UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN
INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

Design and development of an OSLC adapter for Google Cloud Services

ALEJANDRO JESÚS VARGAS PÉREZ

2022

TRABAJO DE FIN DE MASTER

Título:	Diseño y Desarrollo de un Adaptador OSLC para Servicios de Google Cloud
Título (inglés):	Design and development of an OSLC adapter for Google Cloud Services
Autor:	Alejandro Jesús Vargas Pérez
Tutor:	Álvaro Carrera Barroso
Departamento:	Departamento de Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	—
Vocal:	—
Secretario:	—
Suplente:	—

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO DE FIN DE MÁSTER

Design and development of an OSLC adapter for Google Cloud Services

JUNIO 2022

Resumen

Hoy en día, los servicios en la nube son los habilitadores clave para el desarrollo de proyectos de Big Data y ofrecen múltiples herramientas para la gestión de un gran volumen de datos. Sin embargo, la existencia de múltiples proveedores de nube implica una falta de flexibilidad e interoperabilidad entre ellos, ya que siempre tienden a obligar a los desarrolladores a utilizar exclusivamente sus servicios y suelen ser difíciles de integrar con otros proveedores o herramientas de terceros. Esto conduce a lo que se conoce como *vendor lock-in*.

Aquí es donde entra en acción Open Services for Lifecycle Collaboration (OSLC), ya que proporciona un conjunto de especificaciones para la integración de herramientas utilizando un modelo flexible basado en *Linked Data*. El uso de una especificación estándar pretende simplificar el proceso de integración y facilitar la interacción entre diferentes herramientas.

En este proyecto se ha diseñado un modelo semántico aplicado a Google Cloud, que define los recursos estándar de la nube, y sienta las bases para poder integrar cada una de las múltiples herramientas y servicios que ofrecen los proveedores de la nube. Mediante la investigación de los servicios de Google Cloud Platform, se han estandarizado sus conceptos y recursos para crear un modelo semántico lo más genérico posible.

Además, se ha desarrollado una implementación de un adaptador OSLC que permite la integración con Google Cloud, siguiendo los estándares OSLC. Esto se ha hecho mapeando los conceptos semánticos a los recursos reales de la plataforma, así como utilizando las APIs de Google para interactuar con ellos. Además, se ha desarrollado un flujo de trabajo con los servicios propios de Google para mantener la sincronización en tiempo real entre los recursos reales en vivo y los recursos semánticos almacenados en el adaptador.

En conclusión, este proyecto ha supuesto un paso adelante en la estandarización de las herramientas y servicios en la nube tomando como referencia Google Cloud, proporcionando un lenguaje común para definir sus recursos y, por tanto, facilitando la integración de las modernas herramientas DevOps en los servicios en la nube.

Palabras clave: Big Data, Servicios en la Nube, Interoperabilidad, OSLC, Datos Enlazados, Estandarización, Modelo Semántico, Google Cloud Platform, API, DevOps

Abstract

Nowadays, cloud services are the key enablers for developing Big Data projects and offer multiple tools for the management of a large volume of data. However, the existence of multiple cloud providers implies a lack of flexibility and interoperability between them, since they always tend to force developers to use exclusively its services and are usually difficult to integrate with other vendors or third-party tools. This leads to a vendor lock-in.

Here, Open Services for Lifecycle Collaboration (OSLC) takes action as it provides a set of specifications for integrating compliant tools using a flexible model based on Linked Data. The use of a standard specification aims to simplify the integration process and facilitates interaction between different tools.

In this project, a semantic model applied to Google Cloud has been designed to define standard cloud resources and lay the foundation for integrating each of the multiple tools and services that cloud providers provide. By researching Google Cloud Platform services, its concepts and resources have been standardized to create the most generic semantic model possible.

Moreover, an implementation of an OSLC adapter that allows integration with Google Cloud, following OSLC standards, has been developed. This has been done by mapping semantic concepts to the actual resources of the platform, as well as using Google APIs to interact with them. In addition, a workflow has been developed with Google's own services to maintain synchronization in real time between the actual live resources and semantic resources stored in the adapter.

In conclusion, this project has led to a step forward in standardizing cloud tools and services using Google Cloud as a reference, providing a common language to define cloud resources and, therefore, facilitating the integration of modern DevOps tools into cloud services.

Keywords: Big Data, Cloud Services, Interoperability, OSLC, Linked Data, Standardization, Semantic Model, Google Cloud Platform, API, DevOps

Agradecimientos

Quiero dedicar este trabajo a todas aquellas personas que me han apoyado y acompañado durante todo el desarrollo de mi máster y mi trabajo final de carrera, a mis padres, mi novia, mi familia y amigos cercanos. También quiero agradecer la ayuda recibida por parte de mis compañeros del departamento GSI, y en especial, el apoyo de mi tutor Álvaro, por aportar grandes ideas y ofrecer soluciones.

Gracias a todos.

Contents

List of Figures

Listings

CHAPTER 1

Introduction

This project has been done as part of SmartDevOps UPM project in the Intelligent Systems Group (GSI).

This chapter introduces the context of the project, including a brief overview of all the different parts that will be discussed in the project. It will also breakdown a series of objectives to be met during the realization of the project. In addition, it will introduce the structure of the document with an overview of each chapter.

1.1 Context

The increased interest of companies to develop Big Data projects to improve their businesses is making open source DevOps tools more valuable and demanded. Therefore, this project has been developed to enable the integration of these tools with semantic technologies and Big Data projects mainly hosted on cloud services. Moreover, standardization and interoperability between tools is essential for the success of these projects and facilitate the integration of a wide variety of vendors and applications.

Standardization is achieved by using the open-source data principles of Linked Data to define resources and its relationships. Following these principles, an open source data model has been used, that is, the Open Services for Lifecycle Collaboration (OSLC) model, which is based on a set of specifications for simplifying tool integration across the software delivery lifecycle. Moreover, it merges these concepts with REST APIs to enable the Web of data by providing open standards to achieve a uniform interface and facilitates the connectivity of data between different sources.

Taking these principles into account, a cloud semantic model based on Google Cloud Platform (GCP) has been defined that extends the OSLC core specification and establishes the basis to enable the integration of multiple cloud providers and different tools. This could allow the operation and monitoring of cloud resources in an open and standard way, so that it is possible to manage them as global resources despite the cloud provider.

In particular, the desired DevOps features are mainly three: scalability, achieved by standardization and interoperability, adaptability to rapid changes, accomplished by a fast service integration, and finally, automation, by providing continuous delivery (CD). The latter can also benefit from this project as it is possible to detect any event on any cloud resource and automate actions in response to that event, so that it can be possible to deploy cloud infrastructure without any interaction with the cloud provider.

Therefore, this project will focus on the development of an OSLC-based adapter that will act as a connector between the semantic world based on the Resource Description Framework (RDF) triples on a graph and real resources that are part of an ecosystem of services offered by cloud providers. From a user's point of view, it will be totally transparent which cloud provider is dealing with, since it will only deal with generic cloud resources identified by semantic concepts.

1.2 Motivation

One of the most important weaknesses on Big Data projects based on cloud services is vendor lock-in, as service providers often force you to adopt all of it services, so it is really difficult to integrate your application or services with external ones. Consequently, there is a lack of interoperability with your resources, and, therefore, big data developers are limited by cloud service providers.

Moreover, the obligation to work with only a service provider limits the possibility of migration of your own service and also, it makes so expensive the operation. Additionally, developers will be forced to learn new tools, so migration will again be challenging, expensive, and slow.

As a consequence, service providers do not offer standardized APIs, and therefore, it is hard to cover every possible tool for each use case. This leads to a lack of flexibility for developers, so they will not be able to cover every desired functionality of their applications and will limit the services offered by their companies.

The solution that this project proposes resides in a standardized multi-cloud environment where organizations are able to deploy whatever type of infrastructure, service, or software without considering the obligation to use a specific cloud provider service and focus solely on generic and standard resources semantically defined. In that way, it will provide more flexibility to organizations when choosing which cloud service to use and reduce to almost none the dependence on a single cloud provider.

As a result, organizations can benefit of the usage of provider-specific services to best fit specific application and infrastructure requirements to their own business needs, enhanced scalability with multiple cloud providers, containers, and microservices, as usually some of these services are only available from a specific cloud provider, whereas now they will be able to integrate all of them, reduced latency, since organizations can choose local public cloud vendors based on each facility location, and finally, cost efficiency and security as they will be able to spread risk of failure across several vendors, and they can choose the provider which offers the best price for a given service.

1.3 Project goals

This project aims to resolve the previous deficiencies of actual cloud service providers mentioned before, so development must be done following OSLC standard and a generic definition of Google Cloud Service must be done in order to open the possibility of integration of any cloud provider. In this way, a vast research and study of the OSLC standard and all of its semantic vocabularies is needed to understand every semantic aspect of the standard so an optimum mapping with cloud resources could be done.

As a conclusion of the study, a standard semantic model of generic cloud providers and their resources is needed, always in accordance with the OSLC standard. In particular, the model will be applied to Google Cloud Services as a cloud provider, but it will be prepared so that any other cloud provider could be integrated on it.

In addition, an adapter will be designed and developed in order to connect a semantic definition of Google Cloud resources with actual resources of GCP. Thus, the adapter will be in charge of the interaction between resources via Google API.

Finally, as a result, a demonstration of the adapter capabilities will be done simulating a realistic scenario of useful resources and interacting with them either with the adapter or via Google, so the synchronization of both services is verified.

In summary, the main goals of this project are as follows.

- Research and study of semantic concepts and OSLC standard.
- Definition of a cloud semantic model following OSLC core concepts.
- Design and development of an adapter to interact with Google APIs.
- Demonstration of the adapter capabilities by interacting with the resources.

1.4 Structure of this document

The remaining of this document is structured as follows:

Chapter 2: Enabling Technologies. This chapter will explain the context on which this project is based, with an introduction and explanation of all the technologies, tools, and resources that will be used in the development of the project.

Chapter 3: Architecture. This chapter will contain the semantic model that will extend the OSLC model, as well as the real architecture of the developed service for this project and the process that has been carried out to develop the components of the system.

Chapter 4: Implementation. Here, the actual development of the different components of the system will be explained in detail.

Chapter 5: Use Cases. In this chapter, different use cases will be presented and explained to test adapter capabilities.

Chapter 6: Conclusion. This chapter will resume the results and conclusions of the project along with a discussion of future work and how to integrate new services with the actual system developed on this project.

Appendixes. Here, the link to access the entire developed code, the impact of the project in terms of social, economic, environmental and ethical impact, and the project budget can be found.

CHAPTER 2

Enabling Technologies

This chapter will explain the context on which this project is based, with an introduction and explanation of all the technologies, tools and resources that will be used in the development of the project.

2.1 Cloud computing

For the last few years, there has been a global trend of enterprises towards digital transformation of their businesses, mainly motivated by scalability and expansion. Today, the use of cloud services has become an industry standard, and almost all companies develop their businesses with some form of cloud computing either in a hybrid environment or in a full cloud environment.

On the one hand, there have been numerous companies that have adopted a hybrid cloud environment as they are still on a first stage of the cloud journey. They have just detected on premises deficiencies and have started to integrate some of their own services with cloud resources and services in order to provide more flexibility to their actual business plan.

On the other hand, newer companies and startups have fully adopted cloud services and are cloud native as they build and run applications that exploit the advantages of the cloud computing delivery model. They often respond sooner to customer demands and bring more adaptability to the actual market needs as they build and operate their applications using a cloud native architecture. The adoption of this model has a strong relationship with DevOps, continuous delivery, microservices, and containers concepts.

Either way, there are clear benefits of cloud computing for companies, such as cost efficiency, as you pay for the computing resources that you need and use; agility and faster time to market, as enterprises can develop new applications and get them into production without worrying about the underlying infrastructure; scalability and flexibility, as enterprises and their users can access cloud services from anywhere with an internet connection, scaling services up or down in response to business growth or surges in traffic; reliability and business continuity, due to redundancy and distributed resources, which makes a perfect plan for disaster recovery; and finally, high security, as they assure low risks due to multiple security mechanisms that cloud providers put into place [18].

Regarding the degree of adoption of cloud services, there have been established different types of cloud computing that will be detailed below.

2.1.1 Types of Cloud Computing

Public cloud are the most common type of cloud computing deployment, where resources such as servers or storage are owned and operated by a third party service provider [3]. Customers do not need to purchase any hardware, software, or supporting infrastructure of their own, and the customer rents a portion of it for a subscription-based or usage-based fee. The principal advantages of public clouds are lower costs, as you do not need to purchase any infrastructure and you pay only for what you use, no maintenance as it is provided by the

service provided, almost unlimited scalability, as resources are always available on-demand to meet your business needs, and high reliability ensuring no point of failure.

Private cloud in contrast with public clouds, its resources are used exclusively by one business or organization. Regarding the physical location of the infrastructure, now hardware is located on on-site data centers or it can be hosted by a third-party service provider and that means, services and infrastructure are always maintained on a private network so the organization has the full access control, security and resource customization as it is on an on-premise infrastructure [8]. The main benefits over a public cloud resides in security and full control in order to meet regulatory compliance requirements with sensitive data.

Hybrid cloud integrates either public cloud services, private cloud services or on-premise infrastructure and provides orchestration, management and application portability across all three [32], resulting on a single, unified and flexible distributed computing environment where organizations can run and scale its traditional or cloud native workloads on an optimum environment that fits perfectly with its necessities. This solution provides organizations with greater control over their private data, since they can store it on a private cloud or a local data-center while simultaneously benefiting from the robust computational resources of a managed public cloud.

Multicloud is an alternative type of cloud computing that encompasses all the types mentioned above and is one of the core principles and what motivates this project. Multicloud means the use of multiple cloud services from more than a cloud vendor or even a mixture of public cloud, hybrid cloud, and on-premises, regardless of the service provider. This strategy not only provides more flexibility for which cloud services an enterprise chooses to use, it also reduces dependence on a single cloud hosting provider relying exclusively on standard resources that can be deployed wherever they deserve.

2.1.2 Cloud services

There are multiple types of cloud service that are accessible across the different cloud service providers, depending on how much control the infrastructure and data enterprises need for their applications. Most businesses using cloud-based platforms use a combination of the different services offered by cloud service providers. These services can be illustrated in the following in Figure 2.1.

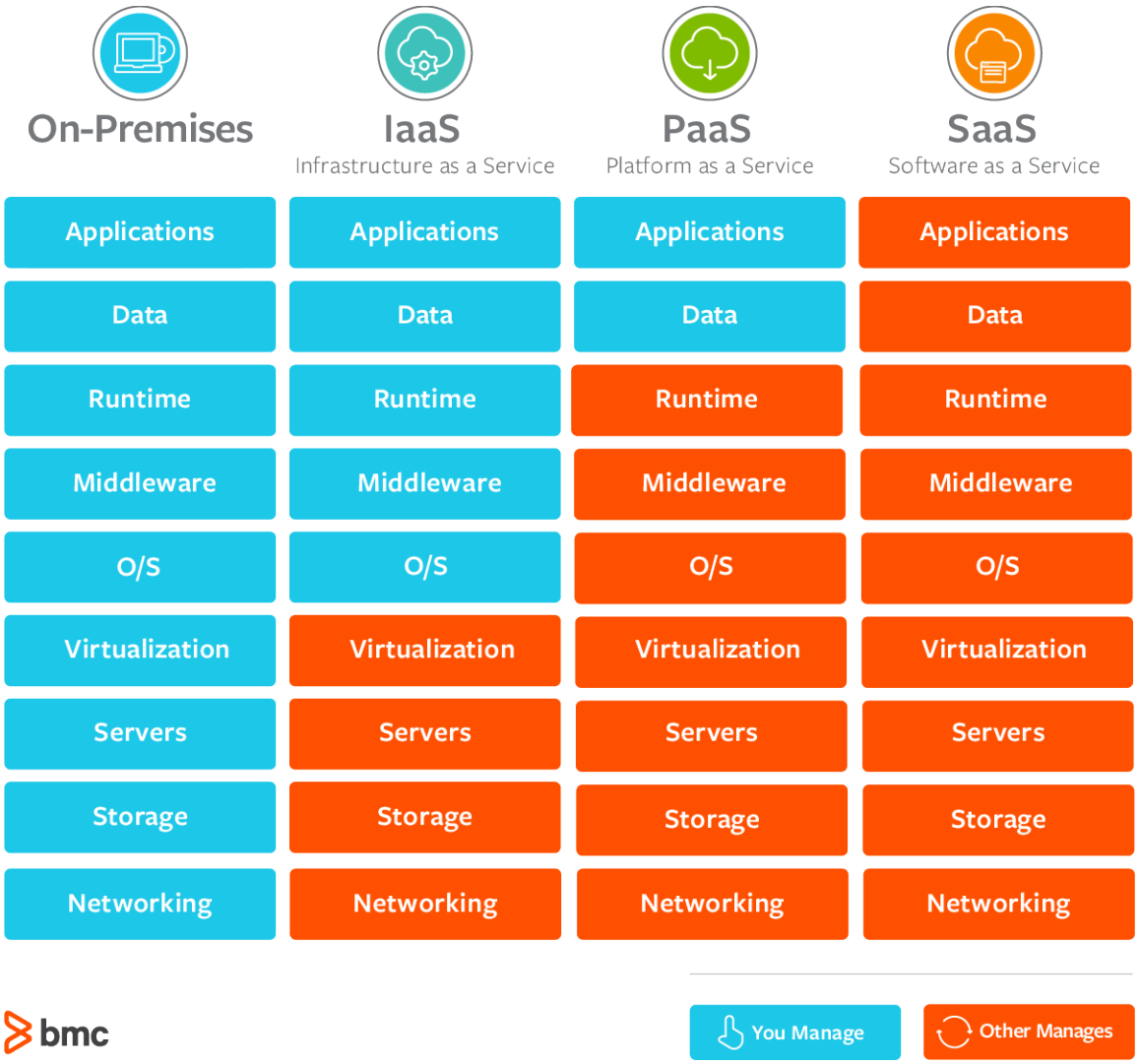


Figure 2.1: Cloud services [26]

IaaS

Infrastructure as a Service (IaaS), as shown in Figure 2.1 [26], is a set of cloud services in which computing resources owned by a service provider are complemented by storage and networking capabilities and are offered to customers on demand. Customers can benefit from the use of all the necessary resources for computing, networking, storage, and other services, without the need of building a data center on premises. In addition, resources are scalable and flexible since it is possible to access them through self-service interfaces, including API and graphical user interfaces (GUI). Some examples of IaaS are DigitalOcean, Amazon Web Services (AWS), Microsoft Azure, or Google Compute Engine (GCE).

PaaS

Platform as a Service (PaaS) provides a complete cloud-based platform for developing, running, and managing applications without the cost, complexity, and inflexibility of building and maintaining that platform on premises [26]. All servers, storage, and networking can be managed by the enterprise or a third-party provider, while developers can maintain management of the applications, and consequently, they can focus only on the actual development of applications and services that add value for their businesses. This model has advantages such as reduced programming time, easier multi-platform development, or efficient management of applications life cycle.

Serverless Computing similar to PaaS, has the same advantages as conventional PaaS, but differs mainly in two important ways: It offloads all responsibility for infrastructure management tasks (scaling, scheduling, patching, provisioning) to the cloud provider, allowing developers to focus specifically on code; and serverless runs code only on demand, when requested by the application, which means that the customer only has to pay for computing power only when the code needs to run and idle computing capacity is never billed.

Software as a Service (SaaS) is the most common form of cloud computing in which a complete cloud application and all its underlying infrastructure and platform are delivered to customers through an Internet browser. Consequently, customers can avoid the responsibility of buying or maintaining infrastructure, platforms, or on-premises software, and prefer a simple cost management through operational expenses (OPEX) with subscription plans, rather than capital expense investments (CAPEX). Moreover, this solution provides an easy way for organizations to start new businesses or improve old ones, as providers can roll out new features that customers can benefit from, mainly because they offer plug-and-play applications and manage everything behind the app.

2.1.3 Cloud DevOps

Development and Operations (DevOps) can be defined as a collaborative and multidisciplinary effort within an organization to automate the continuous delivery of new software versions. while guaranteeing their correctness and reliability [22]. DevOps team consists of a fusion between developers, administrators, QA and security experts, and potentially other specialists as networking experts and allows a shorter cycle software development in an agile way, using a common language, the Cloud.

Regarding the multiple benefits of combining Cloud and DevOps, they provide a quicker time to market thanks to the faster availability of simplified development processes and environments, as well as reduce complexity and system maintenance by automation and

infrastructure-as-code approaches, resulting in an increase on security by reducing errors by automation of repeatable processes.

