploy a new node to service the workload. The measured time for the deployment and configuration of a new node was approximately 5 minutes. Only one of the jobs, execution number 9 in Figure 5.12, was the one that was completed in the newly deployed node, hence the longest waiting and total execution time. After about three minutes, if there are no jobs to process, CLUES turns the node off again.

The results in each of the deployment methods are different. In the case of AWS Lambda we see how the first execution takes longer but the execution time of the other functions is similar because Lambda can process up to 1000 functions concurrently. In the case of AWS Batch this does not happen in the same way mainly because the jobs go to a queue where they wait for available resources to be scheduled. However, using AWS Batch in these architectures allows you to implement other applications that do not comply with the AWS Lambda restrictions, and it is also possible to use acceleration devices such as

GPUs to obtain better performance. The use of a local platform with OSCAR allows budget savings by obtaining execution times similar to public Cloud platforms such as AWS and without the restrictions of certain environments such as Lambda.

An analysis of the cost of the platform is shown in Table 5.2. The table breaks down the prices based on the AWS Lambda and Amazon EC2 resources used. In the case of AWS Lambda, AWS Batch and Amazon S3 its use on the platform has been made clear. AWS CloudWatch is for monitoring, storage and access to log files. In the case of Amazon S3 and Amazon CloudWatch, prices are given in a general way as they depend on the size of the files to be processed and the size of the logs stored respectively. For example, to process 1000 images of an average size of 150KB, which are the ones used in this case study, the cost of Amazon S3 would be \$ 0.0034 per month.

It is also important to note that using the free tier of all these AWS services can reduce the cost to virtually zero. For example, in the case of AWS Lamba, the free tier includes one million free requests per month and 400,000 GB per seconds of computing time per month, which would be sufficient for this use case.

The costs of implementation in a local Cloud depend largely on the volume of use and the capacity of the platform. Expenses related to electricity or specialized personnel who guarantee their activity and safety are not included. The minimum infrastructure should contemplate 2 nodes for the services of the medium-sized platform where the storage required for the input and output files depends largely on the intended use of the platform.

Using the Azure Total Cost of Ownership (TCO) Calculator²⁴, for a computing environment as the same characteristics as the one referenced in the local Cloud, and taking as reference the 386 seconds (Figure: 5.12) of the maximum execution time of a job, a cost of \$0.048 per execution is obtained. With AWS

²⁴Total Cost of Ownership - <https://azure.microsoft.com/en-us/pricing/tco/calculator/>

Table 5.2: Cost of the platform in a public (AWS) and local Cloud.

	Service	Resource Provided	Prices	Execution Time (s)	Cost (per execution)
	AWS Lambda	2048MB RAM	\$0,000033333/100ms	4	\$0,00013332
AWS	AWS Batch	m3.medium EC2 instances (1 vCPU 4GB)	\$0,067 per hour	297	\$0,0055275
	Amazon S3	First 50TB	\$0,023 per GB	-	-
	Amazon CloudWatch	Store and access log files	\$0,50 per GB	-	-
	Local Cloud	OSCAR	2 nodes (1vCPU 4GB)	-	386

Batch and AWS Lambda you get lower costs per execution as shown in table 5.2.

5.3.3 Discussion

The designed platform is based on tools that allow the implementation of the FaaS model. SCAR enables the deployment of containers out of Docker images as serverless functions in AWS. The execution modes in SCAR allow the processing of long-running models involving large Docker images, as well as short-run and image models that comply with AWS Lambda restrictions. The sudden increase in workload (uploading hundreds of files to the S3 bucket) can be handled seamlessly with the *lambda* execution mode by running the functions asynchronously. Requests are automatically queued until AWS Lambda supplies the on-demand computing capabilities necessary to process invocations in parallel.

On the other hand, the *batch* execution mode also allows handling large amounts of data that are executed in a longer time and with different characteristics that need to be provisioned through EC2 instances (virtual machines). Jobs are automatically queued and delegated to compute environments based on the available resources. Processing in compute environments can be done using acceleration devices by provisioning GPU-based virtual machines. In this way it is possible to limit the cost at the expense of parallel processing or provision more resources to obtain the results faster, depending on the user's budget.

Execution in a private Cloud making use of the OSCAR framework allows the implementation of machine learning and artificial intelligence models on a serverless platform in which no hardware costs are incurred and the lock-in imposed by public Cloud providers is avoided. The use of OSCAR allows the configuration of infrastructures that grow and decrease elastically and in addition to the possibility of scale-to-zero the working nodes in case of not using resources, which translates into energy efficiency. Another interesting element to keep in mind is that OSCAR is integrated with EGI Federated Cloud [66], an IaaS-type Cloud, made of private and academic Clouds with virtualized resources and built around open standards.

The research [68] performs a series of experiments to run Amazon MXNet machine learning framework in Lambda. Experimental results show that a serverless platform is suitable for machine learning model inference. In this research, the tests carried out are to obtain the prediction on models that are already integrated within the AWS platform and that comply with the limitations of Lambda. The fundamental objective is to measure the performance in terms of processing time, scalability and memory used in the inference of models that meet the limits imposed on AWS Lambda. In [115] SerFer is presented as a machine learning application inference system on the AWS platform. In this system the inference is restricted to AlexNet [78] and the implementation is based on a component that orchestrates the execution of the inference and that is constantly executed in an EC2 instance, which generates a permanent cost.

In the proposed solution, the field of application has been extended to models that are not integrated into the AWS platform, that use Tensorflow and that are not limited by Lambda restrictions, something that is pending in the research mentioned above. Furthermore, the use of these models is done in a simple way, through a web interface, so that users can obtain the prediction results without having specific skills working with AWS or machine learning models. One of the main advantages of the proposed platform is that it only generates cost when the services are being used, in addition to automatically scaling where the processing capacity is increased according to demand.

The system described in [22] presents serverless services for predicting deep learning models, called Barista²⁵. This approach proposes a local environment for the development of a framework that allows predictions to be run by selecting the virtual machine configuration depending on the service level objectives, cost and execution time. For a local cloud, it seems like a good solution, but it is necessary to have a machine powerful enough to implement the framework. However, the architecture designed in this thesis makes it easy to inference pre-trained machine learning models into public and private Cloud at a reduced cost.

The platform as an inference system for machine learning applications is designed as an architecture that automatically scales according to processing needs, which scales to zero by introducing a delay in the start of processing, while saving on budget. Once the function is initialized the results can be obtained in acceptable processing times.

Developing this architecture greatly facilitates the inference phase of ML models by making use of public and private Clouds. The users, according to their possibilities, can select whether to deploy the models in a local Cloud with sufficient resources for the processing of the files or otherwise, execute the inference phase on the AWS infrastructure. The predictions are obtained through serverless services which favors the reduced cost, since expenses are only incurred when the application is used. This solution represents a step forward in simplifying the inference phase of machine learning models and artificial intelligence applications.

5.4 Chapter Conclusions

The third serverless computing strategy of the research was addressed in this chapter. The presented platform facilitates the use of artificial intelligence models through a web interface using AWS or a private Cloud as back-end execution. The architecture used SCAR for running applications packaged in

²⁵Barista - <https://zivgitlab.uni-muenster.de/pria/Barista>

Docker images as functions in AWS Lambda, which are triggered in response to certain events. It also adopted OSCAR to deploy serverless function in an on-premises Cloud. Model integration was implemented through the DEEPaaS API a service that provides easy access to machine learning, deep learning, and artificial intelligence models. To facilitate access to the models by users, a web interface was developed that allows interaction with the model's functionalities in the inference phase.

As use cases, models with different deployment methods were used, in OSCAR or AWS, and with execution modes that included AWS Lambda, for the execution of short jobs and that comply with the limitations of Lambda, and AWS Batch for the execution of long jobs that did not comply with the Lambda limitations. Results showed scale-to-zero capabilities to minimize service costs and automatic scaling capabilities as the number of jobs to process increased. A discussion of these results was conducted taking into account other platforms that provide access to machine learning and artificial intelligence models.

Conclusions and Future Work

This chapter presents the general conclusions of this thesis. First, the main contributions of each of the designed architectures are summarized. Next, future works that may continue to be developed as part of the research are described. Finally, the chapter concludes with a reference to the scientific publications and projects that have supported the lines of investigation of the thesis.

6.1 Conclusions

With the development of this research, a response has been given to the general objective set at the beginning of the thesis, referring to the design of different serverless computing strategies. The designs have been developed in public and private Cloud platforms, together with the use of acceleration devices that allowed to efficiently address the problems in data processing.

This research has focused on different serverless computing strategies that allow the execution of scientific applications and the resolution of larger problems. The proposed solutions are developed in the AWS Cloud using SCAR and other AWS services. Also for local implementations OSCAR is used and in OSCAR, a serverless on-premises framework that runs on top of an elastic Kubernetes cluster.

Firstly, CloudTrail-Tracker has been developed, a serverless platform that provides aggregated information about the use of a shared AWS account. The tool is composed of an event processing back-end that automatically collects evidence of user activity. The platform has focused on the registration of activities developed by students enrolled in AWS-related subjects.

The implemented educational web dashboard includes graphic components that facilitate the interpretation of the information by the instructor and the students. The interface allows students to know precisely the degree of completion of each laboratory practice in order to promote self-regulation. The dashboard also helps system administrators control the resources that are used in AWS accounts. The tool has been put into production with practically no cost and the initial satisfaction results point to the goodness of the support to facilitate progress in the completion of courses related to AWS.

The development of CloudTrail-Tracker is completely serverless in the AWS Cloud, where the back-end and the front-end work without the need to provision virtual machines (EC2 instances) at practically zero cost, which constitutes a model of reference for the design of future applications in the Cloud. As a result of the research carried out, we consider that CloudTrail-Tracker constitutes a tool that provides great benefits (control and monitoring) in environments where a shared AWS account is used, so the use of the application is restricted to AWS. Despite the fact that AWS offers the CloudTrail service to find out the activities carried out in the last 90 days, this time is insufficient when it comes to applications that require querying for the events in a longer period of time. In addition, the way events are represented in CloudTrail hinders the ability to obtain aggregated metrics.

These limitations detected in this service are resolved with the development of CloudTrail-Tracker. Despite the tool is currently focused on the education sector, it is considered a general-purpose tool. Changes could be easily applied to deal with other project-based wider scope.