Moreover, cloud-based operations eliminate downtime, as in the process of applying automation, developers create stateless applications, increasing reliability, availability, failover capability, and finally customer satisfaction [5].

The combination of advantages of an efficient Cloud DevOps team results in an increased scalability, one of the main challenges of Big Data applications. The use of infrastructure-as-code and microservice-oriented and stateless designs, enables fast scaling and auto-scaling and allows organizations to manage an optimum capacity for meeting business demand and consequently, reducing overall infrastructure costs and increasing the overall scope of solutions.

2.1.4 Google Cloud Platform (GCP)

As a Cloud Service Provider, Google Cloud is a public cloud that offers a wide variety of services, from Iaas, Paas to SaaS solutions that covers almost all of possible use cases for any type of business. Google offers from computing services to storage and database, networking, big data tools, machine learning, Internet of things, identity and security, data transfer or cloud AI services.

In this project, as a semantic model of GCP is defined, only some of its services are categorized, and it will be explained below.

Computation Service

Google Cloud offers users the ability to manage and store computing with a wide variety of options, from working in a serverless environment, using a managed application platform, building cloud-based infrastructure to facilitate maximum control and flexibility, or leveraging container technologies to achieve maximum flexibility[17]. Hereunder some of computing services are explained going from less level of abstraction to more level:

- **Google Compute Engine (GCE):** offers an IaaS service that provides secure and customizable virtual machines hosted on Google's infrastructure.
- **Google Kubernetes Engine (GKE):** is an easy-to-use cloud-based Kubernetes service used for running containerized applications. It is a management and orchestration system for Docker containers and container clusters that run within Google's public cloud services.
- **Google App Engine (GAE):** is a PaaS that offers Google Cloud services for building scalable web applications and IoT backends that scale automatically based on the

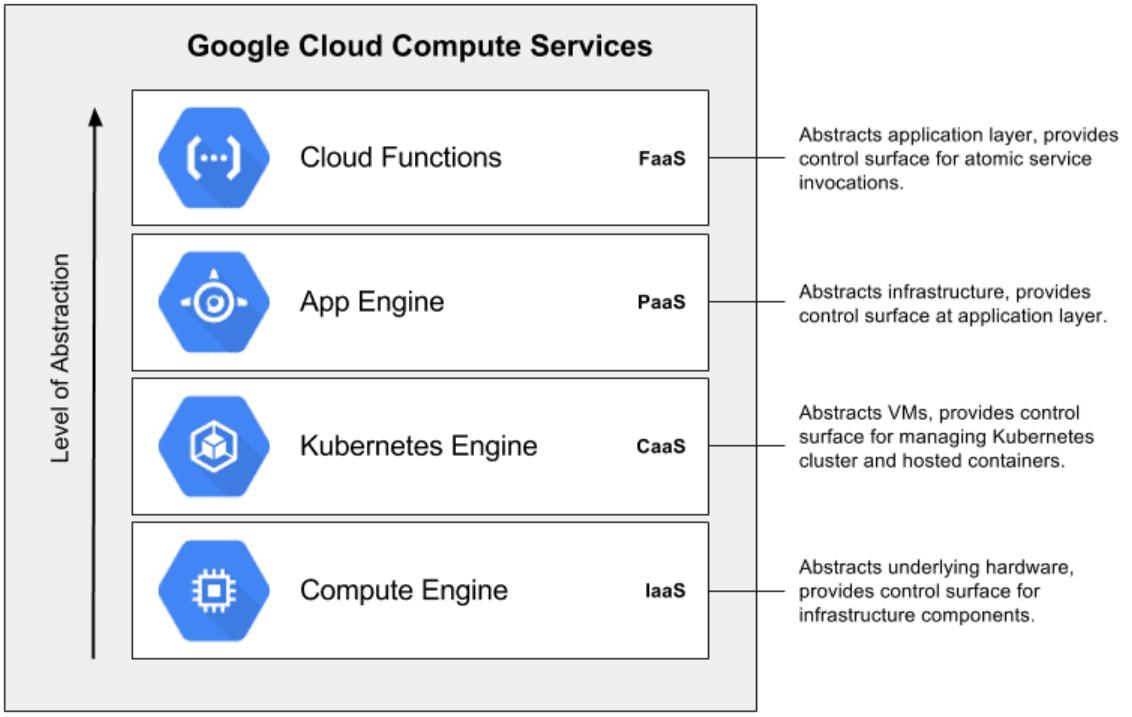


Figure 2.2: Google Cloud Computing services [28]

traffic received. Developers can also use a Software Development Kit (SDK) to develop software products that run on App Engine.

- **Google Cloud Functions:** is a serverless computing type of service to create functions that handle cloud events. It is a solution for developers to create single-purpose, stand-alone functions that respond to cloud events without the need to manage the server or runtime environment, and it can be categorized as a Function as a Service (FaaS). Works well for applications with variable traffic patterns, as it is highly elastic and has minimal operational overhead since it is a serverless platform.

Storage Service and Databases

Google offers a bunch of storage and databases services to cover every type of data, from unstructured data with Cloud Storage, to structured data, where it differentiates transactional workloads from data analytics workload. Inside transactional workload, it offers SQL databases as Cloud SQL or Cloud Spanner, and No-SQL ones as Cloud Datastore. On the analytics workload side, it can be found in Cloud Bigtable and BigQuery.

Figure 2.3 shows the different categorization of data services according to the nature of the data. The most relevant ones which will also be defined on the semantic model explained later on are:

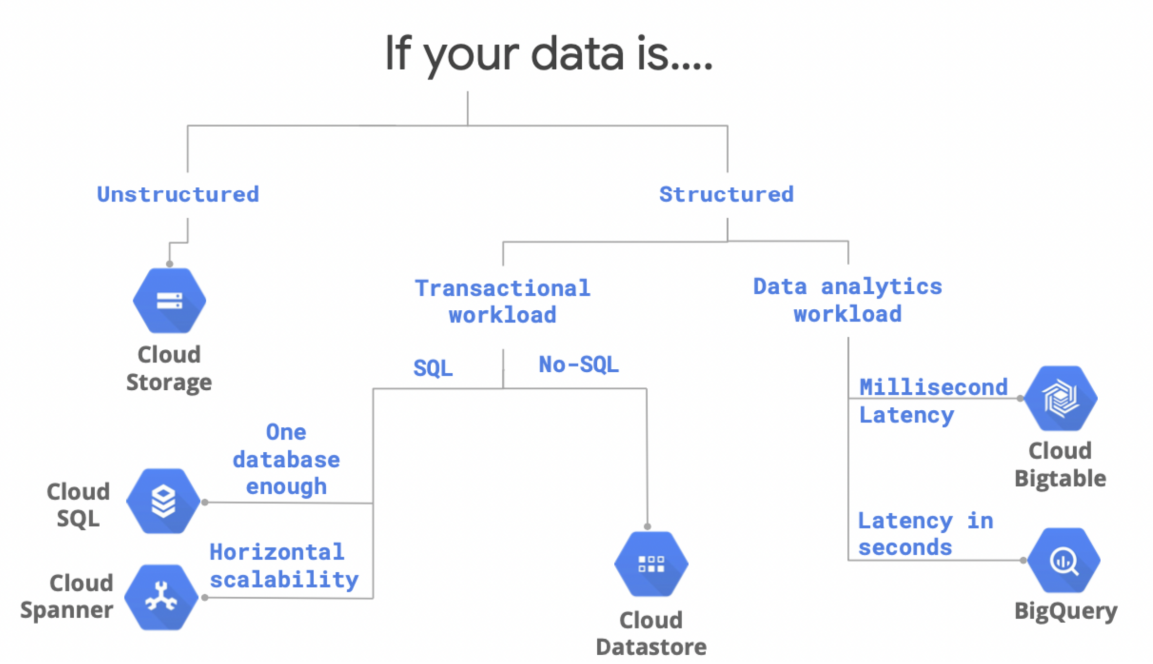


Figure 2.3: Google Cloud Storage and Databases services [17]

- **Google Cloud Storage (GCS):** is the object storage service that provides out-of-the-box features such as object versioning or fine-grain permissions (per object or per bucket), which can make development easy and help reduce operational overheads [1].
- **Google Cloud SQL:** is a fully managed relational database service for MySQL, PostgreSQL, and SQL Server [17], where it is possible to run the same relational databases with rich extension collections, configuration flags, and the developer ecosystem, but without the need of self management.
- **Google Cloud Datastore:** is a highly scalable NoSQL database that handles sharding and replication, resulting in a highly available and durable database that scales automatically and handles application load. It provides variety of capabilities such as ACID transactions, SQL-like queries, or indexes.
- **Google Cloud BigQuery:** is a highly scalable serverless data warehouse that does not require infrastructure management and provides support for analysis of petabyte-scale data volume. It also enables data scientists and data analysts to build and operationalize ML models on planet-scale structured or semi-structured data, directly inside it, using simple SQL.

Furthermore, there are other Google Cloud services that will be used in this project that will interact with the OSLC Adapter module.

Google Cloud Pub/Sub

Google Cloud Pub/Sub is a messaging service for exchanging event data among applications and services. It enables the creation of events producers and consumers' systems, called publishers and subscribers [19]. Publishers communicate with subscribers asynchronously by broadcasting events, and subscribers can establish either a pull subscription, where they need a request, or a push subscription, where the messages are automatically delivered to them.

Google Cloud Logging

Google Cloud Logging is a fully managed real-time log management tool with storage, search, analysis, and alerting capabilities. It enables to search, sort, and analyze logs by using simple and flexible query statements. It also offers rich histogram visualization, a simple field explorer, and the ability to store and save the queries made [16].

Another capability of the service is that it is possible to set alerts to notify whenever a specific message appears on the incoming logs, or it is possible to use Cloud Monitoring to alert on log-based metrics previously defined.

2.2 Open Services for Lifecycle Collaboration

2.2.1 Introduction

Open Services for Lifecycle Collaboration (OSLC) is a set of specifications that are developed to simplify tool integration across the software delivery lifecycle. To avoid long-standing obstacles to effective integrations for lifecycle products, a standard has been developed to support the creation of large-scale and easily maintainable integrations in a heterogeneous tools environment. OSLC is based on the principles of the World Wide Web and Linked Data to create a cohesive set of specifications that can enable products, services, and other distributed network resources to interoperate successfully [21].

OSLC is motivated by domain-driven scenarios that inspire standardization of common capabilities across different disciplines such as change management, requirements management, and quality management, as well as Application Lifecycle Management (ALM) and DevOps, Product Lifecycle Management (PLM), and Integrated Service Management (ISM). OSLC focuses on software lifecycle management as it ensures it meets a set of scenarios and requirements; however, it can be used by tools from any other domain and cross-domain scenarios as IoT or Customer Relationship Management (CRM) [24]. Tools that use OSLC specifications allow an easy tool integration from different vendors and better share information among tools.

As defined by W3C, linked data follows the guidelines of using Uniform Resource Identifier (URI) as names for resources, HTTP URIs so that people can look up those names, useful information by using the standards (Resource Definition Framework - RDF SPARQL) and including links to other URIs so people can discover more resources.

Moreover, the use of linked data makes it possible to access data directly from another product as enables meaningful relationship and traceability across data linkages. In that way, tools can be enabled as OSLC consumers, providers, or both. OSLC consumers can retrieve data in the form of resources from tools that are enabled as OSLC providers [20].

As explained above, OSLC covers multiple domains with respect to different tools with different capabilities. In Section 2.2.2, some of them will be explained.

2.2.2 OSLC Domains

OSLC provides different specifications for use in tools regarding change, configuration, and asset management domains supporting scenarios motivated from Application Lifecycle Management (ALM), Product Lifecycle Management (PLM), Integrated Service Management (ISM), Cloud Computing, and DevOps.

Change Management (CM) domain provides specifications for product change requests, activities, tasks and relationships between those and related resources such as requirements, test cases or architectural resources; **Configuration Management** domain provides specifications for managing versions and configurations of linked data resources form multiple domains; **Asset Management (AM)** domain provides specifications for cataloging, govern, manage, searching for, and maintaining assets [10].

In each of its domains, OSLC defines a set of specified HTTP-based RESTful interfaces in terms of HTTP methods: GET, POST, PUT and DELETE, HTTP response codes, mime type handling, and resource formats.

Automation

Focusing on the scope of this project, the OSLC domain to be considered is the Automation domain. Automation resources define automation plans, automation requests, and automation results of the software development, test, and deployment lifecycle. This domain refers to the use of IT systems such as servers, workstations, and smart hand-held devices to improve efficiency and reduce the need for manual human interactions in the lifecycle of software development, test, and deployment [11].

2.2.3 OSLC Core

OSLC Core defines the overall approach to Open Services for Lifecycle Collaboration (OSLC) based specifications and capabilities that extend and complement the W3C Linked Data Platform (LDP). It covers specific capabilities that are often needed across various domains. In Figure 2.4, some of the core concepts of OSLC are shown.

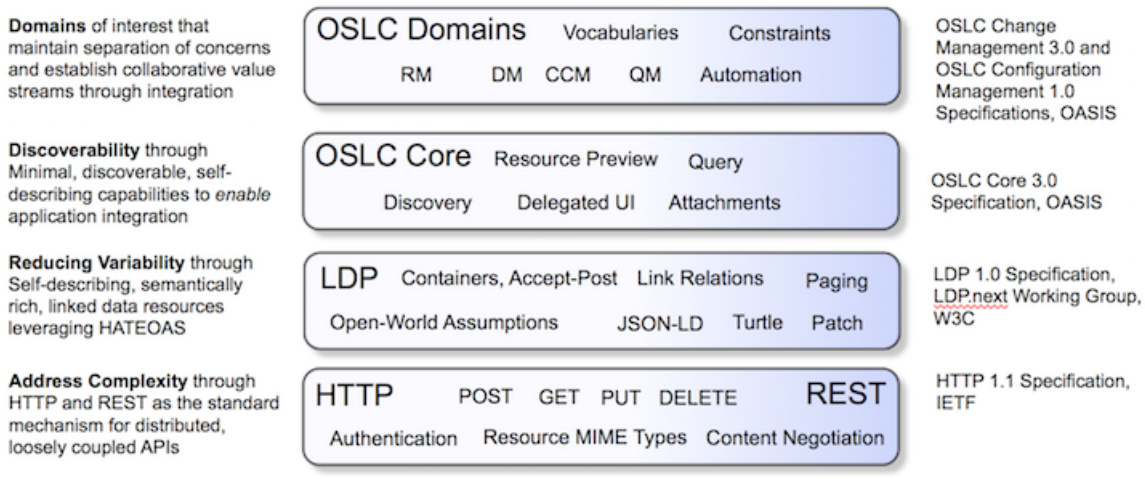


Figure 2.4: OSLC Core 3.0 Architecture [24]

Regarding the OSLC Core specification, some concepts can be defined:

Discovery

It defines a capability providing client applications a standard way to introduce servers to determine which type of resource the server supports, how to preview, select or create instances of those resource, and any constraints on resource creation or update. In that way, it allows clients to determine what capabilities are provided by a server so they can adapt and integrate with different servers in support of end user integration scenarios.

Some of the key usage scenarios for discovery specifications can be a certain authentication model (such as OAuth2[1] or HTTPS Basic), creating resources with a given type, user interface previews of a given resource, querying resources to select specific instances and property values or adding attachments to resources [12].

In the figure 2.5, all the different components of the discovery capabilities of the OSLC Core specification can be found. In the following, some of the most important concepts will be explained.

[1] https://oauth.net/2/

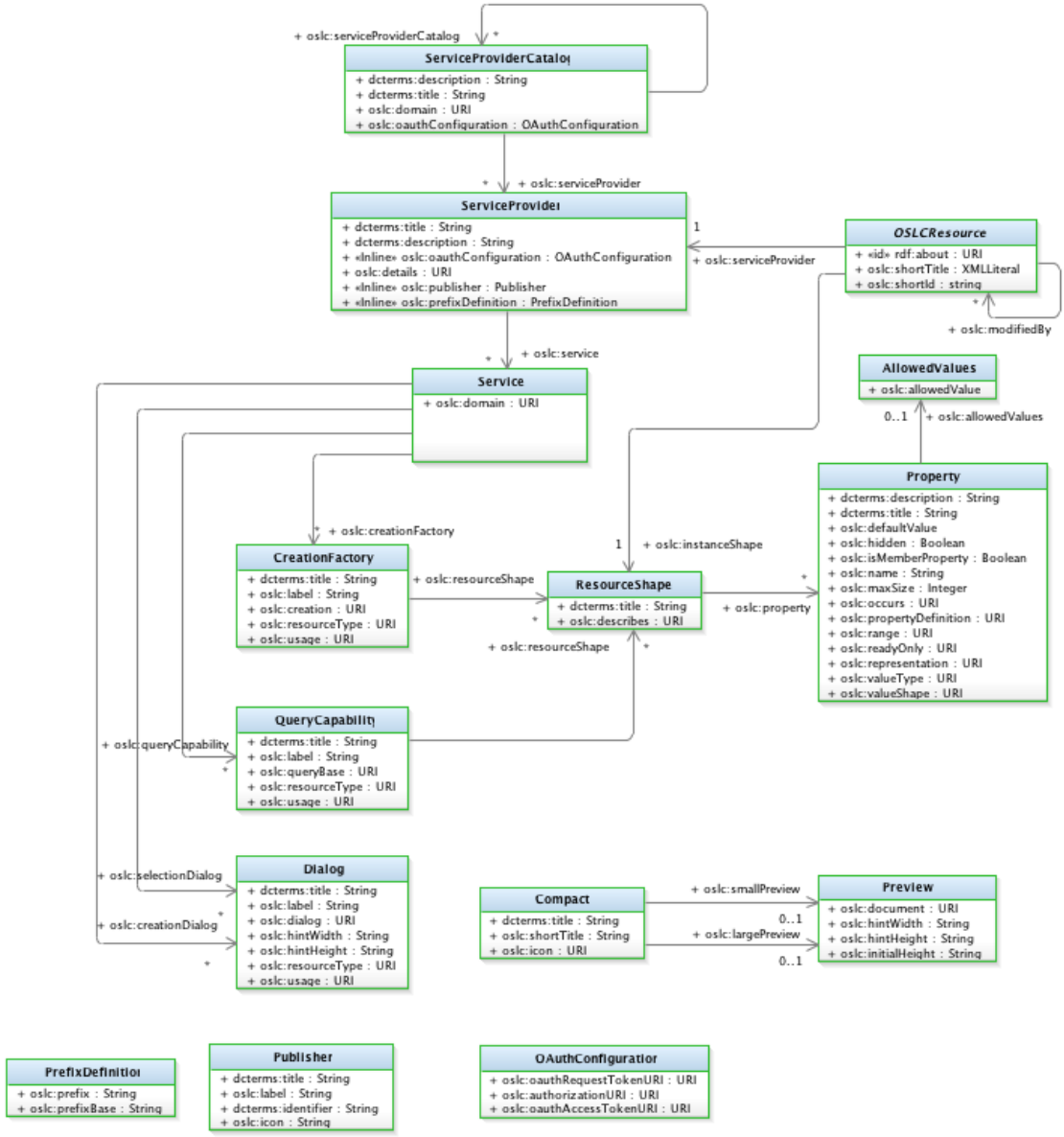


Figure 2.5: OSLC Core 3.0 Architecture [24]

Service Provider Catalog describes an OSLC server that offers a set of service providers, and also may include other nested catalogs. In the case of the actual project, the service provider catalog is Google Cloud Services catalog, which contains multiple service providers.

A **Service Provider** contains different services. As Google Cloud provides numerous services, in this case, service providers are mapped with each of the services that Google offers, i.e. Cloud Storage, Compute Engine or Kubernetes Engine. Each of the ones offers different **Services** such as creation of buckets (GCS), creation of virtual images (GCE), or

creation of clusters (GKE).

Creation Factory describes a capability to create resources including a new catalog capable of creating and containing new resources via HTTP POST. Also, service providers have a **Query Capability** to retrieve information about resources via HTTP GET or POST.

Furthermore, either the creation factory or the query capability has to meet some properties depending on the service they are modifying or getting resources. In this way, a **Resource Shape** is defined with a set of resources. Each of the resources has a different **Properties**, and, moreover, each property has some **Allowed Values** to be assigned to a determined property of a resource. A visual explanation of the relationship of these last concepts can be found in Figure 2.6.

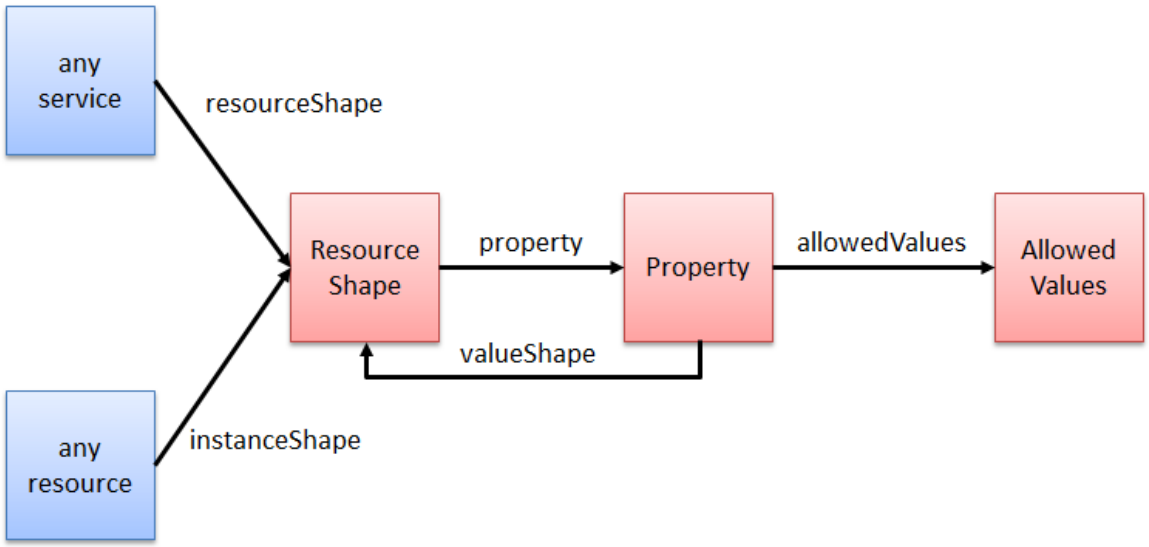


Figure 2.6: Diagram of main concepts and relations [24].

Resource Preview

It describes how a client application can display links and embed rich previews of resources from other applications. Also, links may have a label and an icon, and previews are HTML markup provided by a server and displayed directly inside the client application. Often appears as a pop-up window when the user mouse passes over a link.

Moreover, servers can provide different previews depending on the user screen, so servers suggest sizes for previews and previews can be resized after they are displayed [13] as shown in Figure 2.7.

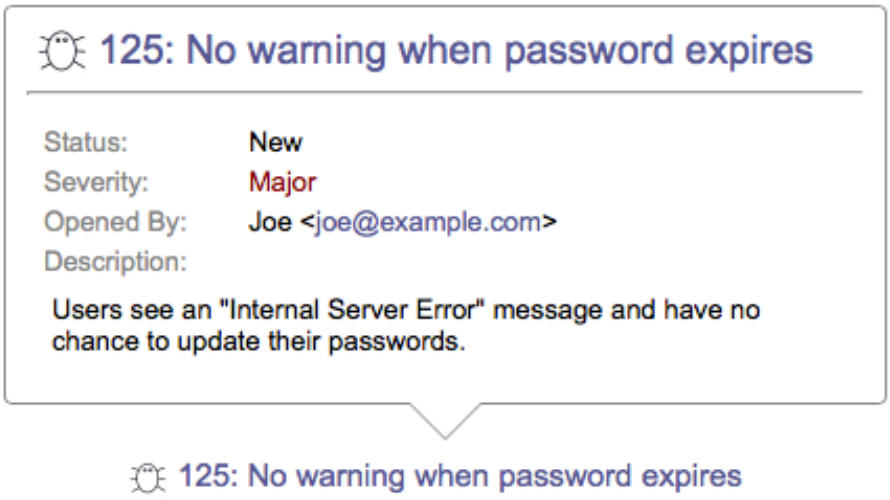


Figure 2.7: Preview for a link [13].

2.2.4 Tracked Resource Set (TRS)

OSLC TRS provides a mechanism for making a set of resources discoverable and expose them in a way that allows clients to discover the exact set of resources in the set, to track all additions and removals over the set, and to track state changes to all resources in the set. In that way, it is able to have a live feed of linked lifecycle data, so other tools can monitor the tracked resources and get metadata information from them.

As well as other specifications, TRS is HTTP-based and follows RESTful principles, so it is possible to retrieve all the lifecycle information of the resources by making an HTTP GET petition.

TRS Server can be defined as a server that contains all information about the life cycle of resources and can also be integrated into an OSLC Server. The server decides which particular resources are in a particular TRS at any moment and also either the TRS or the linked data contents of each Tracked Resource can vary over time.

There are two concepts that are defined in the TRS, a **Base** and a **Change Log**. The first is a Linked Data Platform (LDP) container that provides an enumeration of the Tracked Resources that make up the TRS. The second one describes a history of the different stages through which the TRS has passed, so it has a set of changes of the different resources. Additionally, the TRS Server can periodically update the Base of a TRS and truncate the Change Log to not saturate the server [14].

Events and Actions

Taking into account the previously mentioned concepts, a TRS therefore defines a base

of resources, and each of one has a change log with all of it changes during time. Throughout its lifecycle, it can occur different events over the base resources, such as creation, deletion, or modification events. Thus, the change log has to contain **Change Events** at some interval.

Depending on the type of Change Event, the Change Log can store the specific **Action** that has been done on the TRS, e.g. a creation event which contains an action of creating a new virtual image, bucket or cluster. By combining both concepts, the TRS is being completed so it contains a Base of resources, Change Logs, Events, and Actions, so the resulting resources are exactly tracked over its lifecycle.

2.3 Tools and technologies

In this section, it will be described all of the tools that have been used for the development of the project. As it has two differentiated parts, these tools will be detailed separately on development tools on the one hand and, on the other side, semantic tools.

2.3.1 Development tools

For the development of the whole code, Python programming language has been used, specifically Python 3.10 version, which has introduced Structural Pattern Matching in the form of a match statement and case statements of patterns with associated actions. Patterns consist of sequences, mapping or primitive data types, as well as class instances. Also, pattern matching allows programs to extract information from complex data types and apply certain actions based on different forms of data [27]. In the following, different libraries and packages that have been used will be explained.

Flask

Flask[2] is a Python-written web application framework based on the Werkzeg Web Server Gateway Interface (WSGI) toolkit and the Jinja2 template engine. The first one is a WSGI toolkit that implements WSGI requests, response objects, and utility functions, and enables a web frame to be built on it. The second is a popular template engine for Python that combines a template with a specific data source to render a dynamic web page [25].

Moreover, Flask does not have a built-in database, so it is totally eligible by the developer. In this project, **Flask-SQLAlchemy**[3] is used as an extension for Flask applications to provide support for SQLAlchemy[4] to it, a Python SQL toolkit, and Object Relational

[2]https://flask.palletsprojects.com/en/2.1.x/
[3]https://flask-sqlalchemy.palletsprojects.com/en/2.x/
[4]https://www.sqlalchemy.org

Mapper that provides the data mapper pattern in a Python language, where classes can be mapped to the database in multiple ways.

An important concept to be introduced is RESTful APIs. It is an application programming interface that conforms to the constraints of the REST architectural style for interaction with RESTful web services. First, an API is a set of definitions and protocols for building and integrating application software, therefore, it is possible to interact with a system to retrieve information or perform a specific function. RESTful APIs uses requests to access and use data that can be used to GET, PUT, POST, and DELETE data types, in order to read, update, create, and delete resources.

To provide and integrate the RESTful API with Flask, the **Flask-RESTful**[5] Flask extension is used. In order to build it, it is necessary to configure URL endpoints grouped by resources. Therefore, it is possible to apply REST operations over the endpoints, so that will trigger the desired operation on a specific resource.

RDFLib

RDFLib[6] is a Python package for dealing with RDF resources, and includes parsers and serializers in order to manage different RDF formats as RDF/XML, N3, NTriples, N-Quads, Turtle, TriX, JSON-LD, RDFa and Microdata. In particular, the format used in this project is mainly RDF/XML. It also has in-memory capabilities to provide RDF storage, so it is possible to query the information using SPARQL queries on the Flask endpoints.

RDFLib uses the term **URIRef** to identify resources that provide a URI reference within an RDF **Graph**. The latter is a set of nodes connected by different relationships, as we can see in the Figure 2.8. RDFLib also provides a term to represent blank nodes, the term **BNode**, to describe a resource for which no URI or literal is given, also called an anonymous resource, which can only be used as a subject or object in a triple.

Furthermore, the **Literal** term is used as attributes values in RDF, such as a person's name, date of birth, or height. It has a datatype as string or double, or a language tag, e.g. English. Also, RDFLib provides the term **Namespace** in order to easily create URIs in a namespace, such as the OSLC namespace: "http://open-services.net/ns/core ".

Requests

Python Requests[7] is the de facto standard for making HTTP/1.1 requests in Python. Abstract the complexities of making requests by using a simple API that allows the developer to focus on interacting with services and consuming data in their application. It

[5] https://flask-restful.readthedocs.io/en/latest/
[6] https://rdflib.readthedocs.io/en/stable/
[7] https://docs.python-requests.org/en/latest/

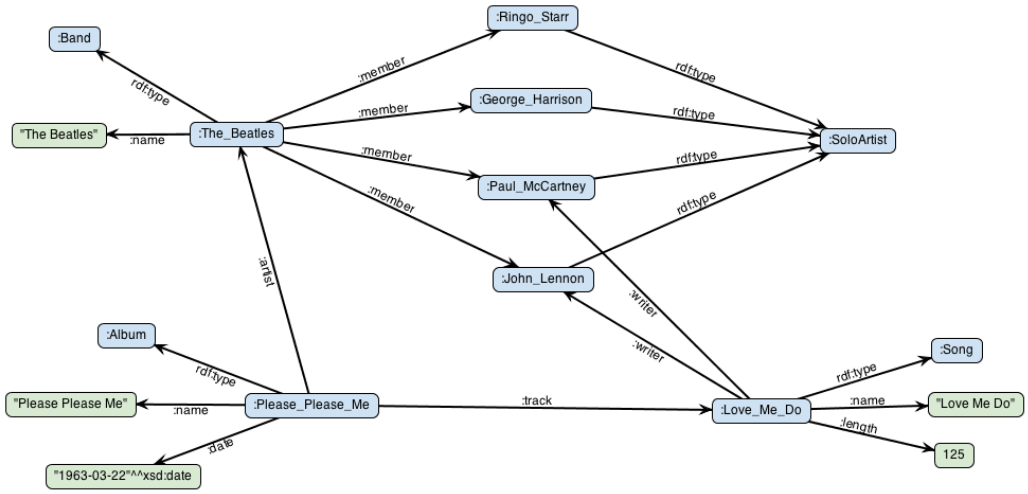


Figure 2.8: RDF Graph example about The Beatles [31]

provides functionality to make GET, POST, PUT, or DELETE methods, manage response errors and status codes, managing content, dealing with headers and reading the message body, among other functions.

Google Python Client

In order to use Google APIs, it is needed to import the Google API Client Library, which is fully supported by Google. It offers different APIs to access its different services. Taking into account the scope of the project, the following packages have been used: **google.cloud.storage** package, to manage GCS resources like Buckets or Objects; **google.cloud.compute** package, for dealing with virtual machines, also called Instances; and **googleapiclient.discovery** package, for building client libraries that interact with Google APIs.

Moreover, to use the Google API Client, one needs to indicate a valid JSON format credential that is generated on GCP after being logged in with an active account. Otherwise, it is not possible to use any of them.

Docker and Docker-Compose

Docker is a software platform that allows running applications in isolated environments called **containers** that has everything the software needs to run including libraries, system tools, code and run-time. It allows us to build, test, and deploy applications in a fast way. A Docker container is a running instance of a Docker image, and they are stored temporarily, whereas images are stored permanently. It is one of the most widely used virtualization tools nowadays.

Docker container is a standardized unit which can be created on the go, in order to deploy

a particular application or environment, such as an Ubuntu container, CentOS container, Flask container, etc., to fulfill the requirement from an operative system point of view.

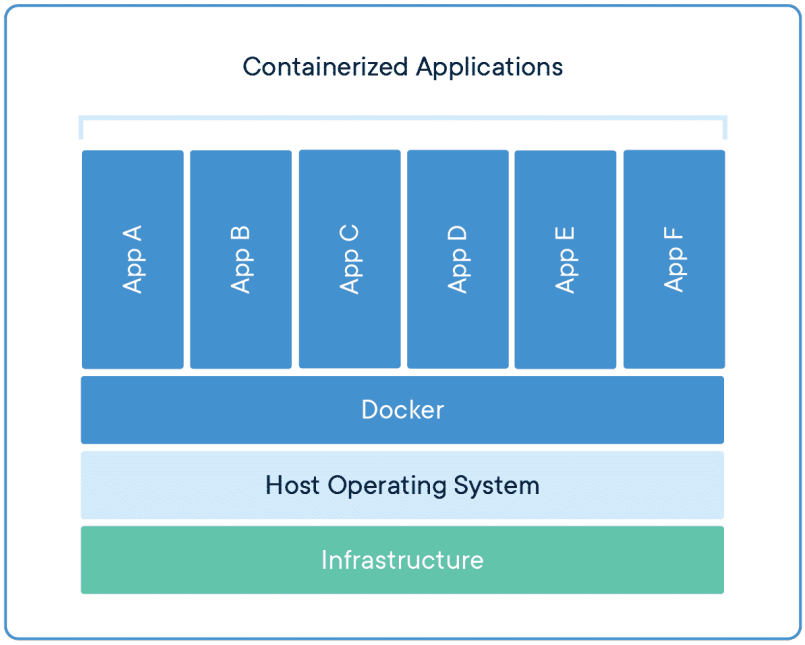


Figure 2.9: Docker Container underlying architecture [7]

As can be seen in figure 2.9, containers can be compared with Virtual Machines (VM), as they have similar resource isolation and allocation benefits, but function differently as containers virtualize the operating system instead of hardware, in the case of VMs.

Compared to figure 2.10, containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, whereas each VM has its own OS managed by a hypervisor. The main comparison is made with respect to three parameters: size, start-up, and integration.

Size, as VMs memory is allocated and in case of no use, it is wasted, whereas with containers, the CPU dynamically allocates memory with the exact amount needed by the container, and there is no waste, so additional containers can be created with the leftover RAM. **Start-up**, as VMs take a lot of time to boot up because the gest OS needs to start from scratch, while the container runs on your host OS and provides a faster bootup. **Integration**, as using VMs, it is only possible to have a limited number of DevOps tools running in it, and, in case multiple instances of different tools are running, multiple VMs are needed. Using Docker Containers, multiple instances of DevOps tools can be used, setting up many instances of them, but all running wither on the same container or in different containers that interact with each other.

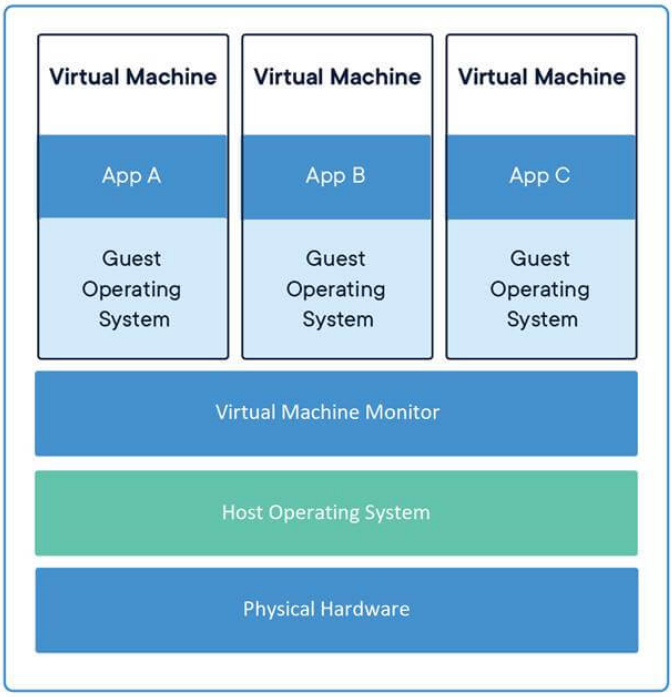


Figure 2.10: Virtual Machine underlying architecture

Consequently, Docker provides multiple advantages for businesses, such as **return on investment and cost savings**, due to the fact that the use of Docker facilitates this type of savings by dramatically reducing infrastructure resources, since fewer of them are required to run the same application. Docker containers also provide an improvement over **standardization and productivity**, as they ensure consistency throughout multiple development and release cycles, standardizing the environment. This allows developers to fix any upgrade if the whole environment is broken, as the code can be rolled back and tested once again.

Moreover, Docker provides **CI efficiency** which enables the possibility of building a container image and using that same image at every step of the deployment process. This allows us to separate non-dependent steps and run them in parallel. Also, it provides benefits in **compatibility and maintainability**, as it provides parity, which means, in terms of Docker, that images can run the same despite the server or laptop on which they are running. Finally, Docker provides support with **multi-cloud platforms** due to portability, since over the last few years, all major cloud computing providers ranging from AWS, GCP or Azure, have embraced Docker and added individual support. Therefore, Docker containers can run inside an Amazon EC2 instance, GCE instance, Rackspace server, or VirtualBox [23].

In addition, Docker-Compose complements Docker as is a tool for defining and running multi-container Docker applications that can handle multiple containers simultaneously

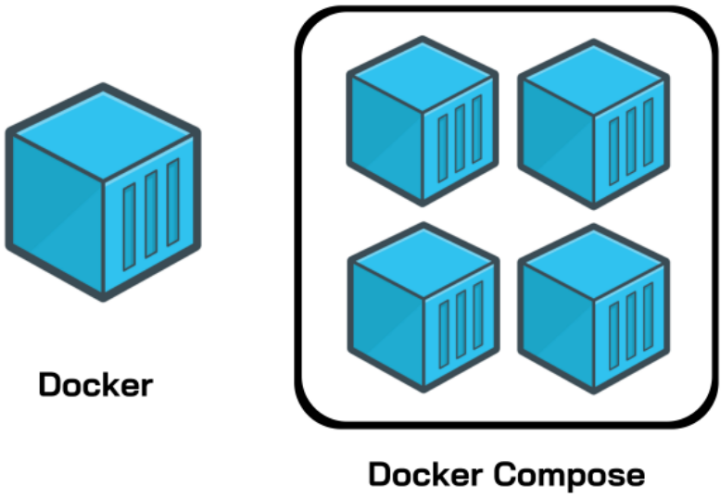


Figure 2.11: Docker Compose illustration

in production, staging, development, testing, and CI environment. It works by applying rules defined in the *docker-compose.yml* file, which is used for configuring the application's services and includes specific rules for defining how they are run. Provides an easy way to start, stop, or rebuild all services with a single command [30].

Common use cases of Docker Compose typically are **automated testing environments**, since it supports automated testing, an essential part of CI/CD so it can create and destroy the required testing environment, and developers can configure the environment needed for running automated end-to-end testing using the appropriate Docker Compose file. Docker Compose also allows **single host deployments**, as containers are designed to run on a single host to focus on development and testing workflows. Finally, Docker Compose provides useful **development environments** since it provides a fast and simple way to start projects due to the fast spin-up of isolated development environments.

DuckDNS

Dynamic Domain Name System (DDNS) services are designed to convert the public IP of a specific machine into a certain domain, as IP addresses are difficult to remember and, in addition, some operators have dynamic IP addressing, which means that the IP will change on time, so it is needed that these services always find where the servers are. Only a configured domain will be needed that will point directly to the public IP of the router.

DuckDNS[8] is a free DDNS service hosted on AWS, which points the DNS subdomains of "duckdns.org" to an IP of choice. It is fully compatible with any Operating System (OS),

[8]https://www.duckdns.org

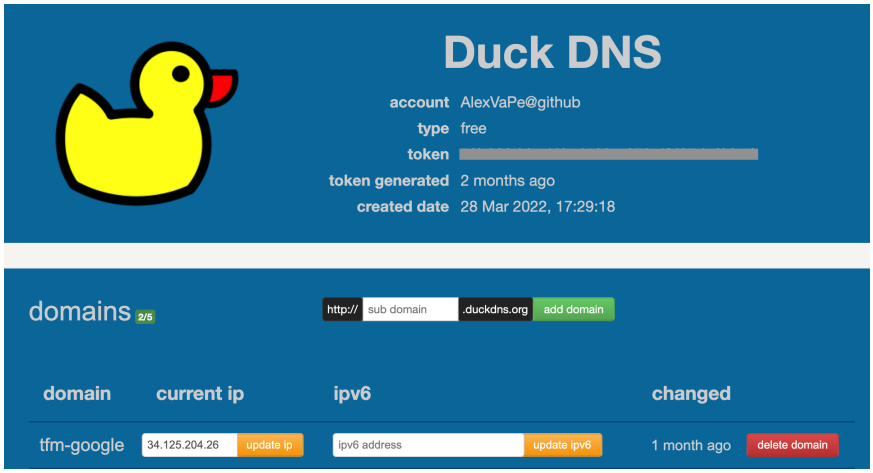


Figure 2.12: DuckDNS webpage

so it can be installed on Windows, Linux, or macOS devices, and its operation is as simple as configuring the service with a repetitive task of the operating system (named cron job), so that a command is executed in order to automatically update the public IP address. The service is also compatible with multiple routers on the market and even NAS servers, since it is as simple as making a request via HTTP or HTTPS following a specific syntax.