Secondly, the use of GPUs was integrated with different virtualization techniques in the serverless on-premises OSCAR platform. Fundamentally, we addressed remote GPU virtualization through rCUDA and lightweight virtualization with NVIDIA-Docker. In particular, the integration of GPUs into an event-driven computing-based serverless platform is evaluated to provide accelerated support for functions that run as jobs in an elastic Kubernetes cluster.

The implemented use case constitutes a real study of echocardiographic movies that uses machine learning techniques to carry out the segmentation process of the movies in images and later the classification of these images depending on the angle of view. Video processing is done in an on-premises serverless Cloud, with nodes enabled to support GPU and CPU.

The results indicate that in single video processing the best approach is the direct access to the GPU. Virtualizing the GPU with rCUDA produces better results than those obtained with CPU usage, but they are far from those obtained with native access to the GPU. However, in the case of rCUDA the ability to support multiple applications access to the same GPU proved to be a powerful feature to support improved parallelism on serverless platforms. The integration of rCUDA in OSCAR constitutes a starting point in the shared access of a GPU by multiple applications on serverless platforms.

The integration of GPUs in a serverless platform constitutes a fundamental element in the adoption of this technology for the processing of scientific applications that require the use of acceleration devices. Until now, there was a certain belief that serverless platforms were limited to CPU usage, as is the case with the large public Cloud providers, that provide these services without access to acceleration devices. With the virtualization

techniques studied in the development of the research, we integrated GPUs in a serverless on-premises platform that demonstrated its applicability by improving performance in terms of processing time. Furthermore, the analysis of the different virtualization techniques used in the study allowed to detect limitations in the integration of these devices in orchestration systems such as Kubernetes, when reserving the use of a GPU for an application, which should be addressed in future works.

Finally, in order to make it easier for users to invoke machine learning models, a serverless cloud platform has been implemented with a simple web interface, using AWS and OSCAR as execution *back-end*, that offers scaling to zero to minimize service costs. Users only need to upload their files through the web interface to generate concurrent processing. The application processes the files and leaves the prediction results in the web interface accessible by users.

Being able to interact with machine learning models from a serverless platform, without the need to define complex jobs and through a web interface, represents a step forward in simplifying the adoption of these models by end users. The development of a tool of these characteristics allows reducing the distance between artificial intelligence, serverless computing and the user, providing a complete abstraction in the use of these technologies. The study carried out demonstrates the benefit of using serverless platforms to address scientific problems related to machine learning and artificial intelligence. Apart from the integration of the inference phase, it is necessary to carry out an analysis to be able to implement the training of these models, which can greatly benefit from the use of acceleration devices. The level of abstraction introduced in the developed platform allows inexperienced users to interact with more complex machine learning models.

Stateless, event-driven behavior, automatic elasticity, and short execution times are the key elements of serverless computing. Machine learning and artificial intelligence are one of the fastest growing areas of computing that can benefit from these features. Serverless computing has been shown to be useful for inference and prediction in public and on-premises cloud environments. The

serverless computing strategies proposed in this thesis foresee a rapid growth in the adoption of the serverless model in scientific applications that require high workloads, use of acceleration devices, which can grow and decrease according to demand and all this at a minimum cost. It is important to note that the serverless model is oriented for bursts of short-lived jobs. Otherwise the cost, in the case of the public Cloud can be dramatically higher compared to the use of virtual machines [43].

The development of this work has allowed to extract a series of lessons. The first one, GPU-enabled computing back-ends for FaaS constitute a very good scenario for avoiding resource lock-in and facilitating time sharing of resources. The results obtained in the experiments carried out support the use of open source serverless platforms, integrated with different virtualization techniques of acceleration devices, for the resolution of complex scientific problems, at a reduced cost. The second one, user-level containerization is a highly interesting working area that overcomes the limitations of multi-tenant data centers. The third one, Cloud orchestration and automated DevOps have facilitated strongly the execution of reproducible experiments. The fourth one, the integration of machine learning and artificial intelligence models in a serverless computing platform facilitates the use of highly complex models by users without experience in this technology. The fifth one, the advantages that serverless computing provides in terms of reduced costs, developers never have to deal with servers and the scalability that it inherently provides, ensures that it is a field of Cloud Computing from which it remains much to be exploited and investigated, for its applicability in solving scientific problems that constitute a challenge today.

With the results obtained in this thesis, we conclude by stating that all the objectives proposed at the beginning of the research have been successfully achieved. With the development of the thesis, different tools based on the serverless model have been generated for users to take advantage of its benefits without a deep knowledge of the technologies used.

6.2 Future Works

Future work includes firstly the maintenance of these tools with the aim of including new functionalities, which implies an arduous work of research and development. Furthermore, it would be anticipated to speak of final versions because there are always improvements that can be implemented.

In the field of learning analytics with CloudTrail-Tracker it is possible to improve the information displayed by customizing for the role of the users who makes use of the tool. It is planned to include additional panels so that the instructor can detect problems of excessive use of resources during a practical laboratory session. For the students, the inclusion of a system that allows showing the average performance with respect to their peers, in addition to accessing the history of events and the degree of completion of the practices, is expected.

The definition of learning itineraries is also proposed so that it is possible to detect differences between the events developed in a practice and the path defined by the instructor. This will allow the teacher to offer specialized help and complementary material to students that the system detects with greater problems. In order to motivate students to carry out the activities, it is planned to include gamification techniques.

In the field of acceleration device virtualization, it is planned to include a plugin in the Kubernetes cluster that will eliminate the bottleneck in the processing of multiple parallel videos currently introduced by the current NVIDIA plugin. This will allow the sharing of GPUs by the same function (Kubernetes pod), which can have a decisive influence on the execution times of the applications. In addition, integrating the OSCAR and SCAR architecture will allow the processing of hybrid workloads.

Finally, in the availability of serverless services for machine learning models on the AWS platform, there are two fundamental lines of action. One focused on incorporating other existing open catalog machine learning models. The other

line refers to including the training/re-training capacity of the models from the web interface, which requires a more exhaustive analysis of the implications. This would allow full adoption of serverless technology in machine learning and artificial intelligence applications.

6.3 Scientific production and fundings for this thesis

The realization of this doctoral thesis has led to the publication of a series of research articles, publications at conferences and research stays, detailed below:

- *Naranjo, D.M.; Prieto, J.R.; Moltó, G.; Calatrava, A. A Visual Dashboard to Track Learning Analytics for Educational Cloud Computing. Sensors 2019, 19, 2952. <https://www.mdpi.com/1424-8220/19/13/2952>. JCR. Instruments & Instrumentation 15/61. Q1.*

Abstract: Cloud providers such as Amazon Web Services (AWS) stand out as useful platforms to teach distributed computing concepts as well as the development of Cloud-native scalable application architectures on real-world infrastructures. Instructors can benefit from high-level tools to track the progress of students during their learning paths on the Cloud, and this information can be disclosed via educational dashboards for students to understand their progress through the practical activities. To this aim, this paper introduces CloudTrail-Tracker, an open-source platform to obtain enhanced usage analytics from a shared AWS account. The tool provides the instructor with a visual dashboard that depicts the aggregated usage of resources by all the students during a certain time frame and the specific use of AWS for a specific student. To facilitate self-regulation of students, the dashboard also depicts the percentage of progress for each lab session and the pending actions by the student. The dashboard has been integrated in four Cloud subjects that use different learning methodologies (from face-to-face to online learning) and the students positively highlight the usefulness of the tool for Cloud

instruction in AWS. This automated procurement of evidences of student activity on the Cloud results in close to real-time learning analytics useful both for semi-automated assessment and student self-awareness of their own training progress.

- *Naranjo, D.M., Gomes, J., David, M., Blanquer, I., & Moltó, G. Comparison of Container-based Virtualization Tools for HPC Platforms. IBERGRID 2019 - Delivering Innovative Computing and Data services to Researchers (23-26 de septiembre de 2019): Comparison of Container-based Virtualization Tools for HPC Platforms. LIP Indico (Indico). (n.d.). Retrieved April 8, 2020, from <https://indico.lip.pt/event/575/contributions/1856/>*

Abstract: Virtualization technologies are a fundamental element in Cloud Computing. Docker is the most known and used container platform worldwide. It is designed for microservices virtualization and application delivery but its model does not fit well with High-Performance Computing (HPC) platforms. HPC environments are multi-user systems where users should only have access to their own data and computing resources. Misconfigured Docker installations pave the way for privilege escalation, including the ability to access other users' data and, at the same time, gaining control of the cluster and computing resources.

In the world of HPC, the focus of containerised applications is not necessarily on DevOps, but on the ability to minimise HPC node configuration and manage applications' software dependencies through containers. Several open source initiatives have addressed this problem of bringing containers to the HPC space such as Singularity, Shifter, CharlieCloud and uDocker. In this sense, Singularity seems to be the most popular container system for HPC centres, but there are alternatives such as uDocker that support the execution of containers in user space, a key feature in HPC platforms. Therefore, it is important to analyze the benefits and drawbacks of these solutions when they are deployed in real HPC system and applied to scientific production applications.

All these tools, with potentially similar characteristics, bring the benefits of the containers to the HPC world. However, it is important to analyze important metrics in order to determine the advantages of one over another. The fields to analyze include, but are not limited to: interaction with Docker, support for Graphics Processing Unit (GPU), support for low-latency interconnects such as InfiniBand, support for Message Passing Interface (MPI), security and portability, privilege model, integration with Local Resource Management Systems (LRMS), among others. The objective of this communication is to show the behaviour and limitations of different container technologies in the context of HPC systems.

- A. Pérez, S. Risco, D. M. Naranjo, M. Caballer and G. Moltó, "On-Premises Serverless Computing for Event-Driven Data Processing Applications," *2019 IEEE 12th International Conference on Cloud Computing (CLOUD), Milan, Italy, 2019*, pp. 414-421.<https://ieeexplore.ieee.org/document/8814513>. GGS Class 2 in the GII-GRIN-SCIE index of conferences.