Let's encrypt

Let's Encrypt is a Certificate Authority that enable HTTPS on a website by providing a valid certificate. It is designed to simplify the acquisition of SSL/TLS digital certificates providing a site's authenticity, while also providing encyption. Its automated processes also help reduce page errors due to out-of-date certificates.

With Let's Encrypt, it is demonstrated control over that domain is demonstrated by using software that uses the Automatic Certificate Management Environment (ACME) protocol running on the web host [9]. It also simplifies through automation, since all the processes involved in providing proof of control for a website, such as obtaining, renewing, and revoking certificates, are automated.

Dehydrated

Dehydrated is a client for signing certificates with ACME-server as Let's Encrypt implemented as a simple executable bash script. It uses the openssl[9] utility to handle keys and certificates. Some of its features include signing a list of domains, signing a custom Certificate Signing Request (CSR), requesting certification when it is about to expire, or revocation of the certificate [29].

[9]https://www.openssl.org

Kafka

Apache Kafka is an open-source streaming data platform originally developed by LinkedIn, but later donated to Apache for further development. It operates as a traditional pub-sub message queue, such as RabbitMQ[10], and allows the publication and subscription to message streams. It is optimized for ingestion and processing streaming data in real time. Streaming data is data that are continuously generated by thousands of data sources, which typically send the data records in simultaneously, so a streaming platform needs to handle the constant flux of data and process them sequentially and incrementally.

It differs from traditional message queues in three key ways. First, Kafka operates as a modern distributed system that runs as a cluster and can scale to handle any number of applications. Second, it is designed to serve as a storage system, since it can store data as long as necessary, whereas most traditional message queues remove messages immediately after the consumer confirms receipt. Finally, Kafka can handle stream processing, computing derived streams and datasets dynamically, rather than simply passing batches of messages [15].

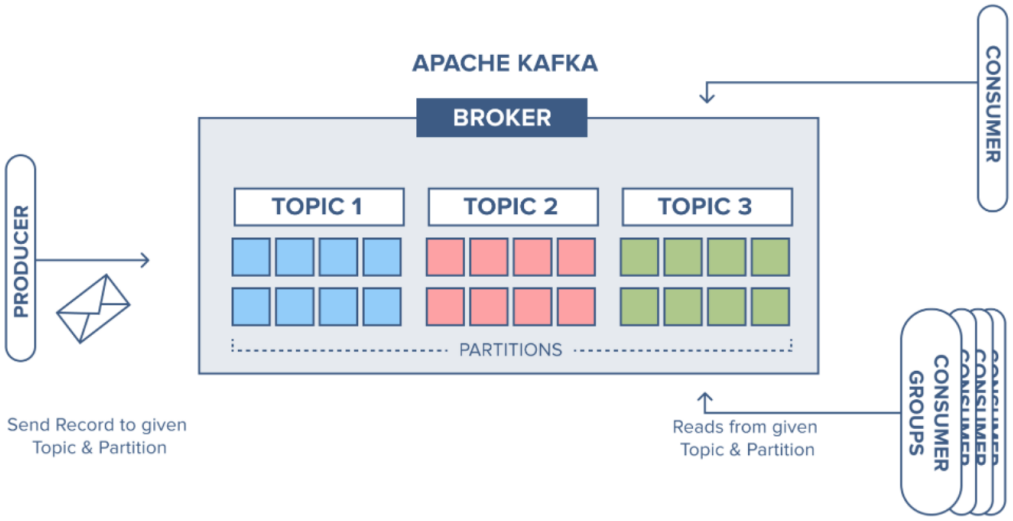


Figure 2.13: Record flow in Apache Kafka

As it is illustrated in figure 2.13, Kafka accepts streams of events written by data producers. It stores records chronologically in partitions across brokers which, in multiplicity, form a cluster. Thus, each record contains information about an event and consists of a key-value pair, with timestamp and a header. also, Kafka groups records into topics, and data consumers pull their data by subscribing to the topics to read the messages.

[10]https://www.rabbitmq.com/

An **event** records any change in something related to the business. It is the form in which you read or write data to Kafka.

Figure 2.14: Kafka message example

Producers are those client applications that publish events in Kafka, and **consumers** are the applications that subscribe, read and process events. The key advantage of Kafka is that producers and consumers are fully decoupled and agnostic of each other, so they can achieve high scalability. That means that producers do not need to wait for consumers.

Sometimes consumers can act as producers and consumers simultaneously. Examples of consumers can be databases, data lakes, or data analytics applications, as they store or analyze the data they receive from Kafka. Thus, Kafka acts as a middleware between producers and consumers.

A **topic** is a category or feed name to which records are stored and published. Producer applications write data in topics, and consumers applications read from them. Moreover, topics in Kafka are always multi-producer and multi-subscriber, since a topic can have zero, one, or many producers that write events to it, as well as zero, one, or many consumers that subscribe to these events. Additionally, topics can be configured differently depending on how much time is needed to retain the messages after consumption. This capability does not affect performance with respect to data size, so storing data for a long time is not negative [2].

Additionally, as is seen in figure 2.13, topics are **partitioned**, which means that topic is spread over a number of buckets located on different Kafka brokers. Each record in a partition is assigned and identified by its unique offset. Partition allows topics to be parallelized by splitting the data into a particular topic across multiple brokers.

When a new event is published to a topic, it is appended to one of the topic's partitions,

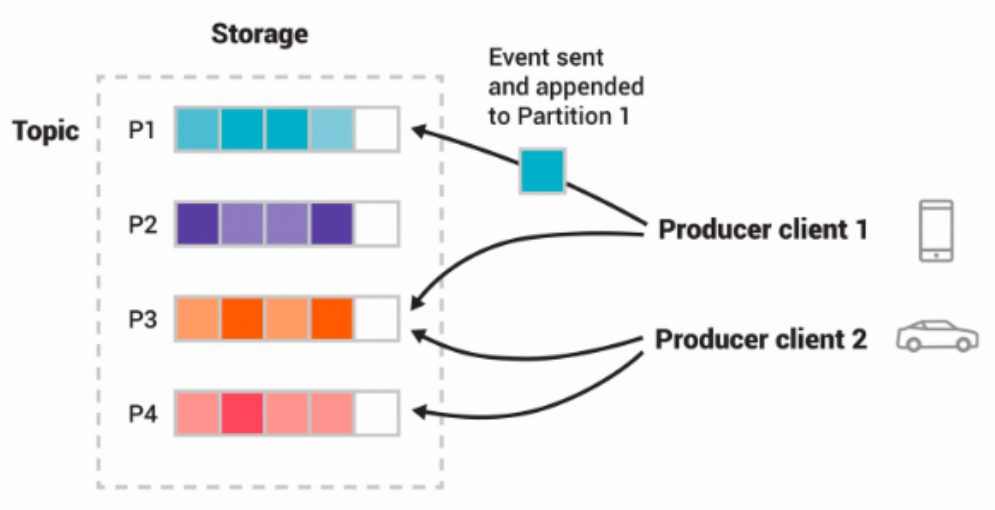


Figure 2.15: Kafka partitions

and, events with the same event key are written to the same partition, so that Kafka guarantees that any consumer of a given topic partition will always read that partition's events in exactly the same order as they were written.

Finally, in Kafka, **replication** is implemented at the partition level. The partition unit of a topic partition is called a replica. Each partition usually has one or more replicas, so that the partitions contain messages that are replicated over a few Kafka brokers in the cluster. The way partition replicas works is that each one has one server acting as a leaderr and the rest of them followers. Thus, the leader replica handles all read-write requests, and the followers replicate the leader. If the leader fails, one of the following servers becomes the leader by default.

2.3.2 Semantic tools

Protégé

Protégé[11] is a free open-source ontology editor and knowledge base framework that allows users to build ontologies for the Semantic Web. It allows to load, save, edit, and visualize ontologies in OWL (Web Ontology Language) and RDF. Moreover, you can import other ontologies to your active one and modify all of its entities, such as classes, object properties, data properties, annotation properties, datatypes, or individuals. It also provides a graphical tool **OWLViz**, to visualize your active ontology classes and dependencies with others. Finally, Protégé ontologies can be exported into a variety of formats, including RDF, OWL, Extended Markup Language (XML), or JSON Linked Data format (JSON-LD). The

[11] https://protege.stanford.edu

actual interface of the application can be found in figure 2.16.

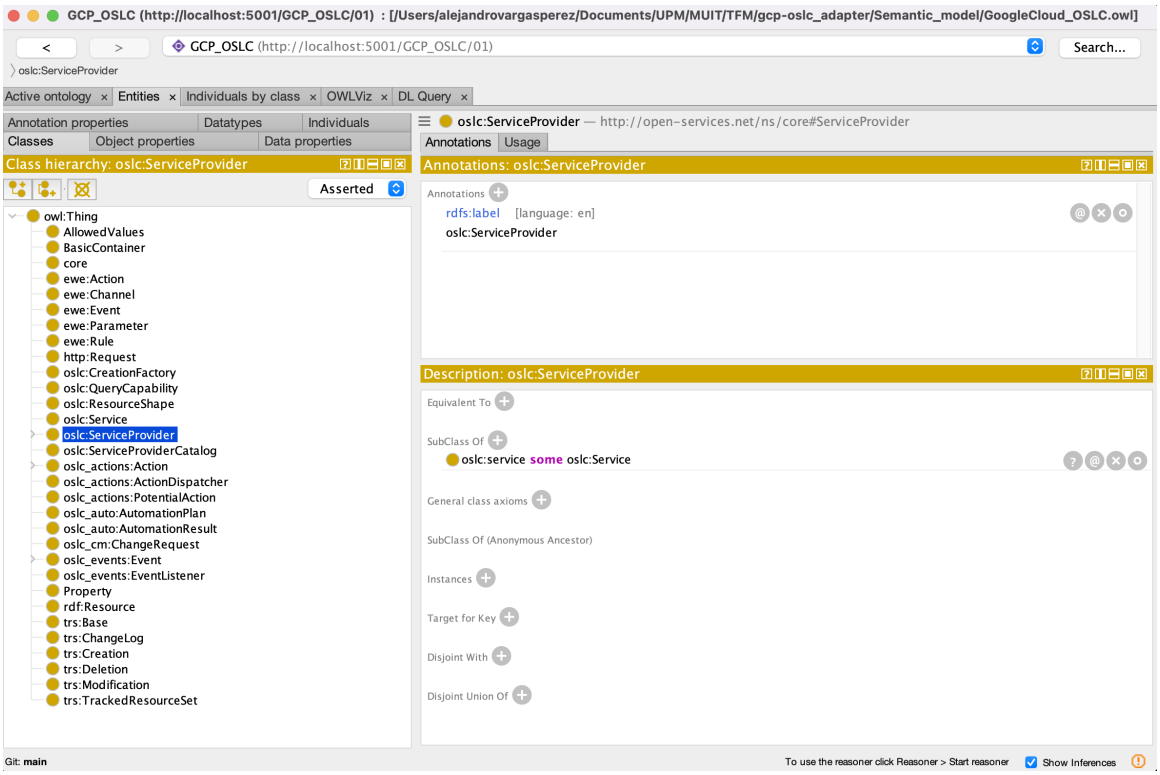


Figure 2.16: Protégé application interface on MacOS

Since one of the primary goals for generating web ontologies is to make data more shareable and accessible, visualization is an essential component of the process. Therefore, Protégé offers tools to transform ontologies into dynamic visualizations to share data structures and taxonomic features. These visualizations can also allow for interactive navigation of the data, creating an intuitive way for collaborators to understand the data structures and relationships.

Furthermore, since Protégé is designed to facilitate collaboration, it also provides customizable versioning and change tracking capabilities. As new data are generated and integrated into the active ontology, each of the modifications done by team members is left documented. This provides organization of the data as well as a ready documentation of the different stages of the project development over time.

Insomnia

Insomnia[12] is an REST client for testing RESTful APIs. It is a free cross-platform desktop framework that includes a user-friendly user interface as well as useful features such as an easy organization of all type of requests, managing multiple environments variables

[12]https://insomnia.rest

with different authentication credentials or tokens, multi-protocol support (REST, SOAP, GraphQL or GRPC) or code generation with code snippets on different languages like Curl, NodeJS, Go, Swift, or Python.

Some of its key features are **exportable workspaces**, providing the capability for exporting a JSON document for your environment and sharing with others; **swappable environments**, since it allows setting environment variables that allow testers to reuse multiple requests, for example setting a "base_url" variable that can be used in multiple environments. It is useful for projects that use different back-end, URL, and HTTP authentication credentials for development, testing, and production, so this feature facilitates the friction of switching between those URLs by allowing users to create an Insomnia environment for each back-end. It also provides **chained request** capability, when there is sometimes an endpoint that provides a token to access other endpoints, and it is possible to set up a request that triggers a generated token endpoint, and then use its response with other endpoints. Finally, it provides **end-to-end encryption**, which means that encryption is performed before sending any data over the network, keys are generated locally, and decryption is performed after receiving data from the network.

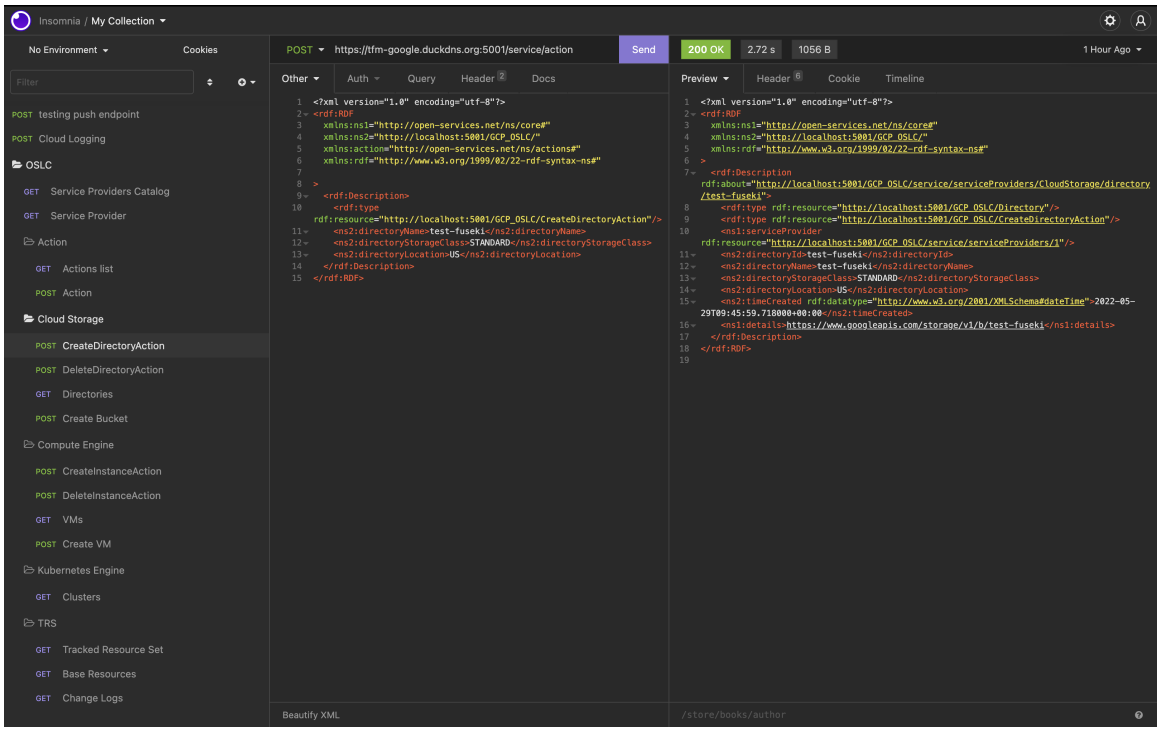


Figure 2.17: Insomnia application interface on MacOS

Therefore, it is possible to create a set of requests within a project, as shown on its interface in Figure 2.17. This tool has been used in this project to make HTTP requests to different endpoints where its body contains an rdf/xml format resource that is processed and

integrated with the cloud ontology generated with Protégé and finally receives a response with a resource in the same format.

Apache Jena Fuseki

Apache Jena Fuseki is an SPARQL server, an open-source RDF graph database that stores triples. It can run as an operating system service, as a Java Web Application (WAR file), and as a standalone server. Thus, it comes in two forms, a single system web app combined with a user interface for administration and query, as can be seen in Figure 2.18, and as a main server, suitable to run as part of a larger deployment, together with Docker or running embedded.

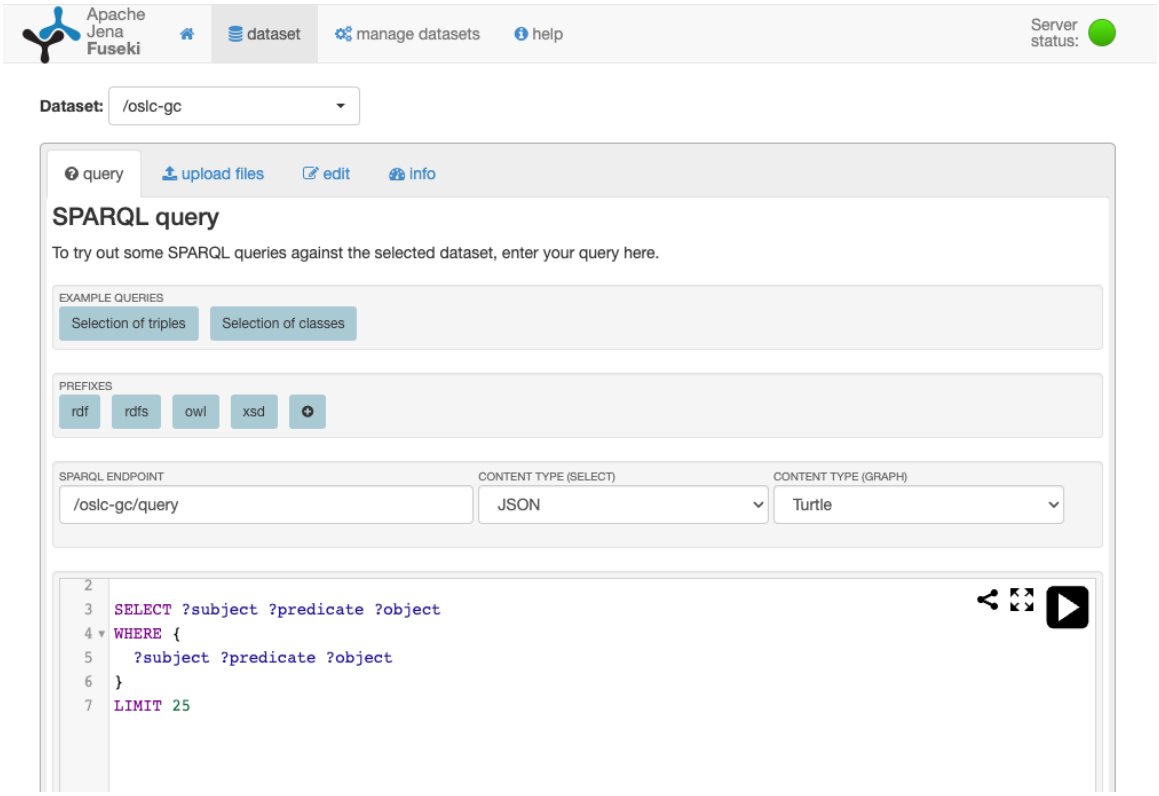


Figure 2.18: Fuseki web UI

Fuseki provides the SPARQL 1.1 protocols for query and update, as well as the SPARQL Graph Store protocol, and it stores the information within different datasets.

CHAPTER 3

Architecture

This chapter presents the methodology used in this work. It describes the overall architecture of the project, with the connections between the different components involved in the development of the project.