Abstract: The advent of open-source serverless computing frameworks has introduced the ability to bring the Functions-as-a-Service (FaaS) paradigm for applications to be executed on-premises. In particular, data-driven scientific applications can benefit from these frameworks with the ability to trigger scalable computation in response to incoming workloads of files to be processed. This paper introduces an open-source framework to achieve on-premises serverless computing for event-driven data processing applications that features: i) the automated provisioning of an elastic Kubernetes cluster that can grow and shrink, in terms of the number of nodes, on multi-Clouds; ii) the automated deployment of a FaaS framework together with a data storage back-end that triggers events upon file uploads; iii) a service that provides a REST API to orchestrate the creation of such functions and iv) a graphical user interface that provides a unified entry point to interact with the aforementioned services. Together, this provides a framework to deploy a computing platform to create highly-parallel event-driven file-processing serverless

applications that execute on customized runtime environments provided by Docker containers that run on an elastic Kubernetes cluster. The usefulness of this framework is exemplified by means of the execution of a data-driven workflow for optimised object detection on video. The workflow is tested under three different workloads which process ten, a hundred and a thousand functions. The results show that the presented architecture is able to process such workloads taking advantage of its elasticity to make a sensible usage of the resources.

- Research stay of 3 months (May, June and July 2019) at the Laboratory of Instrumentation and Particle Physics (LIP) in Lisbon.

Summary of the Stay: One of the fundamental research lines of the thesis is containerized computing. The most used tool for the execution of containers is Docker but in user spaces, such as HPC centers, Docker is not very used because it needs root access. There are other tools that allow the execution of containers in these environments such as udocker. In the research stay, we worked with the udocker developers, which allowed us to understand all the functionalities of the tool at a greater level of depth. udocker is integrated in SCAR to execute containers out of Docker images in environments where root access is not available, as it is the case of AWS Lambda. These tools were used in the implementation of the platform described in the section 5. The work in those months was fundamentally oriented to carry out a study and comparison of tools that would allow the execution of containers in user spaces. The tools analyzed were Singularity, Shifter, CharlieCloud and udocker, since they are the most outstanding in the field of execution of Docker containers in user spaces. The results of this comparison were presented at the IBERGRID 2019 conference referenced above and a summary of these results is described in the section 5.1.2. As conclusions of the work done in the stay it was determined that udocker in comparison with the other tools analyzed is more user-oriented, since even the command line imitates the Docker.

- *Naranjo D.M., Risco, S., de Alfonso, C., Pérez, A., Blanquer, I., and Moltó, G. (2020). Accelerated serverless computing based on GPU virtualization. Journal of Parallel and Distributed Computing, 139, 32-42. <https://doi.org/10.1016/j.jpdc.2020.01.004>. JCR. Comp. Science, Theory & Methods, 43/105. Q2.*

Abstract: This paper introduces a platform to support serverless computing for scalable event-driven data processing that features a multi-level elasticity approach combined with virtualization of GPUs. The platform supports the execution of applications based on Docker containers in response to file uploads to a data storage in order to perform the data processing in parallel. This is managed by an elastic Kubernetes cluster whose size automatically grows and shrinks depending on the number of files to be processed. To accelerate the processing time of each file, several approaches involving virtualized access to GPUs, either locally or remote, have been evaluated. A use case that involves the inference based on deep learning techniques on transthoracic echocardiography imaging has been carried out to assess the benefits and limitations of the platform. The results indicate that the combination of serverless computing and GPU virtualization introduce an efficient and cost-effective event-driven accelerated computing approach that can be applied for a wide variety of scientific applications.

- *Naranjo, Diana M, Risco, S., G. Moltó and Blanquer, I., "A Serverless Gateway for the Execution of Open Machine Learning Models on AWS" Gateways 2020, Conference hosted online. Contribution type: Demo.*

Abstract: Serverless computing has become a paradigm for the implementation and development of a wide range of applications based on events in the Cloud. Today, with the rise of machine learning models and artificial intelligence applications, both Cloud providers and large companies are focusing their efforts on offering these services to their customers. This article proposes a serverless computing platform to facilitate running tasks for inference using machine learning models on

the AWS Cloud provider. The back-end grows elastically based on execution needs and features scale-to-zero features to minimize costs, while the front-end provides a simplified user experience to trigger the inference phase for machine learning models. The results demonstrate that the proposed solution constitutes a step forward on simplifying the Cloud-based execution of machine learning and artificial intelligence models.

The doctoral thesis has been developed within the “Subvenciones del Programa Santiago Grisolia” for the promotion of scientific research, technological development and innovation in the Comunitat Valenciana, grant number GrisoliaP/2017/071 from the Conselleria d’Educació of the Generalitat Valenciana.

The author would like to thank the Spanish “Ministerio de Economía, Industria y Competitividad” for the project “BigCLOE” under grant reference TIN2016-79951-R. This project aims to develop an open platform to deploy frameworks for massive data processing (Big Data) and High Performance Computing frameworks based on containers. The platform will be developed through multiple advanced Cloud infrastructures that optimise the energy, the computational resources and manage elasticity capabilities. It also aims to isolate the execution of workloads and access to specific computing devices such as FPGA and GPU, through containers.

Part of this work was also supported by the project DEEP-Hybrid-DataCloud, European Union’s Horizon 2020 Research and Innovation Programme, under Grant 777435. The general objective of this project is to promote the use of intensive computing services by different communities and research areas, and their support by the corresponding e-Infrastructure providers and open source projects. The project will integrate and improve existing components in the Cloud ecosystem, developing a service that supports intensive computing techniques that require access to specialized devices such as GPUs, for the processing of large datasets.

6.4 Software Developments

This section refers to the open-source developments that are part of the research process, as well as a short summary of the repository and the author's contribution.

- **CloudTrail-Tracker:** The URL of the service in production is `http://cloudtrailtracker.cursocloudaws.net` and it is accessible as a demonstration through the credentials (user: demo/password: demoDem0!). This research has two fundamental repositories on GitHub:
 1. **Back-end Development** (<https://github.com/grycap/cloudtrail-tracker>): CloudTrail-Tracker is a tool that provides enhanced information on the use of multiple users of an AWS account. It essentially consists of a serverless back-end composed of an AWS Lambda function, which is activated for the storage in Amazon DynamoDB of the information collected by AWS CloudTrail in Amazon S3. It also has a REST API based on the Amazon API Gateway service that queries the events stored in DynamoDB through a Lambda function.
 2. **Front-end development** (<https://github.com/grycap/cloudtrail-tracker-ui>): CloudTrail-Tracker-UI is a web portal based on Vue.js, a Javascript framework that allows the development of user interfaces. The web application consults the CloudTrail-Tracker REST API to view stored information related to the use of AWS services by different users who share an AWS account. The interface authentication process is built on AWS Cognito, an AWS tool that makes it easy to control access to web and mobile applications. The web service is also serverless as it is available through Amazon S3.

The development of the project had the contribution of several developers. In this sense, the main contribution of the author of this thesis was

related to the general design of the architecture, establishing security configurations, developing the web portal, doing a cost analysis, and, finally providing support for the maintenance of the implemented platform.

- OSCAR (Open Source Serverless Computing for Data-Processing Applications) <https://github.com/grycap/oscar>: OSCAR is an open source platform to support the Functions as a Service (FaaS) computing model for file processing applications. OSCAR is based on the processing of files through events that are generated from uploading a file to a storage system such as Amazon S3 or MinIO. Processing runs in custom runtime environments provided by Docker containers running on an elastic Kubernetes cluster. The platform can be automatically deployed across multiple clouds to create highly-parallel event-driven file-processing serverless applications.

The development of this platform has the participation of several developers. In this sense, the main contribution of the author of this thesis was related to the development of a web interface that would allow users to display and configure functions in a visual and simple way, in addition to the development of a server that, on the one hand, served the interface web and on the other hand allows interaction with underlying OSCAR technologies such as OpenFaaS and MinIO. In addition, the author also implemented GPU support in OSCAR to accelerate the execution of functions and supported the different use cases explained in previous sections.

- Serverless Machine Learning Inference <https://github.com/grycap/scar-deepaas-ui>: The objective of this platform is to facilitate the execution of machine learning and artificial intelligence models through a web interface. This platform is composed of a back-end, in which the serverless functions are deployed, one for each model, for the processing of the input files. The deployment of the functions in AWS is through SCAR, a framework to run Docker container images transparently in AWS

Lambda. The serverless OSCAR platform is used for the deployment of the models in a private Cloud.

In addition, the platform has a user interface served from Amazon S3 that allows:

1. Select whether to run the inference phase on AWS (public Cloud provider) or OSCAR (a serverless platform that can be deployed on-premises).
2. Select the model from which you want to obtain the prediction result, in this case four models were selected as case studies, three models from the DEEP Open Catalog¹ and another model from the SCAR repository².
3. Upload the files to process either to Amazon S3 or MinIO depending on whether the deployment method has been selected in AWS or OSCAR respectively. In addition, the result of the prediction can be consulted through the web if the deployment is carried out in AWS or in the user's EGI DataHub³ space.

The design and implementation of this architecture has been developed by the author of this thesis with the aim of facilitating the inference process of machine learning and artificial intelligence models through serverless platforms. The inference phase is fundamentally based on DEEPaaS API, a REST API that allows easy access to machine learning and artificial intelligence models. For its integration with the developed platform, it was necessary to add a new functionality that would allow obtaining the result of the prediction through the command line. The integration of machine learning and artificial intelligence models in on-premises and public serverless platforms allows exploring new solutions in intensive data processing.

¹DEEP Open Catalog - <https://marketplace.deep-hybrid-datacloud.eu/>

²SCAR Repository - <https://github.com/grycap/scar/tree/master/examples/darknet>

³EGI DataHub - <https://datahub.egi.eu>

Bibliography

- [1] *Ad Hoc Big Data Processing Made Simple with Serverless MapReduce / AWS Compute Blog*. URL: <https://aws.amazon.com/es/blogs/compute/ad-hoc-big-data-processing-made-simple-with-serverless-mapreduce/> (visited on 03/24/2020) (cit. on p. 21).
- [2] Carlos Alfonso, Amanda Calatrava, and Germán Moltó. “Container-based Virtual Elastic Clusters”. In: *Journal of Systems and Software* 127 (Jan. 2017). DOI: 10.1016/j.jss.2017.01.007 (cit. on p. 22).
- [3] Carlos de Alfonso, Miguel Caballer, Fernando Alvarruiz, and Vicente Hernández. “An energy management system for cluster infrastructures”. In: *Computers & Electrical Engineering* 39.8 (2013), pp. 2579–2590. URL: <http://www.sciencedirect.com/science/article/pii/S0045790613001365> (cit. on p. 73).
- [4] Alibaba. *Alibaba Cloud Function Compute*. URL: <https://www.alibabacloud.com/products/function-compute> (cit. on p. 26).