3.1 Introduction

This project is part of the SmartDevOps project, carried out by the GSI-UPM department of ETSIT University. As part of the project, an introduction of the global architecture is needed to understand every aspect and motivation of the actual project. In that way, in this section, it will be explained the different blocks that participate on the performance of the global project as well as a detailed description of the architecture of the actual project of the document. Then, a detailed explanation of the actual project will be included in the next section.

Firstly, a general idea of the blocks of the global project is needed. Mainly, it is composed of the following blocks: a **Big Data Environment** that contains all the possible useful tools to be integrated depending on the scope of the solution, where it can also be included in cloud technologies as is explained on this project, a set of **DevOps** tools to interact with the whole Big Data environment, an **OSLC Adapter** that matches the entire environment resources with semantic vocabulary and where **Event Automation** is included, and finally, the actual users and developers. This general idea is illustrated in the figure 3.1.

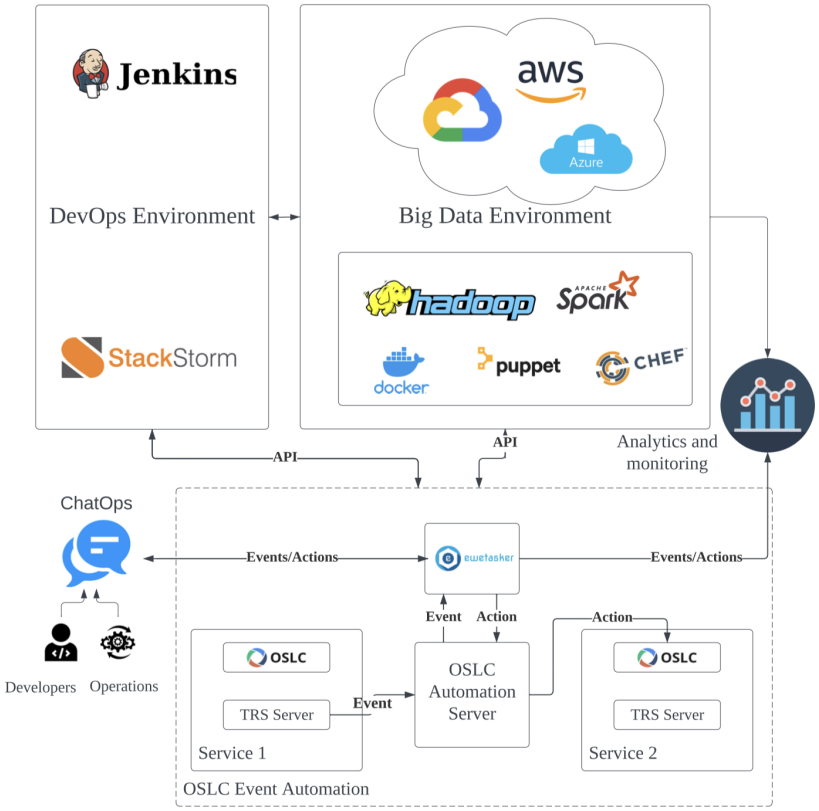


Figure 3.1: Generic structure of SmartDevOps project

3.2 Global architecture

Taking into account the structure mentioned above, a global architecture of the Smart-DevOps project has been developed more in depth and can be found in Figure 3.2. The complete set of blocks that conforms to the global architecture of the project is shown and allows a better understanding of the context of the actual project, whose architecture will be detailed in Section 3.4.

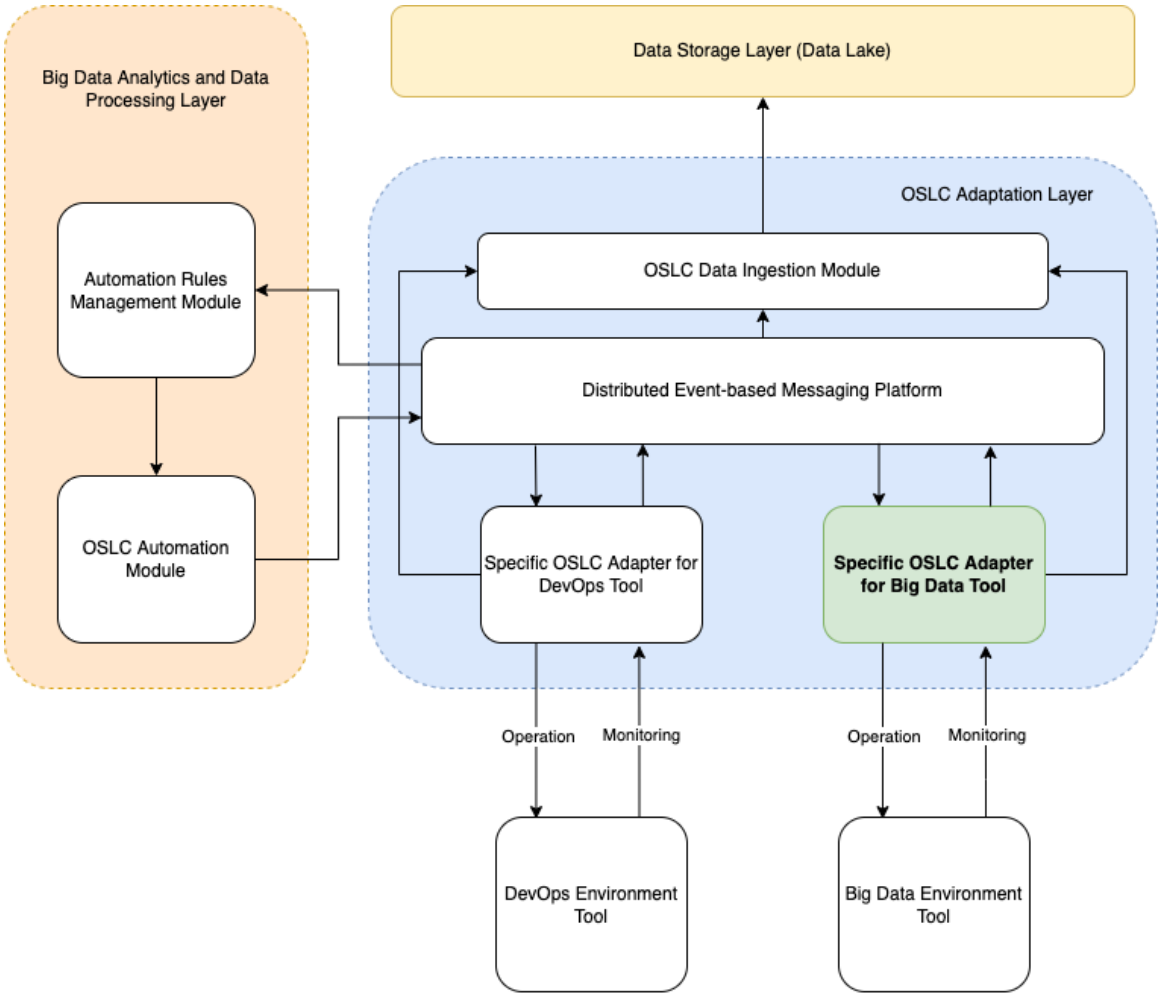


Figure 3.2: Global architecture of SmartDevOps project

The global architecture of the SmartDevOps project is composed of the following blocks:

Big Data environment

Composed of any of the actual existing Big Data tools nowadays, from Apache Hadoop with it different solutions to storage, with HDFS, it data transfer tools as Apache Sqoop or Flume, it resource management solution YARN, it data streaming solution Apache Kafka,

it in-memory data processing Apache Spark or it NoSQL database solution HBase.

Also, it covers latest Big Data technologies on cloud platforms, as ones that offer the different vendors like Google with Google Cloud Platform, Amazon with Amazon Web Services or Microsoft with Azure. Almost every Big Data tool mentioned above can be replaced with services that offer any of the three cloud providers.

DevOps environment

DevOps tools usually can be linked with the Big Data environment, as they provide automation capabilities in order to deploy efficiently infrastructure that allows Big Data tools to run over it. Here, we can find tools such as Jenkins or Stackstorm that can be integrated into the architecture to provide automation of code deployment in response to events. For example, Jenkins can be linked to GitHub, so when it detects a repository change on a specific branch, it can trigger a code that deploys that change in the live environment.

Data Storage Layer

This module contains different blocks that are in charge of the storage of resources. Thus, it is comprised of Semantic Data, temporal series/metrics, and natural language data. It is in charge of storing data from the output of the OSLC Adaptation layer, as well as from the massive data ingestion layer. Also, it exposes these data to be used by the Data Visualization layer.

Big Data Analytics and Data Processing Layer

Here, the Automation Rules Management module can be found, as well as the OSLC Automation module. The automation module and the OSLC adaptation layer are the most important blocks of the project, since all flows flow into and out of them.

OSLC Automation Module is one of the most important modules of the global project, as all workflows interact with it and participate in the core idea of the project, the automation of events.

The module in charge of these events is **Automation Rules Management module**, which is made up of EWETasker[1], a Task Automation Server (TAS), to provide automation capabilities to the architecture, based on the Evented Web (EWE) ontology. This TAS consists of events that, if certain conditions are met, trigger Rules that execute actions.

To manage the interaction between OSLC services and TAS, an Automation OSLC Server with a standard interface is needed. Therefore, it allows us to define automated workflows that interact with all of the resources of the different services.

[1] https://ewetasker.readthedocs.io/en/latest/ewetasker.html

OSLC Adaptation

The OSLC Adaptation module is one of the core modules of the actual project, as it will be in charge of communication with the Big Data infrastructure and generate OSLC resources from the received information. It also works in the opposite direction, as it can process HTTP petitions in a compatible OSLC format to act later on the infrastructure. It is also connected to this module a ChatOps, which allows developers and operations specialists to perform **Operation** and **Monitorization** actions, in order to communicate and execute orders by using natural language trigger actions in the rest of the system through the OSLC interface. Actions triggered by the EWETasker rule motor will be displayed in the chat.

The **core objective** of the actual project is the development of an OSLC Adapter to cover all the functions mentioned above, and, in this case, the Big Data Infrastructure is hosted on Google Cloud Platform, but it will be developed in order to facilitate integration with any other cloud provider.

The developed OSLC Adapter integrates seamlessly into the architecture shown in figure 3.2, as it can interact with the Big Data tool and communicates with a Distributed Event-based Messaging Platform by sending messages containing OSLC Events. As will be explained in Section 3.4.2, a Kafka messaging platform will be used to receive these events.

3.3 Cloud Semantic Model

In the first part of the development of the project, it has been done a deep research of OSLC standard and core concepts, including all of its domains and specifications in order to be able to build a generic and standard cloud semantic model that extends OSLC specification and allows to build an OSLC Adaptation module, as previously mentioned, that is able to integrate cloud resources with semantic OSLC resources.

It has extended the core vocabulary of OSLC, identified by the URI **"http://open-services.net/ns/core"**, and some new subclasses have been created under the generic OSLC Classes. Hereafter, the different classes that have been extended so that any cloud service is covered and identified will be explained. For the naming convention, it would be used **oslc** as an abbreviation for referring to OSLC Core vocabulary, and all of it Classes would be named with the "vocabulary:Class" convention.

Moreover, an extension of the OSLC Action vocabulary ("http://open-services.net/ns/actions") has also been done, so it will also be explained after the Core extension. This extension will enable full compatibility with numerous types of actions on the infrastructure of choice, as it is possible to create any type of action extending the vocabulary developed on this project, and cover any use case for different applications.

3.3.1 Service Provider Catalog

Following all the core concepts of OSLC that have been explained in Section 2.2.2, a Service Provider Catalog contains Service Providers, and, in this case, three individuals have been included in the OSLC Class of the ServiceProviderCatalog, Google Cloud Storage (GCS), Google Compute Engine (GCE), and Google Kubernetes Engine (GKE), and it can be represented in rdf format as can be found in 3.1.

Listing 3.1: Service Provider Catalog RDF representation

```
@prefix dc: <http://purl.org/dc/terms/> .
@prefix oslc: <http://open-services.net/ns/core#> .

<http://localhost:5001/GCP_OSLC/service/serviceProviders/catalog>
  dc:title "Google Cloud Catalog" ;
  dc:description "Google Cloud Platform Catalog" ;
  a <http://open-services.net/ns/core#ServiceProviderCatalog> ;
  oslc:serviceProvider <http://localhost:5001/GCP_OSLC/service/
      serviceProviders/1>, <http://localhost:5001/GCP_OSLC/service/
      serviceProviders/2>, <http://localhost:5001/GCP_OSLC/service/
      serviceProviders/3> .
```

There, it represents the Google Cloud Catalog of services using DCTERMS[2] and the OSLC Core vocabulary, and is populated with the three service providers' resources of the services mentioned above.

3.3.2 Service Providers

Again, following the standard concepts, the Service Provider concept explained in Section 2.2.2 has been extended to cover the usual cloud services that are common in Big Data applications. This extension does not cover all the categories of cloud services, as there are numerous services, but it provides the basis for enabling the integration of any of the actual cloud services that cloud vendors provide. This categorization can be found in Figure 3.3.

Regarding the Service Provider Core concept and taking into account the multiple services that cloud providers offer, it has been divided into subcategories of different types of service providers, so all of the services are covered and distinguished depending on the scope of the application that is wanted to integrate. Therefore, in this project two main categories have been designed, or speaking in semantic terms, two Classes that are SubClasses of ServiceProvider Class: Computation Service Class and Data Storage Class.

[2]https://www.dublincore.org/specifications/dublin-core/dcmi-terms/

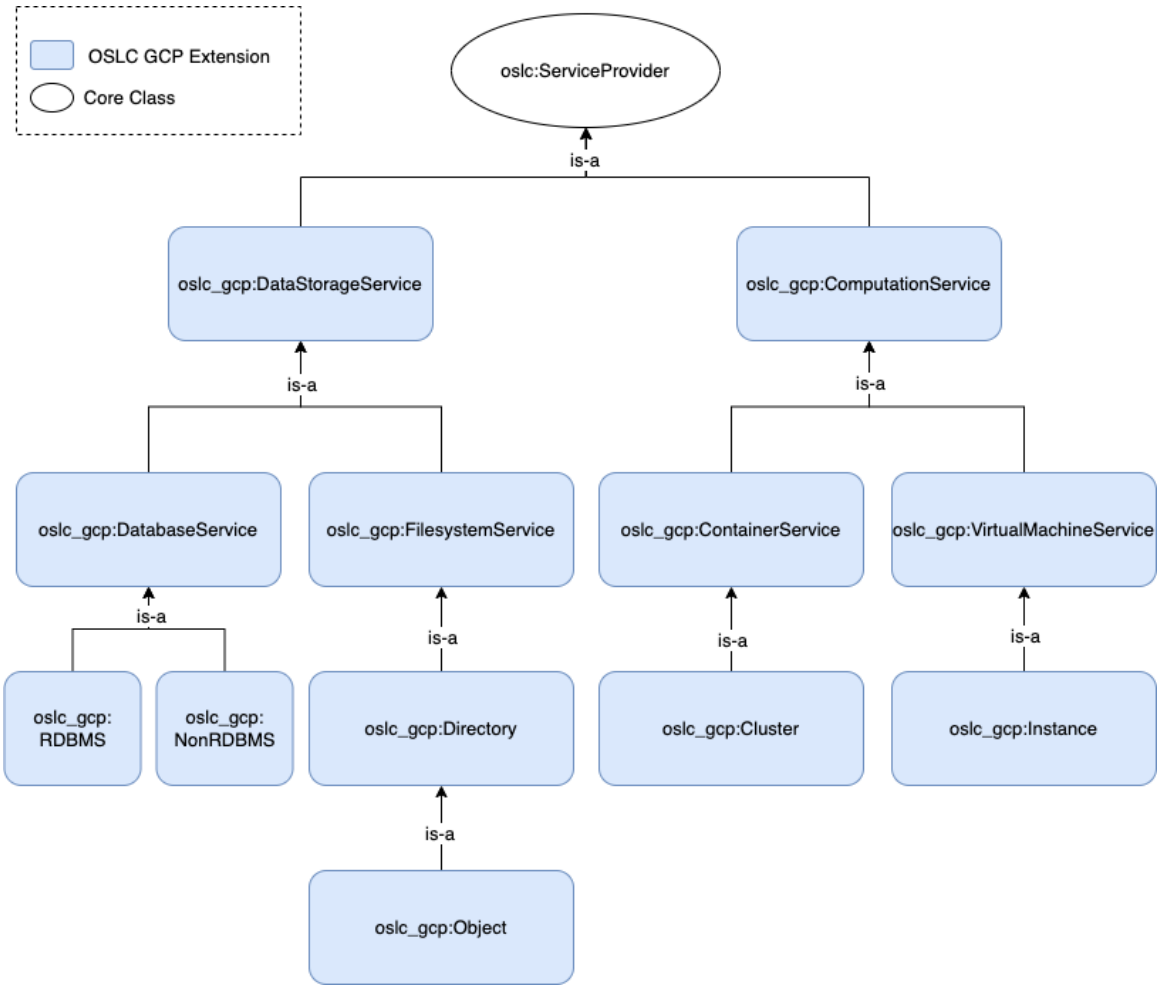


Figure 3.3: OSLC Service Provider extension

Computation Service

On the one hand, oslc:ServiceProvider Class has been divided into **ComputationService** SubClass, referring to any type of cloud service related to computation resources in the cloud. Moreover, Computation Service is subdivided into two other subclasses, **VirtualMachineService** and **ContainerService**.

Virtual Machine Service. As an individual of this SubClass, it has been included in Google Compute Engine (GCE) and, as basic unit, **Instance** SubClass is defined for a specific virtual machine instance of the service. Under this SubClass, it is also possible to integrate any kind of virtual machine service of any cloud provider as it can be the EC2 service of AWS. In Listing 3.2 an rdf representation of GCE is shown as an individual of the ComputationService Subclass.

Listing 3.2: Google Compute Engine Service Provider RDF representation

```
@prefix oslc: <http://open-services.net/ns/core#> .
@prefix oslc_gcp: <http://localhost:5001/GCP_OSLC/> .

<http://localhost:5001/service/serviceProviders/2>
  a <http://open-services.net/ns/core#ServiceProvider>, <http://localhost
      :5001/GCP_OSLC/VirtualMachineService> ;
  oslc:service [
    a oslc:Service ;
    oslc:creationFactory [
      a oslc:CreationFactory ;
      oslc:resourceType <http://localhost:5001/GCP_OSLC/Instance> ;
      oslc:label "Creation Factory" ;
      oslc:creation <http://localhost:5001/service/serviceProviders/2/
          instance>
    ] ;
    oslc:queryCapability [
      a oslc:QueryCapability ;
      oslc:resourceType <http://localhost:5001/GCP_OSLC/Instance> ;
      oslc:label "Query Capability" ;
      oslc:queryBase <http://localhost:5001/service/serviceProviders/2/
          instance>
    ]
  ] ;
  oslc_gcp:virtualMachineServiceId "ComputeEngine" ;
  oslc_gcp:virtualMachineServiceTitle "Google Compute Engine" ;
  oslc_gcp:virtualMachineServiceDescription "VirtualMachineService" .
```

There, GCE is defined with OSLC Core properties showing different capabilities as oslc:creationFactory, including the capability of creating new instances, oslc:queryCapability, as it is possible to query and retrieve the active instances, oslc:queryBase, with the URI to use for queries, or oslc:resourceType with the expected resource type URI that is returned when querying it.

To improve the description of cloud resources, some data properties and annotation properties have been added. Hence, it has been created **virtualMachineServiceId** as a data property that means the id of the virtual machine service, in this case, "ComputeEngine". Furthermore, the following annotation properties have been created: **virtualMachineServiceTitle** and **virtualMachineServiceDescription**, to provide a proper title and description of the VM service.

Listing 3.3: Virtual Machine Service Instance RDF representation

```
@prefix oslc_gcp: <http://localhost:5001/GCP_OSLC/> .
@prefix oslc: <http://open-services.net/ns/core#> .

<http://localhost:5001/GCP_OSLC/service/serviceProviders/ComputeEngine/
    instance/835202599134764705>
  oslc_gcp:instanceZone "https://www.googleapis.com/compute/v1/projects/
      weighty-time-341718/zones/us-west4-b" ;
  oslc_gcp:instanceCreationTimestamp "2022-03-30T11:24:15.415-07:00" ;
  oslc:details "RUNNING" ;
  a oslc_gcp:Instance ;
  oslc_gcp:instanceName "tfm-oslc" ;
  oslc:serviceProvider <http://localhost:5001/GCP_OSLC/service/
      serviceProviders/2> .
```

As an example of an actual resource of this class, it can be found in Listing 3.3 as an rdf representation. To better describe all the properties of this type of resource, the following data properties have been created: **instanceName**, which refers to the name of the instance, and **instanceZone**, which specifies the zone in which that instance has been created. Additionally, the annotation property **instanceCreationTimestamp** has been created to refer to the timestamp of when the instance was created.