- [5] M. Amaral, J. Polo, D. Carrera, I. Mohomed, M. Unuvar, and M. Steinder. “Performance Evaluation of Microservices Architectures Using Containers”. In: *2015 IEEE 14th International Symposium on Network Computing and Applications*. 2015, pp. 27–34 (cit. on p. 2).
- [6] Amazon. *Amazon Simple Storage Service (Amazon S3)*. URL: <http://aws.amazon.com/s3/> (visited on 01/01/2011) (cit. on p. 6).
- [7] Amazon. *Amazon Web Services (AWS)*. URL: <http://aws.amazon.com> (cit. on p. 5).
- [8] Amazon Web Services. *AWS Lambda*. URL: <https://aws.amazon.com/lambda> (cit. on p. 26).
- [9] Apache. *OpenWhisk*. URL: <https://openwhisk.apache.org/> (cit. on p. 28).
- [10] *Apache Mesos*. URL: <http://mesos.apache.org/> (cit. on p. 6).
- [11] Carlos Arango, Rémy Dernas, and John Sanabria. “Performance Evaluation of Container-based Virtualization for High Performance Computing Environments”. In: *Revista UIS Ingenierías* 18.4 (2017), pp. 31–42. ISSN: 1657-4583. DOI: 10.18273/revuin.v18n4-2019003 (cit. on pp. 22, 23).
- [12] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. *A view of cloud computing*. 2010. DOI: 10.1145/1721654.1721672. URL: <http://portal.acm.org/citation.cfm?doid=1721654.1721672> (cit. on pp. 4, 20).
- [13] Ammar Ahmad Awan, Ching-Hsiang Chu, Hari Subramoni, and Dhableswar K. Panda. “Optimized Broadcast for Deep Learning

- Workloads on Dense-GPU InfiniBand Clusters: MPI or NCCL?” In: *Proceedings of the 25th European MPI Users’ Group Meeting*. EuroMPI’18. Barcelona, Spain: Association for Computing Machinery, 2018. ISBN: 9781450364928. DOI: 10 . 1145 / 3236367 . 3236381. URL: <https://doi.org/10.1145/3236367.3236381> (cit. on p. 9).
- [14] *AWS Batch Integration | SCAR Documentation*. URL: <https://scar.readthedocs.io/en/latest/batch.html> (visited on 03/02/2020) (cit. on p. 97).
- [15] *AWS Lambda | Pricing*. URL: <https://aws.amazon.com/lambda/pricing/> (visited on 04/03/2020) (cit. on p. 47).
- [16] *AWS Named as a Leader in Gartner’s Infrastructure as a Service (IaaS) Magic Quadrant for the 9th Consecutive Year | AWS News Blog*. URL: <https://aws.amazon.com/es/blogs/aws/aws-named-as-a-leader-in-gartners-infrastructure-as-a-service-iaas-magic-quadrant-for-the-9th-consecutiveyear/> (visited on 07/27/2020) (cit. on p. 20).
- [17] *AWS Named as a Leader in Gartner’s Magic Quadrant for Cloud AI Developer Services | AWS News Blog*. URL: <https://aws.amazon.com/es/blogs/aws/aws-named-as-a-leader-in-gartners-magic-quadrant-for-cloud-ai-developer-services/> (visited on 07/27/2020) (cit. on p. 21).
- [18] *aws-samples/pywren-workshops: Various workshop labs that make use of pywren to massively process data in parallel with AWS Lambda*. URL: <https://github.com/aws-samples/pywren-workshops> (visited on 03/24/2020) (cit. on p. 21).

- [19] A. Azab. “Enabling Docker Containers for High-Performance and Many-Task Computing”. In: *2017 IEEE International Conference on Cloud Engineering (IC2E)*. 2017, pp. 279–285 (cit. on p. 24).
- [20] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, and Philippe Suter. “Serverless computing: Current trends and open problems”. In: *Research Advances in Cloud Computing*. Singapore: Springer Singapore, 2017, pp. 1–20. ISBN: 9789811050268. DOI: 10.1007/978-981-10-5026-8_1. arXiv: 1706.03178. URL: http://link.springer.com/10.1007/978-981-10-5026-8_1 (cit. on pp. 26, 37, 39).
- [21] Sushmita N Bhatnagar. “An audit of malignant solid tumors in infants and neonates.” In: *Journal of neonatal surgery* 1.1 (2012), p. 5. ISSN: 2226-0439. URL: <http://www.ncbi.nlm.nih.gov/pubmed/26023364> (cit. on p. 8).
- [22] A. Bhattacharjee, A. D. Chhokra, Z. Kang, H. Sun, A. Gokhale, and G. Karsai. “BARISTA: Efficient and Scalable Serverless Serving System for Deep Learning Prediction Services”. In: *2019 IEEE International Conference on Cloud Engineering (IC2E)*. 2019, pp. 23–33. URL: <https://arxiv.org/abs/1904.01576> (cit. on p. 121).
- [23] Anirban Bhattacharjee, Yogesh Barve, Shweta Khare, Shunxing Bao, Aniruddha Gokhale, and Thomas Damiano. “Stratum: A Serverless Framework for the Lifecycle Management of Machine Learning-based Data Analytics Tasks”. In: *2019 USENIX Conference on Operational Machine Learning (OpML 19)*. Santa Clara, CA: USENIX Association, May 2019, pp. 59–61. ISBN: 978-1-939133-00-7. URL: <https://www.usenix.org/conference/opml19/presentation/bhattacharjee> (cit. on p. 29).

-
- [24] A Brito, D Ardagna, I Blanquer, A Evangelinou, E Barbierato, M Gribaudo, J Almeida, AP Couto, and T Braga. “D3.4 EUBra-BIGSEA QoS infrastructure services intermediate version”. In: *EUBRA BIGSEA Deliverable 3* (2017). URL: <https://www.eubra-bigsea.eu/deliverable-34-eubra-bigsea-qos-infrastructure-services-intermediate-version> (cit. on p. 85).
- [25] Erik Brynjolfsson and ANDREW McAfee. “The business of artificial intelligence”. In: *Harvard Business Review* (2017), pp. 1–20 (cit. on p. 8).
- [26] *Build the future of Open Infrastructure*. URL: <https://www.openstack.org/> (visited on 02/11/2020) (cit. on p. 6).
- [27] Deborah L Butler and Philip H Winne. “Feedback and self-regulated learning: A theoretical synthesis”. In: *Review of educational research* 65.3 (1995), pp. 245–281. URL: <https://www.jstor.org/stable/1170684?seq=1> (cit. on p. 53).
- [28] Miguel Caballer, Carlos de Alfonso, Fernando Alvarruiz, and Germán Moltó. “EC3: Elastic Cloud Computing Cluster”. In: *Journal of Computer and System Sciences* 79.8 (2013), pp. 1341–1351. ISSN: 00220000. DOI: 10.1016/j.jcss.2013.06.005. URL: <http://authors.elsevier.com/sd/article/S0022000013001141> (cit. on p. 73).
- [29] Miguel Caballer, Ignacio Blanquer, Germán Moltó, and Carlos de Alfonso. “Dynamic Management of Virtual Infrastructures”. In: *Journal of Grid Computing* 13.1 (2015), pp. 53–70. ISSN: 1570-7873. DOI: 10.1007/s10723-014-9296-5. URL: <http://link.springer.com/article/10.1007/s10723-014-9296-5> (cit. on p. 73).
- [30] Miguel Caballer, Germán Moltó, and Ignacio Blanquer. “Guest Editor’s Introduction: Special Issue on Cloud Computing Orchestration”. In:

Journal of Grid Computing 16 (Jan. 2018). DOI: 10.1007/s10723-018-9427-5 (cit. on p. 21).

- [31] Amanda Calatrava, Eloy Romero, Germán Moltó, Miguel Caballer, and Jose Miguel Alonso. “Self-managed cost-efficient virtual elastic clusters on hybrid Cloud infrastructures”. In: *Future Generation Computer Systems* 61 (2016), pp. 13–25. ISSN: 0167739X. DOI: 10.1016/j.future.2016.01.018. URL: <http://authors.elsevier.com/sd/article/S0167739X16300024> (cit. on p. 73).
- [32] Scott Callaghan, Philip Maechling, Patrick Small, Kevin Milner, Gideon Juve, Thomas H Jordan, Ewa Deelman, Gaurang Mehta, Karan Vahi, Dan Gunter, Keith Beattie, and Christopher Brooks. “Metrics for heterogeneous scientific workflows: A case study of an earthquake science application”. In: *The International Journal of High Performance Computing Applications* 25.3 (2011), pp. 274–285. DOI: 10.1177/1094342011414743. URL: <https://doi.org/10.1177/1094342011414743> (cit. on p. 7).
- [33] Eduardo Camacho-Ramos, Ana Jimenez-Pastor, Ignacio Blanquer, Fabio García-Castro, and Ángel Alberich-Bayarri. “Computer aided diagnosis for Rheumatic Heart Disease by AI applied to features extraction from echocardiography”. URL: <https://www.atmosphere-eubrazil.eu/computer-aided-diagnosis-rheumatic-heart-disease-ai-applied-features-extraction-echocardiography>. Poster presented at the European Society of Medical Imaging Informatics (EUSOMII) Annual Meeting, Valencia 18-19 October 2019 (cit. on p. 74).
- [34] R. S. Canon and A. Younge. “A Case for Portability and Reproducibility of HPC Containers”. In: *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*. 2019, pp. 49–54. URL: <https://doi.org/10.1109/ICNC47766.2019.00010>

-
- // conferences . computer . org / sc19w / 2019 / pdfs / CANOPIE - HPC2019 - 7nd7J7oXB1GtzeJIHi79mM / 3AyZkyZV1hldzPU6UEo655 / 30qM2Lkt9DqiE2sDu1jvaS.pdf (cit. on p. 23).
- [35] Enol Fernández del Castillo, Diego Scardaci, and Álvaro López García. “The EGI Federated Cloud e-Infrastructure”. In: *Procedia Computer Science* 68 (2015). 1st International Conference on Cloud Forward: From Distributed to Complete Computing, pp. 196–205. ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2015.09.235>. URL: <http://www.sciencedirect.com/science/article/pii/S187705091503080X> (cit. on p. 74).
- [36] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. “The rise of serverless computing”. In: *Communications of the ACM* 62.12 (2019), pp. 44–54. ISSN: 15577317. DOI: 10.1145/3368454 (cit. on pp. 4, 6, 39).
- [37] *Cloud Access to Mammograms Enables Earlier Breast Cancer Detection / Imaging Technology News*. URL: <https://www.itnonline.com/content/cloud-access-mammograms-enables-earlier-breast-cancer-detection> (visited on 06/16/2020) (cit. on p. 8).
- [38] *Cloud Computing for Education*. URL: https://aws.amazon.com/education/?nc1=h{_}ls (visited on 03/24/2020) (cit. on p. 21).
- [39] David Corral-Plaza, Juan Boubeta-Puig, and Manuel Resinas. “Un Recorrido por los Principales Proveedores de Servicios de Machine Learning y Predicción en la Nube”. In: *Actas de las XIV Jornadas de Ingeniería de Ciencia e Ingeniería de Servicios (JCIS 2018)*. 2018, pp. 1–10. URL: <http://hdl.handle.net/11705/JCIS/2018/013> (cit. on p. 29).

- [40] B. S. Dordević, S. P. Jovanović, and V. V. Timcenko. “Cloud Computing in Amazon and Microsoft Azure platforms: Performance and service comparison”. In: *2014 22nd Telecommunications Forum Telfor (TELFOR)*. 2014, pp. 931–934. URL: <https://ieeexplore.ieee.org/document/7034558> (cit. on p. 20).
- [41] Jose Duato, Antonio J. Pena, Federico Silla, Rafael Mayo, and Enrique S. Quintana-Orti. “rCUDA: Reducing the number of GPU-based accelerators in high performance clusters”. In: *2010 International Conference on High Performance Computing & Simulation*. IEEE, 2010, pp. 224–231. ISBN: 978-1-4244-6827-0. DOI: 10.1109/HPCS.2010.5547126. URL: <http://ieeexplore.ieee.org/document/5547126/> (cit. on p. 32).
- [42] Simon Eismann, Joel Scheuner, Erwin Van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina L Abad, Guayaquil Ecuador, and Alexandru Iosup. *A Review of Serverless Use Cases and their Characteristics SPEC RG Cloud Working Group*. Tech. rep. 2020. URL: <https://arxiv.org/pdf/2008.11110.pdf> (cit. on p. 27).
- [43] A. Eivy and J. Weinman. “Be Wary of the Economics of "Serverless" Cloud Computing”. In: *IEEE Cloud Computing 4.2* (2017), pp. 6–12. URL: <https://www.computer.org/csdl/magazine/cd/2017/02/mcd2017020006/13rRUx0ge83> (cit. on p. 127).
- [44] Tanya Elias. “Learning analytics: Definitions, Processes and Potential”. In: *Learning* (2011), pp. 1–22. URL: https://www.researchgate.net/publication/327220025_Learning_Analytics_Definitions_Processes_and_Potential (cit. on p. 25).
- [45] Alex Ellis. *OpenFaaS*. URL: <https://www.openfaas.com/> (cit. on pp. 6, 28).

- [46] Erwin van Eyk, Alexandru Iosup, Simon Seif, and Markus Thómmes. “The SPEC Cloud Group’s Research Vision on FaaS and Serverless Architectures”. In: *Proceedings of the 2nd International Workshop on Serverless Computing*. WoSC17. Las Vegas, Nevada: Association for Computing Machinery, 2017, pp. 1–4. ISBN: 9781450354349. DOI: 10.1145/3154847.3154848. URL: <https://doi.org/10.1145/3154847.3154848> (cit. on p. 4).
- [47] *Fetch Standard*. URL: <https://fetch.spec.whatwg.org/> (visited on 09/07/2020) (cit. on p. 110).
- [48] *Fission*. URL: <https://fission.io/> (cit. on p. 28).
- [49] Geoffrey Charles Fox, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. “Status of Serverless Computing and Function-as-a-Service(FaaS) in Industry and Research”. In: *ArXiv abs/1708.08028* (2017). URL: <https://arxiv.org/abs/1708.08028> (cit. on p. 3).
- [50] Sara de Freitas, David Gibson, Victor Alvarez, Leah Irving, Kam Star, Sven Charleer, and Katrien Verbert. “How to use gamified dashboards and learning analytics for providing immediate student feedback and performance tracking in higher education”. In: *Proceedings of the 26th International Conference on World Wide Web Companion*. International World Wide Web Conferences Steering Committee. 2017, pp. 429–434. URL: <https://dl.acm.org/doi/10.1145/3041021.3054175> (cit. on p. 53).
- [51] Sebastián Risco Gallardo. *Plataforma Serverless Híbrida de Procesado de Datos*. Tech. rep. 2019. URL: <https://riunet.upv.es/handle/10251/125892> (cit. on p. 66).

- [52] *Getting to the Heart of HPC and AI at the Edge in Healthcare*. URL: <https://www.nextplatform.com/2018/03/28/getting-to-the-heart-of-hpc-and-ai-at-the-edge-in-healthcare/> (visited on 06/16/2020) (cit. on p. 7).
- [53] V. Giménez-Alventosa, Germán Moltó, and Miguel Caballer. “A framework and a performance assessment for serverless MapReduce on AWS Lambda”. In: *Future Generation Computer Systems* (2019). ISSN: 0167739X. DOI: 10.1016/j.future.2019.02.057. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0167739X18325172> (cit. on p. 27).
- [54] Giulio Giunta, Raffaele Montella, Giuseppe Agrillo, and Giuseppe Coviello. “A GPGPU Transparent Virtualization Component for High Performance Computing Clouds”. In: *Euro-Par 2010 - Parallel Processing*. Ed. by Pasqua D’Ambra, Mario Guarracino, and Domenico Talia. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 379–391. ISBN: 978-3-642-15277-1. URL: https://link.springer.com/chapter/10.1007/978-3-642-15277-1_37 (cit. on p. 32).
- [55] Jorge Gomes, Emanuele Bagnaschi, Isabel Campos, Mario David, Luís Alves, João Martins, Pina João, Alvaro López-García, and Pablo Orviz. “Enabling rootless Linux Containers in multi-user environments: The udocker tool”. In: *Computer Physics Communications* 232 (2018), pp. 84–97. ISSN: 0010-4655. DOI: <https://doi.org/10.1016/j.cpc.2018.05.021>. URL: <http://www.sciencedirect.com/science/article/pii/S0010465518302042> (cit. on pp. 14, 23, 96).
- [56] Google. *Google Cloud Functions*. URL: <https://cloud.google.com/functions/> (cit. on p. 26).
- [57] Google. *Knative*. URL: <https://github.com/knative/> (cit. on pp. 6, 28).

- [58] Vishakha Gupta, Ada Gavrilovska, Karsten Schwan, Harshvardhan Kharche, Niraj Tolia, Vanish Talwar, and Parthasarathy Ranganathan. “GViM: GPU-accelerated Virtual Machines”. In: *Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing*. HPCVirt '09. Nuremburg, Germany: ACM, 2009, pp. 17–24. ISBN: 978-1-60558-465-2. DOI: 10.1145/1519138.1519141. URL: <http://doi.acm.org/10.1145/1519138.1519141> (cit. on p. 32).
- [59] Scott Hendrickson, Stephen Sturdevant, Tyler Harter, Venkateshwaran Venkataramani, Andrea C Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. “Serverless Computation with openLambda”. In: *Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing*. USENIX Association, 2016, pp. 33–39. ISBN: 00384941. DOI: 10.1111/j.1540-6237.2011.00758.x. URL: <https://dl.acm.org/citation.cfm?id=3027047> (cit. on p. 28).
- [60] *High Performance Computing and deep learning in medicine: Enhancing physicians, helping patients | Shaping Europe’s digital future*. URL: <https://ec.europa.eu/digital-single-market/en/news/high-performance-computing-and-deep-learning-medicine-enhancing-physicians-helping-patients> (visited on 06/16/2020) (cit. on p. 7).
- [61] G. Hu, Y. Zhang, and W. Chen. “Exploring the Performance of Singularity for High Performance Computing Scenarios”. In: *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. 2019, pp. 2587–2593. URL: <https://ieeexplore.ieee.org/abstract/document/8855563> (cit. on p. 24).
- [62] L. N. Hyseni and A. Ibrahim. “Comparison of the cloud computing platforms provided by Amazon and Google”. In: *2017 Computing*

- Conference*. 2017, pp. 236–243. URL: <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8252109> (cit. on p. 20).
- [63] IBM. *IBM Cloud Functions*. URL: <https://www.ibm.com/cloud/functions> (cit. on p. 26).
- [64] *IDC’s 2020 Enterprise Infrastructure Predictions | IDC Blog*. URL: https://blogs.idc.com/2019/12/02/top-10-worldwide-enterprise-infrastructure-2021-predictions/?utm{_}source=FutureScapes19{\&}utm{_}medium=webpage{\&}utm{_}campaign=12.02{_}EB{_}Blog (visited on 02/11/2020) (cit. on p. 2).
- [65] *Implementing a Serverless AWS IoT Backend with AWS Lambda and Amazon DynamoDB | AWS Compute Blog*. URL: <https://aws.amazon.com/es/blogs/compute/implementing-a-serverless-aws-iot-backend-with-aws-lambda-and-amazon-dynamodb/> (visited on 03/24/2020) (cit. on p. 21).
- [66] *Integration with the EGI Federated Cloud - oscar documentation*. URL: <https://o-scar.readthedocs.io/en/latest/egi-integration.html> (visited on 09/17/2020) (cit. on p. 120).
- [67] Sergio Iserte, Francisco J Clemente Castelló, Rafael Mayo, Francisco J Clemente-Castelló, Adrián Castelló, and Enrique S Quintana-Ortí. “Enabling GPU Virtualization in Cloud Environments Integrated Approaches to Linear Algebra Libraries View project COST Action IC1305 Network fo Sustainable Ultrascale Computing-NESUS View project Enabling GPU Virtualization in Cloud Environments”. In: (). DOI: 10 . 5220 / 0005780502490256. URL: <http://www.penguincomputing.com> (cit. on p. 32).
- [68] V. Ishakian, V. Muthusamy, and A. Slominski. “Serving Deep Learning Models in a Serverless Platform”. In: *2018 IEEE International*