Container Service. Google Kubernetes Engine (GKE) has been included as an individual in the Container Service SubClass of the ontology. As a basic unit of container services, a **Cluster** Subclass has been created where clusters of container services can be represented and integrated. In Listing 3.4 an rdf representation of GKE is shown as an individual of the ContainerService SubClass.

Listing 3.4: Container Service Provider RDF representation

```
@prefix oslc: <http://open-services.net/ns/core#> .
@prefix oslc_gcp: <http://localhost:5001/GCP_OSLC/> .

<http://localhost:5001/service/serviceProviders/3>
  a <http://open-services.net/ns/core#ServiceProvider>, <http://localhost
      :5001/GCP_OSLC/ContainerService> ;
  oslc:service [
    a oslc:Service ;
    oslc:creationFactory [
      a oslc:CreationFactory ;
      oslc:resourceType <http://localhost:5001/GCP_OSLC/Cluster> ;
      oslc:label "Creation Factory" ;
```

```
      oslc:creation <http://localhost:5001/service/serviceProviders/3/
          cluster>
    ] ;
    oslc:queryCapability [
      a oslc:QueryCapability ;
      oslc:resourceType <http://localhost:5001/GCP_OSLC/Cluster> ;
      oslc:label "Query Capability" ;
      oslc:queryBase <http://localhost:5001/service/serviceProviders/3/
          cluster>
    ]
  ] ;
  oslc_gcp:containerServiceId "GoogleCloudKubernetesEngine" ;
  oslc_gcp:containerServiceTitle "Google Cloud Kubernetes Engine" ;
  oslc_gcp:containerServiceDescription "ContainerService" .
```

There, it is semantically defined as an individual of a Container Service, and it is represented with OSLC Core attributes and properties, as well as other data properties and annotation properties. It has been created **containerServiceId** data property with the identification of the service, and **containerServiceTitle** and **containerServiceDescription** annotation properties to identify the service and describe it correctly.

Listing 3.5: Container Service Cluster RDF representation

```
@prefix oslc_gcp: <http://localhost:5001/GCP_OSLC/> .
@prefix oslc: <http://open-services.net/ns/core#> .

<http://localhost:5001/GCP_OSLC/service/serviceProviders/
    GoogleCloudKubernetesEngine/cluster/cluster-1>
  oslc_gcp:clusterStatus "RUNNING" ;
  oslc:details "RUNNING" ;
  oslc_gcp:clusterName "cluster-1" ;
  oslc:serviceProvider <http://localhost:5001/GCP_OSLC/service/
      serviceProviders/3> ;
  oslc_gcp:clusterMasterVersion "1.21.9-gke.1002" ;
  a oslc_gcp:Cluster .
```

In Listing 3.5, an example of a Cluster individual can be found where some of its properties are shown and provide useful information about the resource. For this type of resource, either data properties and annotation properties have been created. Regarding the data properties, the properties **clusterName** and **clusterStatus** have been created, providing information about the name of the cluster and the state of it, as shown with a "RUNNING" state. Regarding the annotation properties, the property **clusterMasterVersion** is created to provide information about the master node of the cluster version, which can be a

very useful piece of information in order to improve integration with other cluster services.

Data Storage Service

On the other hand, a SubClass **DataStorageService** has been created to integrate any type of service related to data storage, databases, or file system services. Thus, it has also been subdivided into **DatabaseService** Subclass and **FilesystemService** Subclass, providing another grade of granularity.

DatabaseService SubClass has been created to cover different database services that cloud vendors usually provide, as there is a lot of it. Due to the existence of relational and non-relational databases, another subdivision has been created in this class, Subclasses **RDBMS** and **NonRDBMS**.

FilesystemService has been created to cover different cloud vendor approaches for providing distributed file system services hosted on cloud machines. As an individual of this SubClass, it has been included in Google Cloud Storage (GCS), as can be found in the Listing 3.6.

Listing 3.6: File System Service Provider RDF representation

```
@prefix oslc: <http://open-services.net/ns/core#> .
@prefix oslc_gcp: <http://localhost:5001/GCP_OSLC/> .

<http://localhost:5001/service/serviceProviders/1>
  a <http://open-services.net/ns/core#ServiceProvider>, <http://localhost
      :5001/GCP_OSLC/FilesystemService> ;
  oslc:service [
    a oslc:Service ;
    oslc:creationFactory [
      a oslc:CreationFactory ;
      oslc:resourceType <http://localhost:5001/GCP_OSLC/Directory> ;
      oslc:label "Creation Factory" ;
      oslc:creation <http://localhost:5001/service/serviceProviders/1/
          directory>
    ] ;
    oslc:queryCapability [
      a oslc:QueryCapability ;
      oslc:resourceType <http://localhost:5001/GCP_OSLC/Directory> ;
      oslc:label "Query Capability" ;
      oslc:queryBase <http://localhost:5001/service/serviceProviders/1/
          directory>
    ]
  ] ;
  oslc_gcp:filesystemServiceId "CloudStorage" ;
```

```
  oslc_gcp:filesystemServiceTitle "Google Cloud Storage" ;
  oslc_gcp:filesystemServiceDescription "FilesystemService" .
```

In compliance with OSLC specification, as a Service Provider individual, it has CreationFactory and QueryCapability properties along with some others. In addition, a **filesystemServiceId** data property has been created to identify the resource and **filesystemServiceTitle** and **filesystemServiceDescription** annotation properties to provide more information about the title and description of the cloud service.

After researching typical file system services provided by cloud vendors and particularly in GCS, it has been concluded that they provide another grade of granularity similar to that that occurs in local storage of personal machines or in HDFS[3], so objects are stored inside a directory. In that way, it has been created **Directory** SubClass that, in GCS are named as Buckets, and **Object** SubClass of the latter, for identifying the proper objects inside directories.

Listing 3.7: File System Service Directory RDF representation

```
@prefix oslc: <http://open-services.net/ns/core#> .
@prefix oslc_gcp: <http://localhost:5001/GCP_OSLC/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://localhost:5001/GCP_OSLC/service/serviceProviders/CloudStorage/
    directory/test1_tfm_oslc>
  oslc:serviceProvider <http://localhost:5001/GCP_OSLC/service/
      serviceProviders/1> ;
  oslc:details "https://www.googleapis.com/storage/v1/b/test1_tfm_oslc" ;
  oslc_gcp:directoryId "test1_tfm_oslc" ;
  oslc_gcp:directoryName "test1_tfm_oslc" ;
  a oslc_gcp:Directory ;
  oslc_gcp:directoryLocation "US" ;
  oslc_gcp:timeCreated "2022-02-18T18:56:48.122000+00:00"^^xsd:dateTime ;
  oslc_gcp:directoryStorageClass "STANDARD" .
```

In Listing 3.7 the different properties of a Bucket resource according to the OSLC Core specification can be seen, such as oslc:ServiceProvider with the URI of the GCS Service Provider, in this case, identified with number one, and the oslc:Details property, showing the URI of the actual Google Cloud resource. Furthermore, some new data properties and annotation properties have been created, providing more detailed information about the cloud file system service. The following data properties have been created: **directoryId** and **directoryName** for identifying the cloud directory, **directoryLocation** to provide

[3]https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

information about the region where the directory is located, and **directoryStorageClass** to define the type of storage selected for the directory, as happens in GCS, with "standards, NEARLINE, COLDLINE, ARCHIVE" types, with different capabilities and different costs. As an annotation property, the **timeCreated** property has been created to specify the moment when the de resource was created.

3.3.3 Actions

Regarding the Automation OSLC domain, an extension of it has also been imported to the actual project from the "http://gsi.dit.upm.es/ontologies/ewe/ns" namespace that belongs to the GSI department from ETSIT University. Furthermore, these domains have been extended under the "http://open-services.net/ns/actionsAction" class, in order to provide different types of action that can be done on the actual OSLC adapter. This extension can be seen in Figure 3.4.

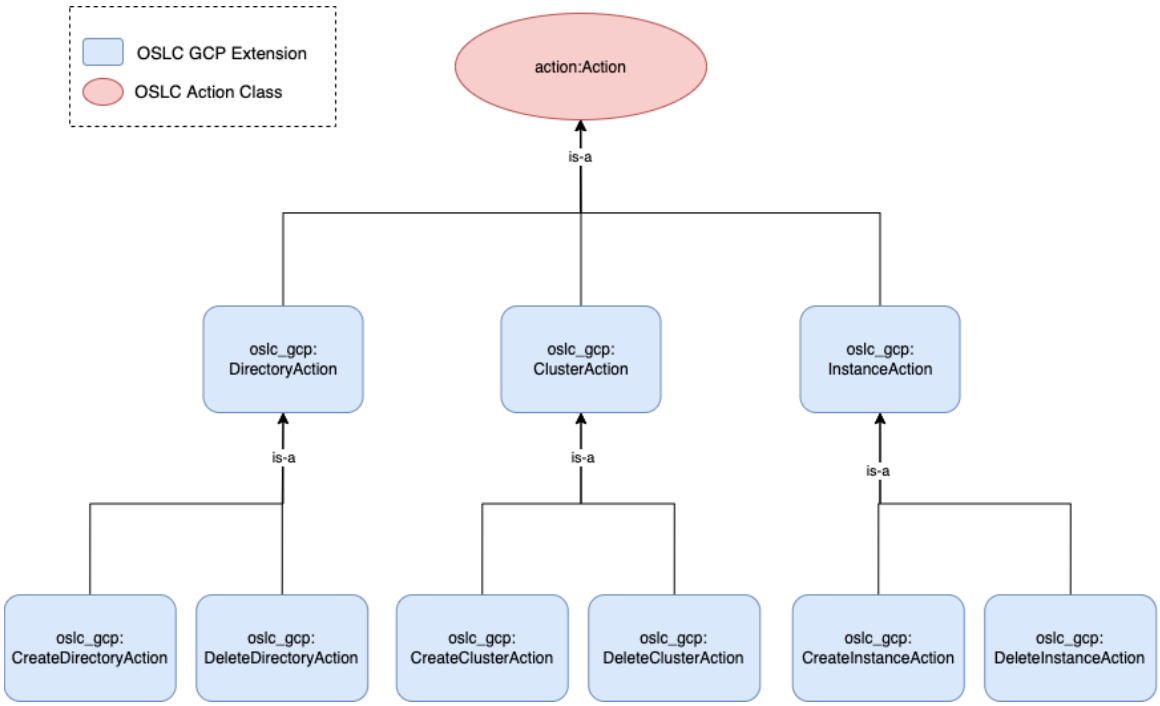


Figure 3.4: OSLC Action extension

There, the different actions that have been created can be seen, which extends the oslc Action vocabulary, in order to categorize multiple actions that can be done over cloud providers. They are basic actions to interact with the three Service Providers implemented in this project. Therefore, subclasses **DirectoryAction**, **ClusterAction**, and **InstanceAction** have been created. In Listing 3.8 an example of an action resource of DirectoryAction SubClass is shown.

Listing 3.8: Directory Action RDF representation

```
@prefix action: <http://open-services.net/ns/actions#> .

<http://localhost:5001/GCP_OSLC/action/1>
  a "http://localhost:5001/GCP_OSLC/CreateDirectoryAction", <http://open-
      services.net/ns/actions#Action> ;
  action:actionResult "OK" ;
  action:actionProvider "1" .

<http://localhost:5001/GCP_OSLC/action/2>
  a "http://localhost:5001/GCP_OSLC/DeleteDirectoryAction", action:Action ;
  action:actionProvider "1" .
```

As can be seen above, no creation of additional properties has been needed to define these resources. The oslc-actions:actionResult property has been used for the result of the desired action, and the oslc-actions:actionProvider property has been used to indicate the ServiceProvider on which the action is done. The type of rdf resource that, in this case, contains two of them, the CreateDirectoryAction and DeleteDirectoryAction SubClasses, is also indicated.

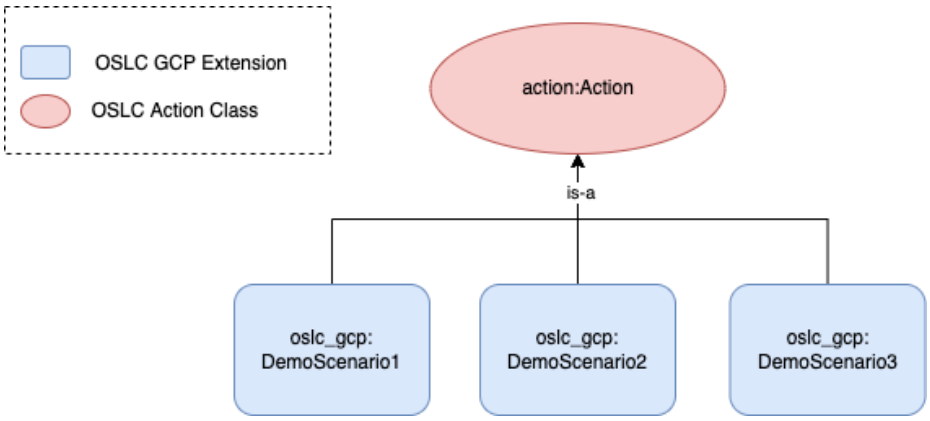


Figure 3.5: OSLC Action Demo Scenarios

In addition, demo scenarios have been created in order to deploy different scenarios that could be matched to a real environment as it can be in production purpose. All of these scenarios are extended from the oslc_actions:Action Class, as it can be seen in Figure 3.5.

3.4 System Architecture

The developed system is made up of different modules that are seamlessly integrated to work with each other. This complete architecture is an implementation of an OSLC Adapter that can be shown in Figure 3.6 and is integrated with the global architecture of Figure 3.2. The core concept of the module is the exchange between actions and events between cloud services and servers. Therefore, the global module will expose Events in response to external actions.

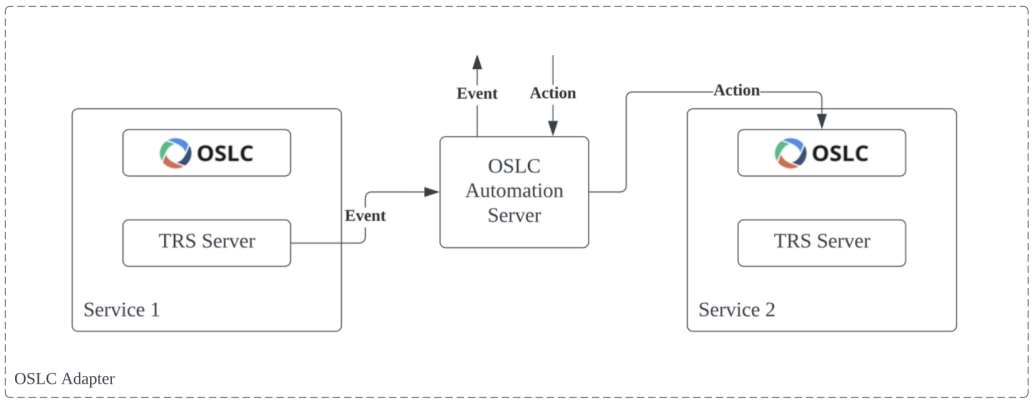


Figure 3.6: OSLC Adapter Generic Architecture

More in depth, the actual implementation of the system architecture can be found in figure 3.7, where there are different modules that will be detailed below. From left to right are found the User Interface, the OSLC Adapter, hosted on a GCE VM, Google Cloud Platform and its Cloud Pub/Sub Service, a Kafka Container, and finally, DuckDNS Server. All of these modules are described below.

3.4.1 User Interface

As the global system is done following RESTful APIs concepts, different endpoints are exposed from the outside in order to access the server that contains the OSLC adapter. Users can interact with the system using multiple methods, but it has been done using the Insomnia REST API Client. Therefore, the client is used to build HTTP petitions to the different endpoints to interact with the semantic resources stored on the server.

As shown in Figure 3.7, users can send through HTTP POST petitions actions against the OSLC adapter by sending in a correct rdf format (in this case, the rdf/xml format has been used) the body of the petition including the resource to create, update, or delete followed by the type of action to be taken. In addition, the type of content should be

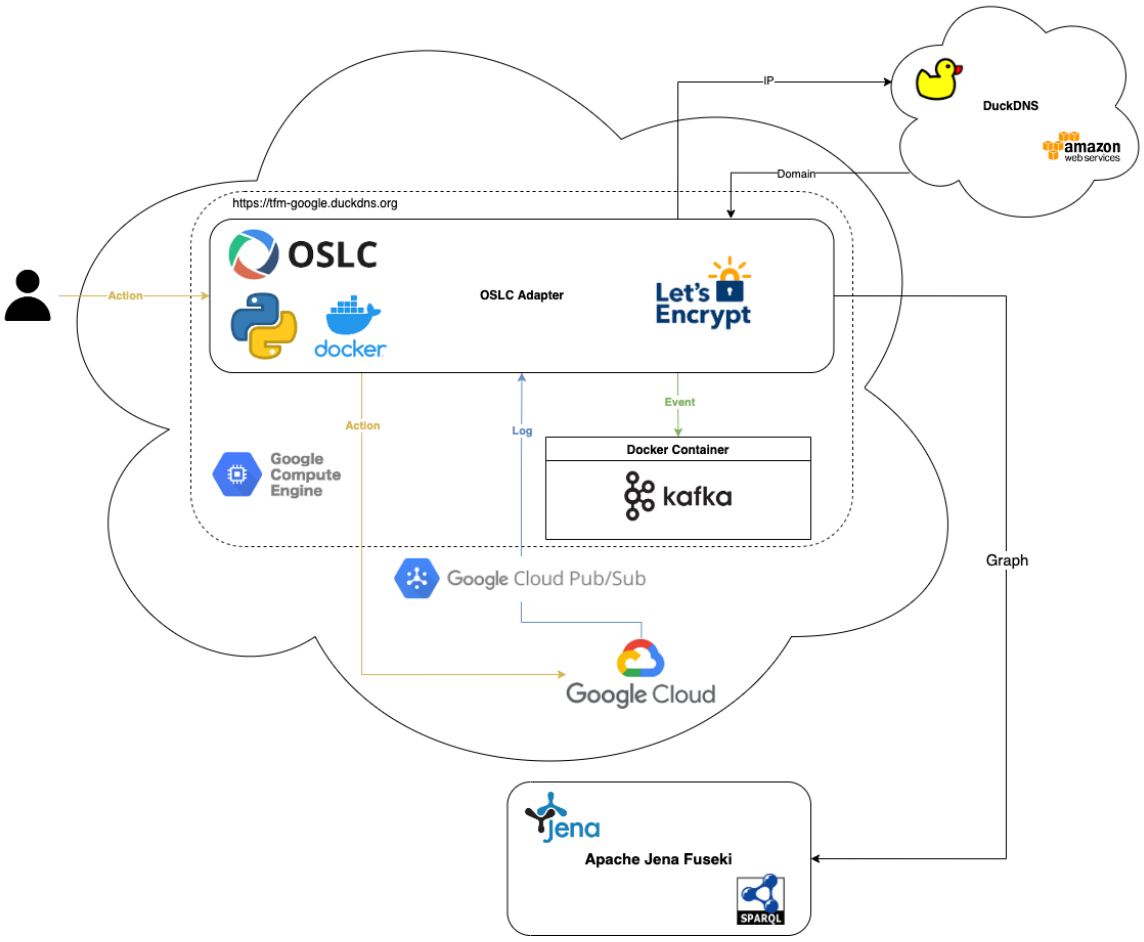


Figure 3.7: OSLC Adapter Architecture

indicated for the correct deserialization of the information.

Regarding the architecture of the global project of Figure 3.2, in a production environment, these actions will be carried out automatically directly by EWETasker in response to certain events when some rules are met. As this system has been developed isolated from the global system, EWETasker Actions has been replicated by a manual interaction of a user in order to test the system.

3.4.2 OSLC Adapter

The OSLC Adapter is made up of different docker containers that provide the necessary services to achieve the complete functionality of the adapter. It is made up of three containers, which will be explained in the following subsections. In summary, the adapter contains a Zookeeper container, a Kafka container for messaging, and a Web container to provide the API functionality of the adapter.

First, **Web container** is a Flask server that has been developed to run on a local machine or a virtual machine hosted on the GCE service on Google Cloud. At first, it was developed locally, but for reasons that will be explained later, it was hosted on a GCE instance on the cloud.