-
- Conference on Cloud Engineering (IC2E)*. 2018, pp. 257–262. DOI: 10.1109/IC2E.2018.00052 (cit. on pp. 29, 120).
- [69] Douglas M Jacobsen and Richard Shane Canon. *Contain This, Unleashing Docker for HPC*. Tech. rep. 2015. URL: <https://www.nersc.gov/assets/Uploads/cug2015udi.pdf> (cit. on p. 23).
- [70] Devki Nandan Jha, Saurabh Garg, Prem Prakash Jayaraman, Rajkumar Buyya, Zheng Li, Graham Morgan, and Rajiv Ranjan. “A study on the evaluation of HPC microservices in containerized environment”. In: *Concurrency and Computation: Practice and Experience* n/a.n/a (). e5323 cpe.5323, e5323. DOI: 10.1002/cpe.5323. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5323> (cit. on p. 24).
- [71] A. Jimenez-Pastor, A. Alberich-Bayarri, F. Garcia-Castro, and L. Marti-Bonmati. “Automatic visceral fat characterisation on CT scans through deep learning and CNN for the assessment of metabolic syndrome.” In: *ECR 2019: Book of Abstracts. Insights into Imaging*. Vol. 10(S1). 2019. URL: http://webquibim.northeurope.cloudapp.azure.com/app/uploads/2019/12/2019_ECR_Automatic-visceral-fat-characterization-on-CT-scans-through-Deep-Learning-and-CNN-for-the-assessment-of-metabolic-syndrome_AJP.pdf (cit. on p. 74).
- [72] Ioana Jivet, Maren Scheffel, Marcus Specht, and Hendrik Drachsler. “License to Evaluate: Preparing Learning Analytics Dashboards for Educational Practice”. In: *Proceedings of the 8th International Conference on Learning Analytics and Knowledge*. LAK ’18. Sydney, New South Wales, Australia: ACM, 2018, pp. 31–40. ISBN: 978-1-4503-6400-3. DOI: 10.1145/3170358.3170421. URL: <http://doi.acm.org/10.1145/3170358.3170421> (cit. on p. 25).

- [73] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. “Occupy the cloud: distributed computing for the 99%”. In: *Proceedings of the 2017 Symposium on Cloud Computing - SoCC '17*. New York, New York, USA: ACM Press, 2017, pp. 445–451. ISBN: 9781450350280. DOI: 10.1145/3127479.3128601. arXiv: 1702.04024. URL: <http://dl.acm.org/citation.cfm?doid=3127479.3128601> (cit. on p. 27).
- [74] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Menezes Carreira, Karl Krauth, Neeraja Yadwadkar, Joseph Gonzalez, Raluca Ada Popa, David A. Patterson, Joao Carreira, and Joseph E. Gonzalez. *Cloud Programming Simplified: A Berkeley View on Serverless Computing*. Tech. rep. 2019. DOI: arXiv:1902.03383v1. arXiv: 1902.03383. URL: <http://arxiv.org/abs/1902.03383><http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.html> (cit. on p. 4).
- [75] Nima Kaviani, Dmitriy Kalinin, and Michael Maximilien. “Towards serverless as commodity: A case of Knative”. In: *WOSC 2019 - Proceedings of the 2019 5th International Workshop on Serverless Computing, Part of Middleware 2019*. Association for Computing Machinery, Inc, 2019, pp. 13–18. ISBN: 9781450370387. DOI: 10.1145/3366623.3368135 (cit. on p. 28).
- [76] Nawsher Khan, A. Noraziah, Tutut Herawan, and Mustafa Mat Deris. “Cloud Computing: Analysis of Various Services”. In: *Information Computing and Applications*. Ed. by Baoxiang Liu, Maode Ma, and Jincai Chang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 397–404. ISBN: 978-3-642-34062-8. URL: https://link.springer.com/chapter/10.1007/978-3-642-34062-8_52 (cit. on p. 20).

- [77] J. Kim, T. J. Jun, D. Kang, D. Kim, and D. Kim. “GPU Enabled Serverless Computing Framework”. In: *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. 2018, pp. 533–540. DOI: 10.1109/PDP2018.2018.00090 (cit. on pp. 33, 87).
- [78] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. Tech. rep. URL: <http://code.google.com/p/cuda-convnet/> (cit. on p. 120).
- [79] Kubernetes. *Kubernetes*. URL: <https://kubernetes.io/> (cit. on p. 6).
- [80] Rakesh Kumar and Shilpi Charu. *An Importance of Using Virtualization Technology in Cloud Computing*. Tech. rep. 2. 2015. URL: <http://gpcpublishing.org/index.php/gjct/article/view/21> (cit. on p. 2).
- [81] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. “Singularity: Scientific containers for mobility of compute”. In: *PLOS ONE* 12.5 (2017). Ed. by Attila Gursoy, e0177459. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0177459. URL: <https://dx.plos.org/10.1371/journal.pone.0177459> (cit. on p. 23).
- [82] Junfeng Li, Sameer G. Kulkarni, K. K. Ramakrishnan, and Dan Li. “Understanding open source serverless platforms: Design considerations and performance”. In: *WOSC 2019 - Proceedings of the 2019 5th International Workshop on Serverless Computing, Part of Middleware 2019*. Association for Computing Machinery, Inc, 2019, pp. 37–42. ISBN: 9781450370387. DOI: 10.1145/3366623.3368139. arXiv: 1911.07449 (cit. on p. 29).
- [83] W. Li, A. Kanso, and A. Gherbi. “Leveraging Linux Containers to Achieve High Availability for Cloud Services”. In: *2015 IEEE*

- International Conference on Cloud Engineering*. 2015, pp. 76–83. URL: <https://ieeexplore.ieee.org/document/7092902> (cit. on p. 3).
- [84] Miron Livny, Jim Basney, Rajesh Raman, and Todd Tannenbaum. “Mechanisms for high throughput computing”. In: *SPEEDUP journal* 11.1 (1997), pp. 36–40. URL: https://research.cs.wisc.edu/htcondor/doc/htc_mech.pdf (cit. on p. 9).
- [85] Álvaro López García. “DEEPaaS API: a REST API for Machine Learning and Deep Learning models”. In: *Journal of Open Source Software* 4.42 (Oct. 2019), p. 1517. ISSN: 2475-9066. DOI: 10.21105/joss.01517. URL: <http://dx.doi.org/10.21105/joss.01517> (cit. on p. 92).
- [86] Alvaro López García, Viet Tran, Andy S. Alic, Miguel Caballer, Isabel Campos Plasencia, Alessandro Costantini, Stefan Dlugolinsky, Doina Cristina Duma, Giacinto Donvito, Jorge Gomes, Ignacio Heredia Cacha, Jesus Marco De Lucas, Keiichi Ito, Valentin Y. Kozlov, Giang Nguyen, Pablo Orviz Fernandez, Zdenek Sustr, Pawel Wolniewicz, Marica Antonacci, Wolfgang Zu Castell, Mario David, Marcus Hardt, Lara Lloret Iglesias, Germán Molto, and Marcin Plociennik. “A Cloud-Based Framework for Machine Learning Workloads and Applications”. In: *IEEE Access* 8 (2020), pp. 18681–18692. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2964386. URL: <https://ieeexplore.ieee.org/document/8950411/> (cit. on p. 92).
- [87] *Magic Quadrant for Cloud Infrastructure as a Service, Worldwide*. URL: <https://www.gartner.com/en/documents/3947472> (visited on 07/27/2020) (cit. on p. 20).
- [88] Clémentine Maurice, Christoph Neumann, Olivier Heen, and Aurélien Francillon. “Confidentiality Issues on a GPU in a Virtualized Environment”. In: *Financial Cryptography and Data Security*. Ed. by

- Nicolas Christin and Reihaneh Safavi-Naini. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 119–135. ISBN: 978-3-662-45472-5. URL: https://link.springer.com/chapter/10.1007/978-3-662-45472-5_9 (cit. on p. 32).
- [89] Peter Mell and Timothy Grance. *The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology*. Tech. rep. 2011. DOI: 10.6028/NIST.SP.800-145. URL: <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf> (cit. on p. 1).
- [90] Lubos Mercl and Jakub Pavlik. “The Comparison of Container Orchestrators”. In: *Third International Congress on Information and Communication Technology*. Ed. by Xin-She Yang, Simon Sherratt, Nilanjan Dey, and Amit Joshi. Singapore: Springer Singapore, 2019, pp. 677–685. ISBN: 978-981-13-1165-9. URL: https://www.researchgate.net/publication/323417485_The_Comparison_of_Container_Orchestrators (cit. on p. 6).
- [91] Microsoft. *Microsoft Azure Functions*. URL: <https://azure.microsoft.com/en-us/services/functions/> (cit. on p. 26).
- [92] *MinIO | High Performance, Kubernetes Native Object Storage*. URL: <https://min.io/> (visited on 06/30/2020) (cit. on p. 66).
- [93] Germán Moltó and Miguel Caballer. “On Using the Cloud to Support Online Courses”. In: *2014 Frontiers in Education Conference (FIE)*. 2014, pp. 330–338. DOI: 10.1109/FIE.2014.7044041. URL: <http://ieeexplore.ieee.org/xpls/abs/all.jsp?arnumber=7044041> (cit. on pp. 50, 58).
- [94] Diana M. Naranjo, Ignacio Blanquer, Germán Moltó, Jorge Gomes, and Mario David. *IBERGRID 2019 - Delivering Innovative Computing and*

Data services to Researchers (23-26 de septiembre de 2019): Comparison of Container-based Virtualization Tools for HPC Platforms. · LIP Indico (Indico). URL: <https://indico.lip.pt/event/575/contributions/1856/> (visited on 04/08/2020) (cit. on p. 94).

- [95] Diana M. Naranjo, José R. Prieto, Germán Moltó, and Amanda Calatrava. “A Visual Dashboard to Track Learning Analytics for Educational Cloud Computing”. In: *Sensors* 19.13 (2019). ISSN: 1424-8220. DOI: 10.3390/s19132952. URL: <https://www.mdpi.com/1424-8220/19/13/2952> (cit. on pp. 41, 45, 49, 53–55, 57).
- [96] Diana M. Naranjo, Sebastián Risco, Carlos de Alfonso, Alfonso Pérez, Ignacio Blanquer, and Germán Moltó. “Accelerated serverless computing based on GPU virtualization”. In: *Journal of Parallel and Distributed Computing* 139 (2020), pp. 32–42. ISSN: 0743-7315. DOI: <https://doi.org/10.1016/j.jpdc.2020.01.004>. URL: <http://www.sciencedirect.com/science/article/pii/S0743731519303533> (cit. on pp. 70, 72, 75, 79–82, 84, 85).
- [97] Bruno R. Nascimento, Andrea Z. Beaton, Maria Carmo P. Nunes, Adriana C. Diamantino, Gabriel A.L. Carmo, Kaciane K.B. Oliveira, Cassio M. Oliveira, Zilda Maria A. Meira, Sandra Regina T. Castilho, Eduardo L.V. Lopes, Iara M. Castro, Vitória M.L.R. Rezende, Graziela Chequer, Taylor Landay, Allison Tompsett, Antonio Luiz P. Ribeiro, and Craig Sable. “Echocardiographic prevalence of rheumatic heart disease in Brazilian schoolchildren: Data from the PROVAR study”. In: *International Journal of Cardiology* 219 (2016), pp. 439–445. ISSN: 0167-5273. DOI: <https://doi.org/10.1016/j.ijcard.2016.06.088>. URL: <http://www.sciencedirect.com/science/article/pii/S0167527316310907> (cit. on p. 75).
- [98] *New-Provisioned Concurrency for Lambda Functions | AWS News Blog*. URL: <https://aws.amazon.com/blogs/aws/new-provisioned->

- concurrency-for-lambda-functions/ (visited on 09/07/2020) (cit. on p. 38).
- [99] NIST. “The NIST Definition of Cloud Computing”. In: *Observatorio Económico EEUU* BBVA Research (2016), pp. 1–8. DOI: 10.6028/NIST.SP.800-145. URL: https://www.bbvaresearch.com/wp-content/uploads/2016/05/160510{_}CloudBanking{_}esp.pdf (cit. on p. 18).
- [100] *Nuclio*. URL: <https://nuclio.io/> (cit. on p. 28).
- [101] NVIDIA. *What is a Virtual GPU*. Blog. 2018. URL: <https://blogs.nvidia.com/blog/2018/06/11/what-is-a-virtual-gpu/> (cit. on p. 31).
- [102] Justice Opara-Martins, Reza Sahandi, and Feng Tian. “Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective”. In: *Journal of Cloud Computing* 5.1 (2016), p. 4. ISSN: 2192113X. DOI: 10.1186/s13677-016-0054-z. URL: <http://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-016-0054-z> (cit. on p. 28).
- [103] *OpenNebula | The open source Cloud Management Platform developed for the Enterprise*. URL: <https://openebula.org/> (visited on 02/11/2020) (cit. on p. 6).
- [104] *Optimizing Lambda Performance for Your Serverless Applications | AWS Online Tech Talks*. URL: https://pages.awscloud.com/Optimizing-Lambda-Performance-for-Your-Serverless-Applications{_}2020{_}0316-SRV{_}OD.html (visited on 07/01/2020) (cit. on p. 38).
- [105] Oracle. *Fn Project*. URL: <https://fnproject.io/> (cit. on p. 28).

- [106] A. Palade, A. Kazmi, and S. Clarke. “An Evaluation of Open Source Serverless Computing Frameworks Support at the Edge”. In: *2019 IEEE World Congress on Services (SERVICES)*. Vol. 2642-939X. 2019, pp. 206–211. DOI: 10.1109/SERVICES.2019.00057 (cit. on p. 28).
- [107] Antonio José Peña Monferrer. “Virtualization of accelerators in high performance clusters”. In: (2013), p. 1. URL: <https://dialnet.unirioja.es/servlet/tesis?codigo=249008> (cit. on p. 32).
- [108] Alfonso Pérez, Germán Moltó, Miguel Caballer, and Amanda Calatrava. “Serverless computing for container-based architectures”. In: *Future Generation Computer Systems* 83 (2018), pp. 50–59. ISSN: 0167739X. DOI: 10.1016/j.future.2018.01.022. URL: <http://linkinghub.elsevier.com/retrieve/pii/S0167739X17316485> (cit. on pp. 27, 94, 96).
- [109] Alfonso Pérez, Sebastián Risco, Diana María Naranjo, Miguel Caballer, and Germán Moltó. “Serverless Computing for Event-Driven Data Processing Applications”. In: *2019 IEEE International Conference on Cloud Computing (CLOUD 2019)*. 2019. URL: <https://ieeexplore.ieee.org/document/8814513> (cit. on pp. 64–66, 73).
- [110] Ferran Pérez, Carlos Reaño, and Federico Silla. “Providing CUDA acceleration to KVM virtual machines in InfiniBand clusters with rCUDA”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9687. 2016, pp. 82–95. ISBN: 9783319395760. DOI: 10.1007/978-3-319-39577-7_7. URL: https://link.springer.com/chapter/10.1007/978-3-319-39577-7_{_}7 (cit. on p. 68).
- [111] Paul R. Pintrich. “Understanding self-regulated learning”. In: *New Directions for Teaching and Learning* 1995.63 (1995), pp. 3–12. DOI: 10.1002/tl.37219956304. eprint: <https://onlinelibrary.wiley.com/>

- doi/pdf/10.1002/tl.37219956304. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/tl.37219956304> (cit. on p. 53).
- [112] Pivotal. *Project riff*. URL: <https://projectriff.io/> (cit. on p. 28).
- [113] Reid Priedhorsky and Tim Randles. “Charliecloud: Unprivileged Containers for User-Defined Software Stacks in HPC”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '17. Denver, Colorado: Association for Computing Machinery, 2017. ISBN: 9781450351140. DOI: 10.1145/3126908.3126925. URL: <https://doi.org/10.1145/3126908.3126925> (cit. on p. 23).
- [114] Jose Ramón and Prieto Fontcuberta. *Portal Web de Analíticas de Uso para Cuentas Compartidas en AWS*. Tech. rep. 2018. URL: <https://riunet.upv.es/handle/10251/106685> (cit. on p. 41).
- [115] Mohan Rao Divate Kodandarama, Mohammed Danish Shaikh, and Shreeshrita Patnaik. *SerFer: Serverless Inference of Machine Learning Models*. Tech. rep. URL: <https://divatekodand.github.io/files/serfer.pdf> (cit. on pp. 30, 120).
- [116] Thomas Rausch, T U Wien, Waldemar Hummer, Vinod Muthusamy, Alexander Rashed, and Schahram Dustdar. *Towards a Serverless Platform for Edge AI*. Tech. rep. 2019. URL: <https://www.usenix.org/conference/hotedge19/presentation/rausch> (cit. on p. 30).
- [117] Carlos Reaño, Adrián Castelló, Sergio Iserte, Antonio Peña, Federico Silla, Rafael Mayo, Enrique S Quintana-Orti, and José Duato. “Virtualización remota de GPUs: Evaluación de soluciones disponibles para CUDA”. In: Sept. 2013, pp. 270–275. URL: https://www.researchgate.net/publication/256388411_Virtualizacion_

remota_de_GPUs_Evaluacion_de_soluciones_disponibles_para_CUDA (cit. on p. 32).

- [118] Carlos Reaño and Federico Silla. “A Performance Comparison of CUDA Remote GPU Virtualization Frameworks”. In: *2015 IEEE International Conference on Cluster Computing*. IEEE, 2015, pp. 488–489. ISBN: 978-1-4673-6598-7. DOI: 10.1109/CLUSTER.2015.76. URL: <http://ieeexplore.ieee.org/document/7307623/> (cit. on p. 67).
- [119] Carlos Reaño and Federico Silla. “On the support of inter-node P2P GPU memory copies in rCUDA”. In: *Journal of Parallel and Distributed Computing* 127 (2019), pp. 28–43. ISSN: 0743-7315. DOI: <https://doi.org/10.1016/j.jpdc.2018.12.011>. URL: <http://www.sciencedirect.com/science/article/pii/S0743731519300255> (cit. on p. 33).
- [120] Carlos Reaño, Federico Silla, Gilad Shainer, and Scot Schultz. “Local and Remote GPUs Perform Similar with EDR 100G InfiniBand”. In: *Proceedings of the Industrial Track of the 16th International Middleware Conference on ZZZ - Middleware Industry '15*. New York, New York, USA: ACM Press, 2015, pp. 1–7. ISBN: 9781450337274. DOI: 10.1145/2830013.2830015. URL: <http://dl.acm.org/citation.cfm?doid=2830013.2830015> (cit. on p. 68).
- [121] Ido Roll and Philip H. Winne. “Understanding, evaluating, and supporting self-regulated learning using learning analytics”. In: *Journal of Learning Analytics* 2.1 (2015), pp. 7–12. DOI: 10.18608/jla.2015.21.2. URL: <https://learning-analytics.info/index.php/JLA/article/view/4491> (cit. on p. 8).
- [122] Rami Rosen. *Resource management: Linux kernel Namespaces and cgroups*. Tech. rep. 2013. URL: <http://ramirose.wix.com/ramirosenwww.haifux.org> (cit. on p. 22).

- [123] O. Rudyy, M. Garcia-Gasulla, F. Mantovani, A. Santiago, R. Sirvent, and M. Vázquez. “Containers in HPC: A Scalability and Portability Study in Production Biological Simulations”. In: *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2019, pp. 567–577 (cit. on p. 24).
- [124] Amazon Web Services. *API Gateway*. URL: <https://aws.amazon.com/api-gateway> (cit. on p. 6).
- [125] *Servicios de cloud computing / Google Cloud*. URL: <https://cloud.google.com/> (visited on 02/11/2020) (cit. on p. 5).
- [126] *Servicios de informática en la nube / Microsoft Azure*. URL: <https://azure.microsoft.com/es-es/> (visited on 02/11/2020) (cit. on p. 5).
- [127] J. Shah and D. Dubaria. “Building Modern Clouds: Using Docker, Kubernetes Google Cloud Platform”. In: *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. 2019, pp. 0184–0189. URL: <https://ieeexplore.ieee.org/document/8666479> (cit. on p. 6).
- [128] Vaishaal Shankar, Karl Krauth, Qifan Pu, Eric Jonas, Shivaram Venkataraman, Ion Stoica, Benjamin Recht, and Jonathan Ragan-Kelley. “numpywren: serverless linear algebra”. In: (2018). arXiv: 1810.09679. URL: <https://arxiv.org/abs/1810.09679> (cit. on p. 27).
- [129] L. Shi, H. Chen, J. Sun, and K. Li. “vCUDA: GPU-Accelerated High-Performance Computing in Virtual Machines”. In: *IEEE Transactions on Computers* 61.6 (2012), pp. 804–816. ISSN: 0018-9340. DOI: 10.1109/TC.2011.112 (cit. on p. 32).