Thus, the adapter is running on a virtual machine hosted on GCP and managed by the GCE service. It is an instance running a Debian-10 image where all the needed software is installed for the adapter. Moreover, the instance is located on the "us-west4-b" region and its firewall is configured with specific traffic rules in order to allow all traffic from any source and ports 80, 5001 and 443. This enables communication via HTTPS from any source on the Internet.

Internally, the adapter is composed of a Docker container that is run in order to start the service. This container has a custom Python image and is used for launching the Flask server and exposing an internal server port so it can be accessed from the Internet, specifically, it is used 5001 port. Therefore, the server is identified by the public URL "https://tfm-google.duckdns.org:5001", so, in order to send petitions, it is needed to point to this URL.

Additionally, within the VM instance, other containers run. Regarding the integration with a distributed event-based messaging platform, **Zookeeper** and **Kafka** are deployed on the machine. The first is used for controller election and cluster membership, despite the unique Kafka broker of the architecture. Other main function that this container has is the topic configuration, since it maintains the configuration of all topics, including the list of existing topics, number of partitions for each topic, and location of replicas.

Kafka container is used to enable the creation of a Kafka producer given the URL and port of the container, and publish Event messages into the event topic, so that it provides the OSLC Adapter, the capabilities to interact with external blocks of the global architecture of SmartDevOps project.

Interaction with GCP

The adapter receives the Action from the user in an HTTP POST form, it deserializes it, and applies internal logic to extract the information. This information is interpreted by the adapter and then triggers a specific action in GCP that can be the creation, modification, or deletion of resources.

Regarding the communication between the adapter and GCP, it is done via API, so after the adapter receives the information and extracts all of the resource details, it uses them to point to a specific Google API to execute the desired action and create the cloud resource.

On the GCP side, it receives an HTTP petition under its APIs, performs the action,

and internally generates logs regarding the resource updates on the platform. To receive these logs, the Google Pub/Sub service is used by creating a specific topic to which the logs will be sent.

These logs are configured to be sent to this topic using the Google Cloud Logging service, which is a simple service that stores all the log information of any changes on the platform. Therefore, the topic is configured as a Log Sink.

Finally, it is configured a Pub/Sub Push Subscription that automatically routes all the log information of Cloud Logging to a specific endpoint, which in this case, is the URL of the OSLC Adapter server, which needs to be publicly accessible and running securely over HTTPS, so a valid SSL certificate is needed to be signed by a Certificate Authority. The OSLC Adapter receives the log in JSON format and after passing through some logic, it updates its semantic resources in order to be synchronized with Google Cloud resources.

Interaction with the Event Server

As a first state of implementation, a simple Flask server has been developed that runs on the same virtual machine as the OSLC Adapter, but instead runs over a different port, the 5002 port. Unlike the adapter, this server is a simple listener of OSLC Events, and its functionality is reduced to listening to the events without any processing.

After the execution of the desired user action on GCP and the resource update in the adapter, the adapter generates an Event resource depending on the action that has been done and it is sent to the Event Server. From the adapter point of view, it is capable of receiving Actions and Logs, and sending Events to an endpoint that, in this project, a simple server has been used but it is possible to integrate any messaging service as Kafka topics to interact with other services.

Interaction with Kafka

In a second stage of implementation and with regard to the complete architecture of SmartDevOps project, a distributed event-based messaging platform based on a Kafka container along with a Zookeeper container has been developed to provide distributed coordination of the messaging cluster of workers.

Kafka is used to create an event message topic and send all interactions with the adapter in the form of events that are generated in response to different actions performed.

When an action is received by the adapter from an external source, it is processed internally, and it performs several actions on the Google Cloud Platform. When finished, an Event resource is generated and integrated with the TRS and is sent to the Kafka topic in json format.

Interaction with DuckDNS

Regarding the public reachability of the server, DuckDNS is used as a dynamic DNS hosted on AWS. Provides a dynamic domain for the server, which is "https://tfm-google.duckdns.org" and enables the dynamic mapping of the domain to the IP address of the server, i.e. the public IP address of the GCE instance on GCP.

Internally, the server needs to be certificated by a Certificated Authority, Let's Encrypt, and in order to sign and generate the certificates, Dehydrated code is used.

The form in which the adapter interacts with this service is by calling a specific endpoint of DuckDNS to update the server IP in case it has been modified by the service provider. In response to this call, DuckDNS reconfigures the domain that was created for the project, to modify the new IP, and continues to work as normal. Therefore, the service will remain operational and publicly accessible despite any change in IP.

CHAPTER 4

Implementation

This chapter contains the details of the followed process for the implementation of the described architecture and a demo scenario is introduced in order to explain how the system is tested.

4.1 Introduction

With respect to the architecture of the system previously described, in this chapter, the process of how the system is implemented and the steps followed will be described. Therefore, it will include a detailed description of the implementation of the OSLC Adapter, regarding internal Python modules developed as the actual server, the different data models, and the API module which contain the core functionality.

Additionally, the process of implementing a distributed event-based system in order to integrate the adapter seamlessly with the global architecture of the project, involving the sending of event resources, will be explained. Finally, a demo scenario will be shown with the objective of showing all of the capabilities of the system.

4.2 HTTPS server

First, a Flask server has been developed to provide a web application to make HTTP petitions from the Internet and, therefore, expose a publicly accessible API to deal with cloud resources.

The server runs over a Docker container based on a custom python image of the 3.10 version, on which a set of required libraries are installed collected in the "requirements.txt" file. Along with this file, other files are copied into the container, such as Google Cloud credentials in JSON format that need to be previously generated through the Google Cloud Console in order to deal with it API, as well as certificate files that allow the Flask server to run over HTTPS. The generation of these files will be explained in Section 4.7.

The server is deployed by performing a *docker-compose up* command, uses the *oslcapi* image with the Python version and all installed libraries, sets environmental variables as the URI of the database and the path of Google credentials, and executes the *flask run* command. This docker-compose file can be found in Listing 4.1.

Listing 4.1: Docker-compose file

```
services:
  web:
    image: oslcapi
    build: .
    command: flask run -h 0.0.0.0 --cert=/code/certs/cert.pem --key=/code/
        certs/privkey.pem
    env_file:
      - ./.flaskenv
    environment:
```

```
      - DATABASE_URI=sqlite:////db/oslcapi.db
      - GOOGLE_APPLICATION_CREDENTIALS=/code/gcp/gcp_credentials.json
    volumes:
      - ./oslcapi:/code/oslcapi
      - ./db/:/db/
      - ./gcp_credentials.json:/code/gcp/gcp_credentials.json
      - ./cert.pem:/code/certs/cert.pem
      - ./privkey.pem:/code/certs/privkey.pem
    ports:
      - "5001:5000"
```

This code is part of the complete *docker-compose.yml* file, where other containers are launched. Here, it can be seen that it will create a container named **web**, with an image of *oslcapi*, which can be seen in the Listing 4.2. At startup, a *flask run* command is executed, which indicates the IP where the server will run, which, in this case, is on localhost, as well as the necessary certificates to enable HTTPS on the server.

The server also provides an SQLite database to store all the semantic resource information in memory, so while running, the information can be queried with SPARQL queries.

Listing 4.2: Dockerfile code of oslcapi image

```
FROM python:3.10

RUN mkdir /code
WORKDIR /code

COPY requirements.txt setup.py tox.ini gcp_credentials.json cert.pem /
    privkey.pem ./
RUN pip install -U pip
RUN pip install -r requirements.txt
RUN pip install -e .

COPY oslcapi oslcapi/
COPY migrations migrations/

EXPOSE 5000
```

All adapter code is stored in the */code/oslcapi* path, which is copied from the oslcapi custom image. As can be seen in the Dockerfile, some files are copied from the active directory to the image, such as the requirements file, GCP credentials in json format, or certificates *.pem* files. Then, all the libraries contained in the requirement file that are needed to run the image are installed by executing the *pip install* command. Finally, the 5000 port is exposed, which is assigned to the 5001 port, as can be seen in the code 4.1.

4.3 Distributed Event-based Messaging Platform

With the objective of accomplishing the integration of the developed OSLC Adapter module with the general solution of the SmartDevOps project, a distributed event-based Messaging Platform has been developed. The principal motivator of this module is the integration with a system that interacts in a distributed way with messages. The selected framework for achieving this requirement has been Kafka.

For deploying Kafka, a Docker container within the docker-compose file has been used, as well as a Zookeeper container. Both containers are based on images "confluentinc/cp-kafka" and "confluentinc/cp-kafka", respectively, as can be seen in Listing 4.3, where a portion of the docker-compose.yml file is shown.

Listing 4.3: Kafka and Zookeeper containers in Docker-compose file

```
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    ports:
      - 22181:2181

  kafka:
    image: confluentinc/cp-kafka:latest
    depends_on:
      - zookeeper
    ports:
      - 29092:29092
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092,PLAINTEXT_HOST://
          localhost:29092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,
          PLAINTEXT_HOST:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
```

As Zookeeper is necessary for Kafka, as it provides, controller election, configuration of topics, access control lists, and membership of the cluster, a container has also been deployed. This container is using confluentinc image in its latest version and, as en-

vironment variables, "ZOOKEEPER_CLIENT_PORT" has been set to port 2181, and "ZOOKEEPER_TICK_TIME" has been set to value 2000. Both variables are necessary for Zookeeper configuration, being the port on which Zookeeper service will be accessible and the basic time unit in milliseconds used by Zookeeper, used to do heartbeats and to set the minimum session timeout (it is twice the tickTime value). Additionally, the internal port on which the Zookeeper service is accessible, port 2181, is mapped to the external port 22181 of the virtual machine.

On the other hand, a Kafka container is deployed using again a confluentinc image in its latest version of Apache Kafka. As Kafka needs Zookeeper to be launched and running, a *depends_on* sentence is used in the docker-compose, by setting a dependence on Zookeeper container that needs to be up until the kafka container is launched.

As configuration variables, a set of environment values has been set for the Kafka container. "KAFKA_ZOOKEEPER_CONNECT" variable is set to link Kafka to Zookeeper, by indicating *zookeeper:2181*, Kafka knows how to get in touch with Zookeeper. The variable "KAFKA_ADVERTISED_LISTENERS" describes how clients can reach the advertised host name. Finally, with "KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR" variable, a single-node cluster is set as it is not necessary to add more nodes to the cluster. The port on which the Kafka service will be accessible is port 29092 internally or externally.

4.4 Data models

4.4.1 OSLC Module

Regarding the data model, since the server stores RDF triples within a graph, a Python mapping has been performed according to the semantic model explained in Section 3.3. Therefore, the first task was to import the generated ontology into the project using the *parse* method of the rdflib Python library. Using this method, it is possible to import the ontology that indicates the format, which in this case is XML.

In this module, five main classes have been developed to correctly map all OSLC Core resources with Python resources. An UML diagram of all classes can be found in Figure 4.1.

In general terms, the UML shows the principal class, OSLCStore, where all resources are initialized. This class contains one type of class per service. Thus, it contains a FilesystemService class, as well as VirtualMachineService and ContainerService classes.

Moreover, the OSLCStore class contains a ServiceProviderCatalog class, which contains all service providers, as well as all actions done. Therefore, it contains n Action classes, one per action performed, and n ServiceProvider classes, one per each service provider

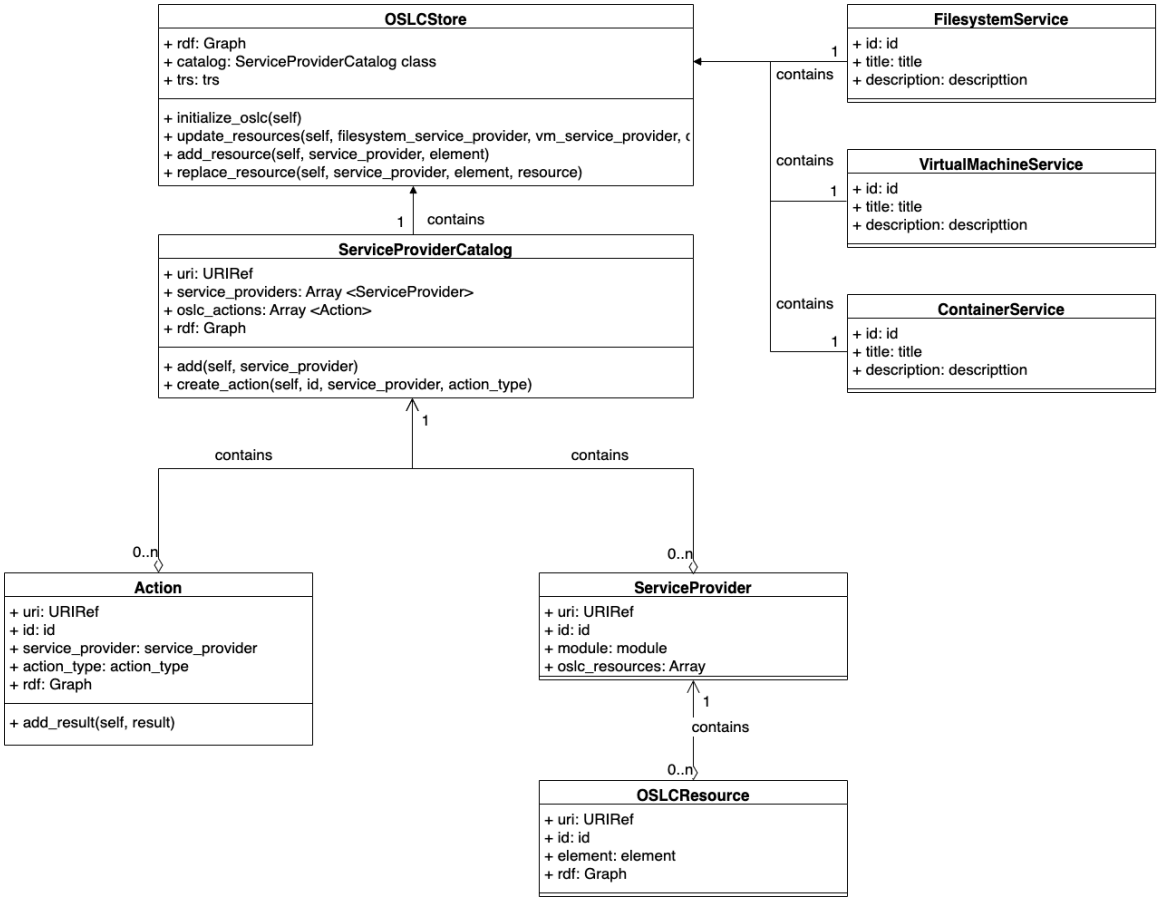


Figure 4.1: UML Diagram of the OSLC Module

that, in this case, it has been developed for the three Google Cloud services that belong to a Filesystem Service, Google Cloud Storage, a VirtualMachineService, Google Compute Engine, and finally a ContainerService, as Google Kubernetes Engine.

Finally, the ServiceProvider class has a one-to-many relationship with the OSLCResource class, since it contains as many resources as exists within each of the service providers. In that way, the GCS service provider will contain different buckets as OSLCResource, GCE will contain the different active instances that are running, and GKE will contain the running clusters as OSLCResource classes.

OSLCStore

At first, the *OSLCStore* class was defined with the actual service provider catalog and the set of tracking resources. To populate and add all the resources of the service provider with all active resources, the function *initialize_oslc()* has been developed.

In this function, OSLC resources are initialized. Each of the three Service Providers is defined and added to the list of catalogs stored in the Service Provider Catalog class,

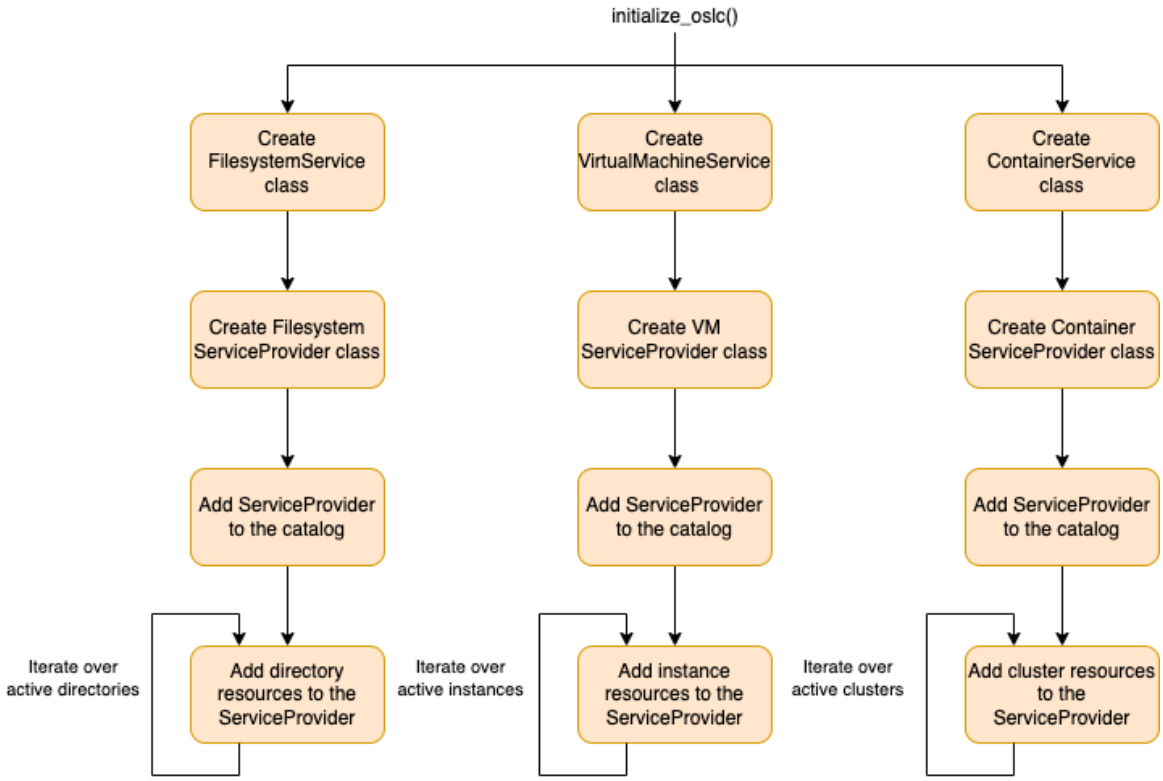


Figure 4.2: initialize_oslc function

which will be explained later. As shown in Figure 4.2, a *cloudStorage* object of class *FilesystemService*, which includes three parameters of ID, title and description, is created, then added as a *ServiceProvider* class, and is included in the list of catalog service providers. In the same way, the same actions are performed with the rest of the service providers, with the VMs and the clusters.

Moreover, the Google API is used to retrieve active buckets on the platform using the *list_buckets()* function, and then they are added to the Service Provider created before, as an *OSLCResource* object, which will be detailed later.

This operation is repeated for each of the Service Providers with the difference of the API code used to retrieve different resources from GCP, as it is retrieving either GCS Buckets, GCE Instances, and GKE Clusters.

Furthermore, a *update_resources* function has been developed to update locally stored server resources to be up-to-date and correlated with real active GCP resources. This function is called from the Views module in Section 4.5.1, when a log message from Google Cloud is received from the server.

The function will first check which of the input parameters that are each of the three service providers is None, so it will only update the Service Provider of choice depending

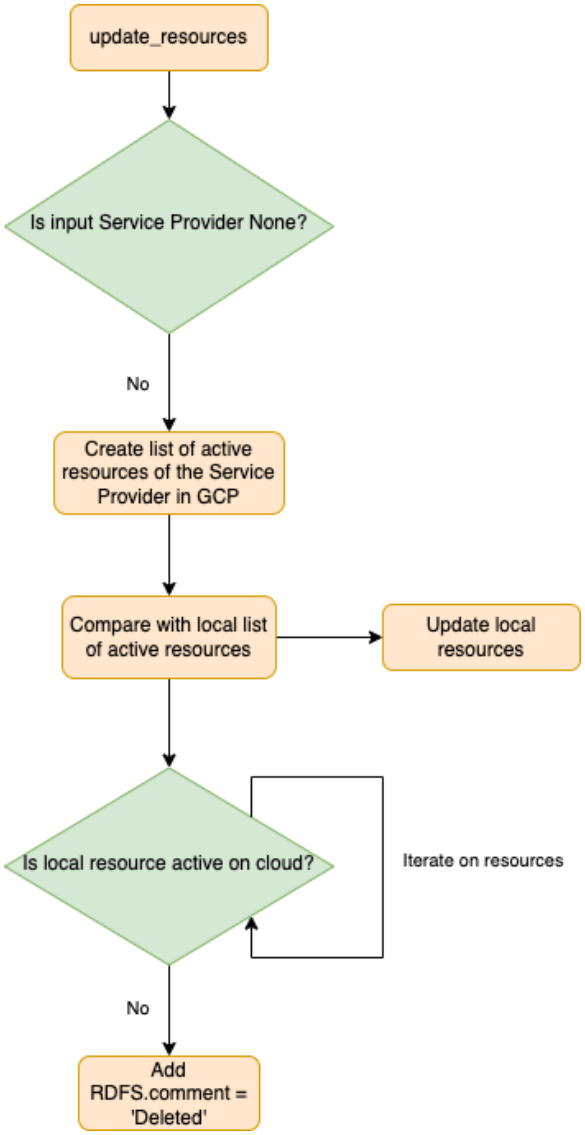


Figure 4.3: update_resources function

on the situation.