- [130] Mohammad Shorfuzzaman, M. Shamim Hossain, Amril Nazir, Ghulam Muhammad, and Atif Alamri. “Harnessing the power of big data analytics in the cloud to support learning analytics in mobile learning environment”. In: *Computers in Human Behavior* 92 (2019), pp. 578–588. ISSN: 0747-5632. DOI: <https://doi.org/10.1016/j.chb.2018.07.002>. URL: <http://www.sciencedirect.com/science/article/pii/S0747563218303248> (cit. on p. 25).
- [131] Josef Spillner, Cristian Mateos, and David A. Monge. “Faaster, better, cheaper: the prospect of serverless scientific computing and HPC”. In: *Communications in Computer and Information Science*. Vol. 796. Springer, Cham, 2018, pp. 154–168. ISBN: 9783319733524. DOI: 10.1007/978-3-319-73353-1_11. URL: http://link.springer.com/10.1007/978-3-319-73353-1_{_}11 (cit. on p. 26).
- [132] Srinivas Sridharan, Karthikeyan Vaidyanathan, Dhiraj Kalamkar, Dipankar Das, Mikhail E. Smorkalov, Mikhail Shiryaev, Dheevatsa Mudigere, Naveen Mellempudi, Sasikanth Avancha, Bharat Kaul, and Pradeep Dubey. “On Scale-out Deep Learning Training for Cloud and HPC”. In: (2018). arXiv: 1801.08030. URL: <http://arxiv.org/abs/1801.08030> (cit. on p. 9).
- [133] Yusuke Suzuki, Shinpei Kato, Hiroshi Yamada, and Kenji Kono. “GPUvm: Why Not Virtualizing GPUs at the Hypervisor?” In: *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, June 2014, pp. 109–120. ISBN: 978-1-931971-10-2. URL: <https://www.usenix.org/conference/atc14/technical-sessions/presentation/suzuki> (cit. on p. 31).
- [134] *Swarm mode overview | Docker Documentation*. URL: <https://docs.docker.com/engine/swarm/> (visited on 02/12/2020) (cit. on p. 6).

- [135] Yassine Tabaa and Abdellatif Medouri. “LASyM: a learning analytics system for MOOCs”. In: *International Journal of Advanced Computer Science and Applications* 4 (June 2013). DOI: 10.14569/IJACSA.2013.040516 (cit. on p. 25).
- [136] H. Tan, Y. Tan, X. He, K. Li, and K. Li. “A Virtual Multi-Channel GPU Fair Scheduling Method for Virtual Machines”. In: *IEEE Transactions on Parallel and Distributed Systems* 30.2 (2019), pp. 257–270. ISSN: 1045-9219. DOI: 10.1109/TPDS.2018.2865341 (cit. on p. 31).
- [137] Manuel Ujaldón. “CUDA achievements and GPU challenges ahead”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9756. 2016, pp. 207–217. ISBN: 9783319417776. DOI: 10.1007/978-3-319-41778-3_20. URL: https://link.springer.com/chapter/10.1007/978-3-319-41778-3_{_}20 (cit. on p. 32).
- [138] Katrien Verbert, Sten Govaerts, Erik Duval, Jose Luis Santos, Frans Van Assche, Gonzalo Parra, and Joris Klerkx. “Learning dashboards: an overview and future research opportunities”. In: *Personal and Ubiquitous Computing* 18.6 (2014), pp. 1499–1514. ISSN: 1617-4917. DOI: 10.1007/s00779-013-0751-2. URL: <https://doi.org/10.1007/s00779-013-0751-2> (cit. on p. 25).
- [139] Camilo Vieira, Paul Parsons, and Vetricia Byrd. “Visual learning analytics of educational data: A systematic literature review and research agenda”. In: *Computers & Education* 122 (2018), pp. 119–135. ISSN: 0360-1315. DOI: 10.1016/j.compedu.2018.03.018. URL: <http://www.sciencedirect.com/science/article/pii/S0360131518300770> (cit. on p. 25).
- [140] J. P. Walters, A. J. Younge, D. I. Kang, K. T. Yao, M. Kang, S. P. Crago, and G. C. Fox. “GPU Passthrough Performance: A

- Comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL Applications”. In: *2014 IEEE 7th International Conference on Cloud Computing*. 2014, pp. 636–643. DOI: 10.1109/CLOUD.2014.90 (cit. on p. 31).
- [141] *What is a Container? / App Containerization / Docker*. URL: <https://www.docker.com/resources/what-container> (visited on 06/14/2020) (cit. on pp. 2, 22).
- [142] *What Is AWS Batch? - AWS Batch*. URL: <https://docs.aws.amazon.com/batch/latest/userguide/what-is-batch.html> (visited on 04/21/2020) (cit. on p. 97).
- [143] Billy Tak ming Wong and Kam Cheong Li. *A review of learning analytics intervention in higher education (2011-2018)*. 2020. DOI: 10.1007/s40692-019-00143-7 (cit. on p. 25).
- [144] Andy Wu. *Taking the Cloud-Native Approach with Microservices*. Tech. rep. 2017. URL: <https://cloud.google.com/files/Cloud-native-approach-with-microservices.pdf> (cit. on p. 3).
- [145] Yuping Xing and Yongzhao Zhan. “Virtualization and Cloud Computing”. In: *Future Wireless Networks and Information Systems*. Ed. by Ying Zhang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 305–312. ISBN: 978-3-642-27323-0. URL: <https://www.springer.com/gp/book/9783642273223> (cit. on p. 21).
- [146] Mengting Yan, Paul Castro, Perry Cheng, and Vatche Ishakian. “Building a Chatbot with Serverless Computing”. In: Dec. 2016, pp. 1–4. DOI: 10.1145/3007203.3007217 (cit. on p. 30).
- [147] Chao Tung Yang, Jung Chun Liu, Hsien Yi Wang, and Ching Hsien Hsu. “Implementation of GPU virtualization using PCI pass-through

- mechanism”. In: *Journal of Supercomputing* 68.1 (2014), pp. 183–213. ISSN: 15730484. DOI: 10.1007/s11227-013-1034-4 (cit. on p. 32).
- [148] Hangchen Yu and Christopher J. Rossbach. “Full Virtualization for GPUs Reconsidered”. In: 2017. URL: <https://www.cs.utexas.edu/~hyu/publication/wddd17-gpvm.pdf> (cit. on p. 31).

Glossary

AI Artificial Intelligence. 7, 9

Amazon S3 Amazon Simple Storage Service. 6, 27, 42, 43, 48, 65, 111

API Application Programming Interface. 4, 6, 31, 32, 43–48, 65–67, 92, 93, 110

AWS Amazon web Services. 5, 6, 13, 20, 26, 27, 30, 33, 41–44, 46–51, 56–60, 91, 94, 96–98, 108–110, 117, 119, 120, 124, 126, 128

BaaS Backend as a Service. 4, 26

CLI Command Line Interface. 93, 96

CLUES CLuster Elasticity System. 73, 117

CMP Cloud Management Platforms. 5, 7

COPs Container Orchestration Platforms. 6

CPU Central Processing Unit. 21, 31, 33, 37, 48, 72, 74, 77, 78, 83, 86, 88, 98, 99, 104, 125

- CUDA** Compute Unified Device Architecture. 32, 33, 67
- EC2** Elastic Cloud Computing. 51, 97, 99, 104, 109, 119, 120, 124
- EC3** Elastic Cloud Computing Cluster. 73
- EDA** Event Driven Architecture. 28
- FaaS** Functions as a Service. 3, 4, 6, 9, 26, 28, 30, 36, 38, 63, 65, 78, 93, 94, 119, 136
- FPGA** Field-programmable gate array. 13, 31
- GB** Gigabyte. 13, 48
- GbE** Gigabit Ethernet. 13, 82
- GCP** Google Cloud Platform. 5, 6, 20
- GPGPU** General Purpose Computing on GPUs. 31, 33, 67, 86
- GPU** Graphics Processing Unit. xvi, 10, 11, 23, 30–33, 63, 64, 66–72, 74–88, 94, 118, 119, 125, 128
- GUI** Graphical User Interface. 65
- HPC** High Performance Computing. 9, 10, 22–24, 67, 94, 95
- HTC** High Throughput Computing. 9
- HTTP** Hypertext Transfer Protocol. 6, 36
- IaaS** Infrastructure as a Service. 1, 2, 19, 20
- IaC** Infrastructure as Code. 104
- IAM** Identity and Access Management. 40, 42, 98

IM Infrastructure Manager. 73

IoT Internet of Things. 21, 29

IP Internet Protocol. 68

IT Information Technology. 2, 4, 20, 34

JPG Joint Photographic Experts Group. 93

JSON JavaScript Object Notation. 44, 93, 105, 106, 112

KVM Kernel-based Virtual Machine. 13, 32

LIP Laboratory of Instrumentation and Particle Physics. 14

LXC Linux Containers. 23

MAU Monthly Active Users. 48

MB Megabyte. 47, 109

ML Machine Learning. 7–9, 11, 29, 76, 91–93, 105, 107, 121

MOOC Massive Open Online Courses. 25

MPI Message Passing Interface. 9, 23, 94

NIST National Institute of Standards and Technology. 18

OSCAR Open Source Serverless Computing for Data-Processing Applications. 64–66, 69–74, 77–79, 83, 84, 86, 124, 125, 128

PaaS Platform as a Service. 1, 2, 19, 20

PCI Peripheral Component Interconnect. 31, 32

PROVAR Programa de Rastreamento da Valvopatia Reumática - Rheumatic Heart Disease Screening Program. 75

RADL Resource Application Description Language. 73

RAM Random Access Memory. 13, 23, 47

RDMA Remote Direct Memory Access. 68

REST Representational State Transfer. 43, 44, 66, 92, 93

RoCE RDMA over converged Ethernet. 68

SaaS Software as a Service. 1, 2, 19

SAN Storage Array Network. 13

SCAR Serverless Container-aware ARchitectures. 27, 94, 96, 98, 107, 109, 128

TCO Total Cost of Ownership. 118

TCP Transmission Control Protocol. 68

VM Virtual Machine. 2, 7, 22, 31, 32, 39

VMCG Virtual Multi-Channel GPU. 31

XaaS Anything as a Service. 17

YAML YAML Ain't Markup Language. 96, 107