Moreover, it will list the active resources of each Service Provider in GCP and create a list for each one. Then, it will compare it with the list of resources of the server, so if there is one missing, it will mark it as deleted with the property *RDFS.comment* and the literal 'Deleted'.

Again, this operation is repeated for each of the Service Providers, so it provides a real status of the active resources on GCP and maintains a synchronization between the OSLC Adapter and GCP resources.

ServiceProviderCatalog

As can be seen in the UML diagram in Figure 4.1, this class has a 1 to 1 relationship with the OSLCStore class and contains multiple Action classes and multiple ServiceProvider classes, in this case, three of them.

ServiceProviderCatalog class is defined with a URI formed with a base URL concatenated with "/service/serviceProviders/catalog" as defined in the OSLC Core specification. This class also contains an array of ServiceProvider class types, which are the three providers that have been explained. It also contains an array of Action class type, with the different actions that are performed through the OSLC Adapter.

Provides a *add* function to add service providers to the catalog and a *create_action* function that generates an Action class and adds it to the array of actions, so that all actions can be listed.

ServiceProvider

The objects of this class are defined with a URI formed of a base URL plus "/service/serviceProviders/" and concatenated with the Service Provider ID previously generated at the initialization of the OSLCStore class. This class contains the necessary parameters for defining an OSLC Service Provider resource, as well as a list of resources for each of the Service Providers.

It is also called a function to map all defined semantic attributes with the actual attributes of the real GCP resources, as can be found in Figure 4.4.

Therefore, depending on the service provider, the function will map different semantic properties to the actual services. For example, for the case of GCS, it will add its ID as an integer that identifies the service provider, a title, which in this case is Google Cloud Storage, and a description of this service. Similarly, it will do the same for the rest of the service providers.

OSLCResource

This class has been created to define the basic OSLC resources contained in each service provider. They are identified with an ID and an URI, but, regarding the implementation of three Service Providers, it is needed to differentiate the resources of each of them as they have different properties. Therefore, it is necessary to pass as a parameter the real element returned by the Google API so that its attributes are read and included as a different OSLC resource. This logic can be found in Figure 4.5.

There, it is shown how, depending on the type of element that is being included as an OSLC resource, different properties are associated with the resource, as well as different URIs, since they are accessible through different endpoints. Finally, by calling the function

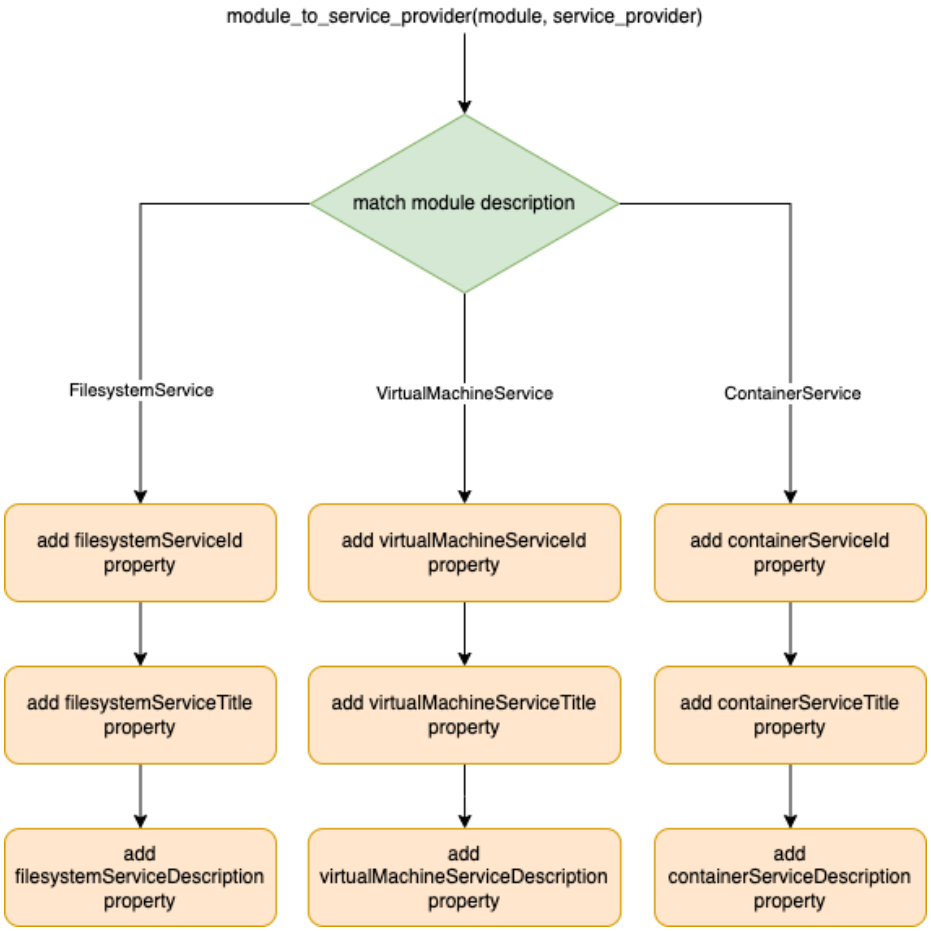


Figure 4.4: module_to_service_provider function

element_to_oslc_resource, the mapping of Python objects and OSLC semantic resources is performed.

As can be seen in Figure 4.6, depending on the type of element to be mapped, the function adds different properties to the OSLC resource. In addition, the different fields of the real resource are taken into account, depending on the API. In this case, it has been figured that GCS buckets have multiple properties that can be extracted from its API. Therefore, different properties have been created in the ontology according to the fields that make up the buckets, instances, and clusters.

Action

Finally, the Action class is created with the main attributes of the Service Provider to which it is pointing, the type of action, and the action result, as can be seen in the UML diagram of Figure 4.1.

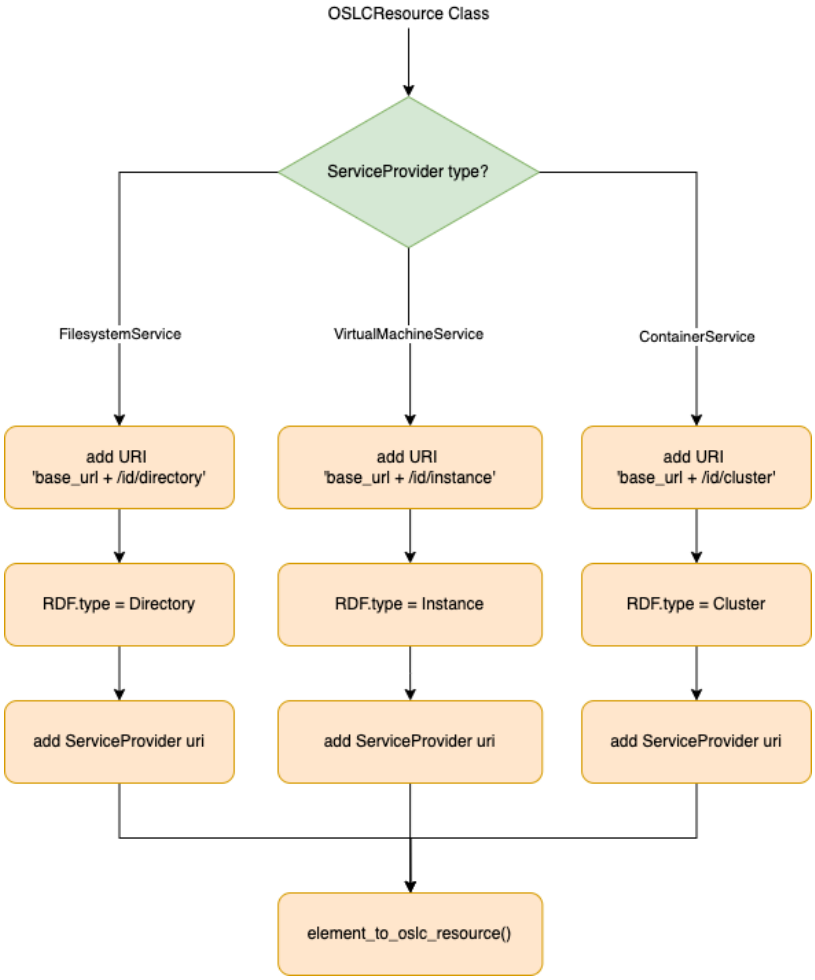


Figure 4.5: OSLCResource class

4.4.2 TRS Module

This module contains all the resource definitions and the necessary functions for the Tracked Resource Set (TRS). Similarly to the *oslc.py* module, a *TRSStore* class is defined and identified by the URI formed by the concatenation of a base URL and the route "/service/-trackedResourceSet". In this endpoint, TRS information can be retrieved as part of the developed API.

This class contains a list of change logs, as well as a list of change events that conform to the TRS, complying with the OSLC TRS specification. As part of the class, it also includes a function called *generate_change_event* which takes as parameters the resource on which the change is made and the action executed and generates the corresponding TRS semantic resources, including them in the TRS class.

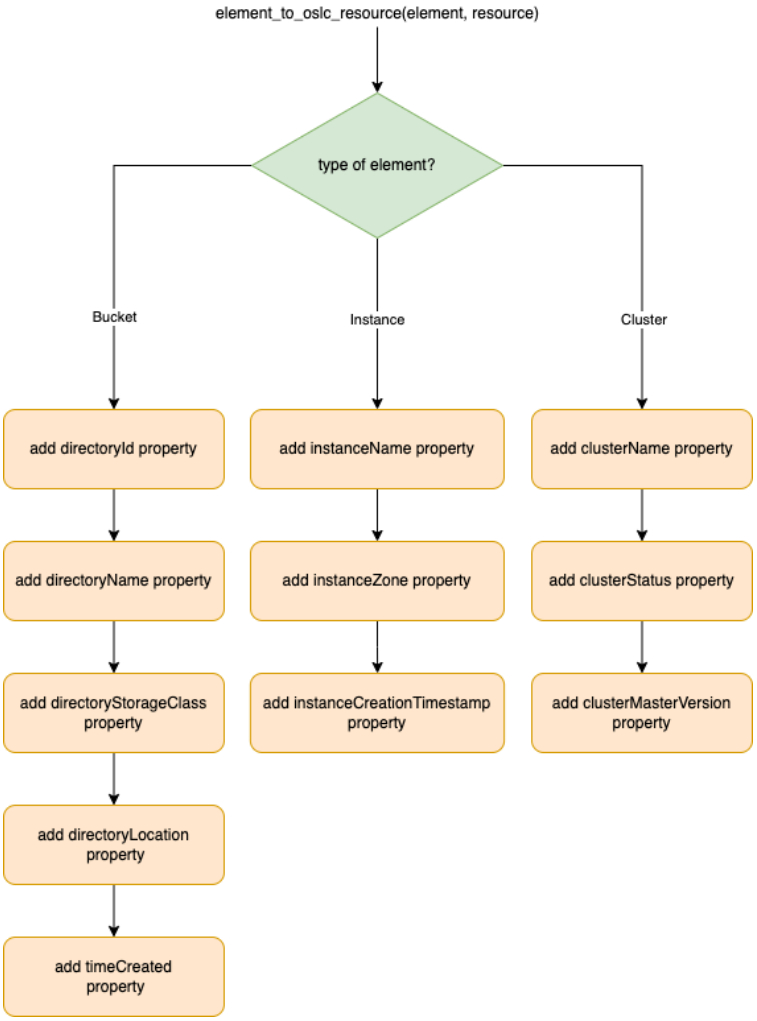


Figure 4.6: element_to_oslc_resource function

4.5 API module

4.5.1 Views

At this point, all Python modules regarding the data models have been explained. The next step in understanding the development of the adapter is to explain the *Views* module. This module mainly contains flask blueprints[1]. Blueprints record operations to execute when registered in an application. They trigger some functions over an endpoint, depending on the type of HTTP request that has been done.

In this way, this module contains a definition of different endpoints to which it is possible to make an HTTP request, and a specific function is triggered. In Figure 4.7, all exposed endpoints can be seen, which provide different functionality from the OSLC adapter in the

[1]https://flask.palletsprojects.com/en/2.1.x/blueprints/

form of an API.

Here, different endpoint definitions can be found. In order to list all of the service providers available, an HTTP GET petition can be made over the "/catalog" endpoint. To retrieve some information about every service provider, the ID can be taken in this list, and then an HTTP GET petition can be sent to the "/serviceProviders/id" endpoint with this ID.

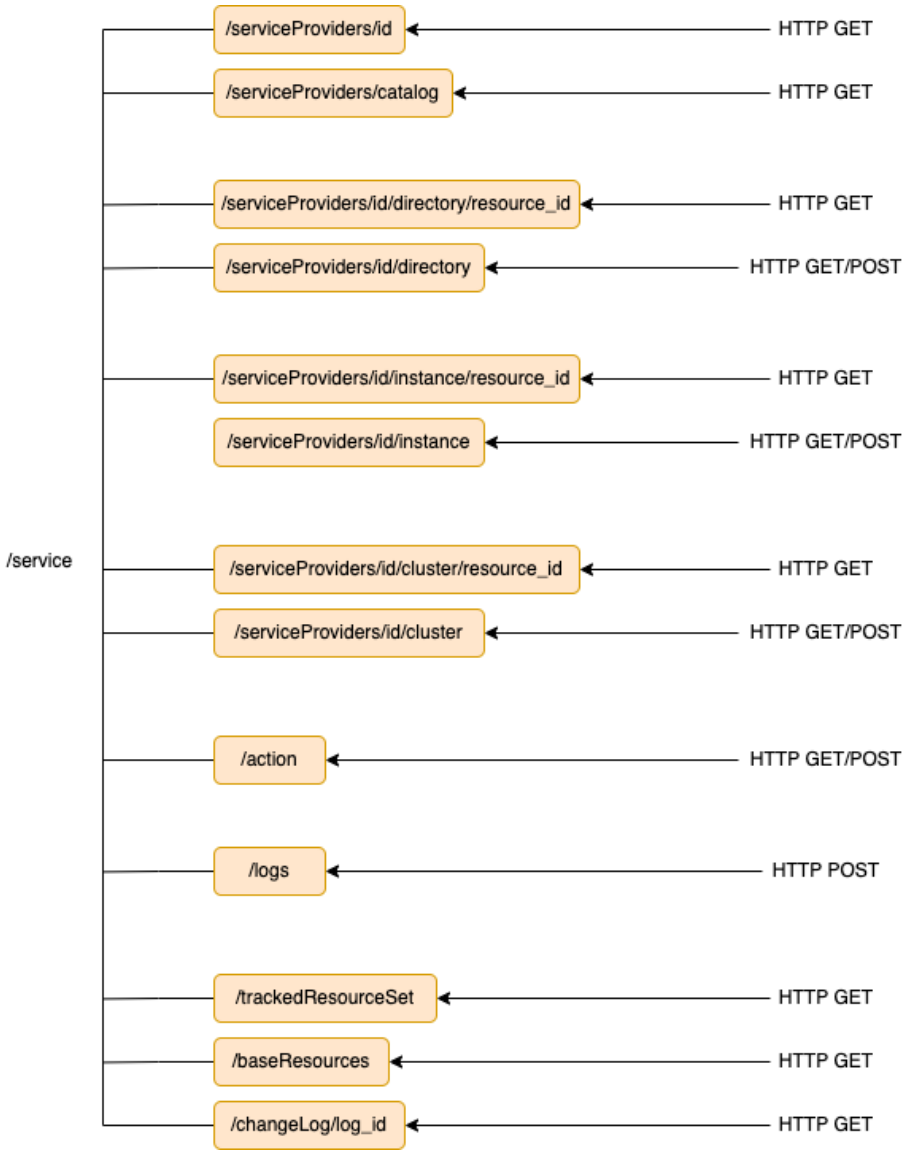


Figure 4.7: Flask endpoints

In case the list active directories want to be retrieved, it is possible to make an HTTP GET petition to the "/serviceProviders/id/directory" endpoint that will respond with a list of directories. If the creation of a directory is desired, it is possible to send an HTTP POST petition to this endpoint, sending a proper body with the correct directory attributes. Addi-

tionally, it is possible to retrieve specific information from a directory resource by making an HTTP GET petition to the "/serviceProviders/id/directory/resource_id" endpoint, given a valid id for the resource.

Similarly, the same petitions can be done to virtual machine instances or clusters by using their respective endpoints.

As additional information, initially an endpoint was developed for each resource of each Service Provider for testing purposes. As a development process, when all functionality of each endpoint was met, all these endpoints were removed leaving only the *"/action"* endpoint and the *"/logs"* endpoint to achieve the desired system operation, as shown in Figure 3.7.

Therefore, by sending HTTP petitions to the "/action" endpoint, all previous operations can be done over the different service providers, so it is not necessary to point to each of the service providers endpoints. The adapter itself will determine to which of the service providers is the action performed.

As will be explained later, the "/logs" endpoint is available by making HTTP POST petitions to enable the communication between GCP and the adapter. Finally, some endpoints regarding the Tracked Resource Set are available to list the resources by making HTTP GET petitions to the different endpoints.

4.5.2 Resources

In module *api.resources*, the *resourceOSLC.py* file can be found and contains all the triggered functions that are called from the different endpoints. As mentioned before, the development has been done taking into account two possible ways to interact with the system: by posting an action from an external system and, internally, sending logs to the adapter in order to update the internally stored resources. Both are explained below.

Action Endpoint

As the way to interact with the adapter is through actions, when an HTTP request is made over the *"/action"* endpoint, it triggers the ***OSLCAction*** class, which contains *get* and *post* functions in response to HTTP GET and HTTP POST requests.

Therefore, when an HTTP GET request is made over this endpoint, an rdf graph is returned with all of the Actions resources that have been created. When an HTTP POST request is made, an rdf/xml formatted graph is sent to the endpoint, so it is necessary to interpret that resource.

First, a simple SPARQL query is defined to retrieve the type of action that is being sent. This query can be found in 4.4.

Listing 4.4: SPARQL Query for Action resources

```
query_action = """
                PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
                PREFIX action: <http://open-services.net/ns/actions#>

                SELECT ?type

                WHERE {
                    ?s rdf:type ?type .
                }
            """
```

Then, the rdf graph is parsed, and the Action object is created depending on the type and the Service Provider on which it is done. As can be seen in 4.8, the result of the query is checked to find the type of resource from which the action is performed, it retrieves the Service Provider ("actionProvider") on which the resource is contained, and finally, the action resource is created.

Finally, once the action resource is created, the function evaluates whether it is a creation action or a deletion action, as shown in 4.8.

If a creation action is sent, the creation of the resource is triggered, and the result of the action is added to the resource action. In addition, it triggers the *generate_creation_event* function that generates the proper event and integrates it with the TRS to finally generate a json-formatted Event and sends it to a **Kafka topic**, ready to be consumed by a Kafka consumer.

Otherwise, if a deletion action is sent, the *delete_resource* function is activated, as well as *generate_deletion_event* to record the event in the TRS. Finally, the event is sent in json format to a Kafka topic, ready to be consumed by a Kafka consumer.

Log Endpoint

As can be seen in figure 3.2, GCP is communicated with the OSLC adapter by sending logs with changes on the platform. This communication will be explained later in Section 4.5.3. Focusing on the code, when an HTTP POST request is sent to the *"/logs"* endpoint, it triggers the *post()* function of the *GCPLogs* class contained in this module.

This function receives the resource that is sent in the source service format that, in this case, is a JSON format, since the sender is Google Cloud Logging service. This format can be seen in 4.5, which shows an extraction of the message sent by Google.

In this json body, several fields can be identified in order to retrieve rich information about the exact action that has been done on GCP. As can be seen, it includes a "logName"

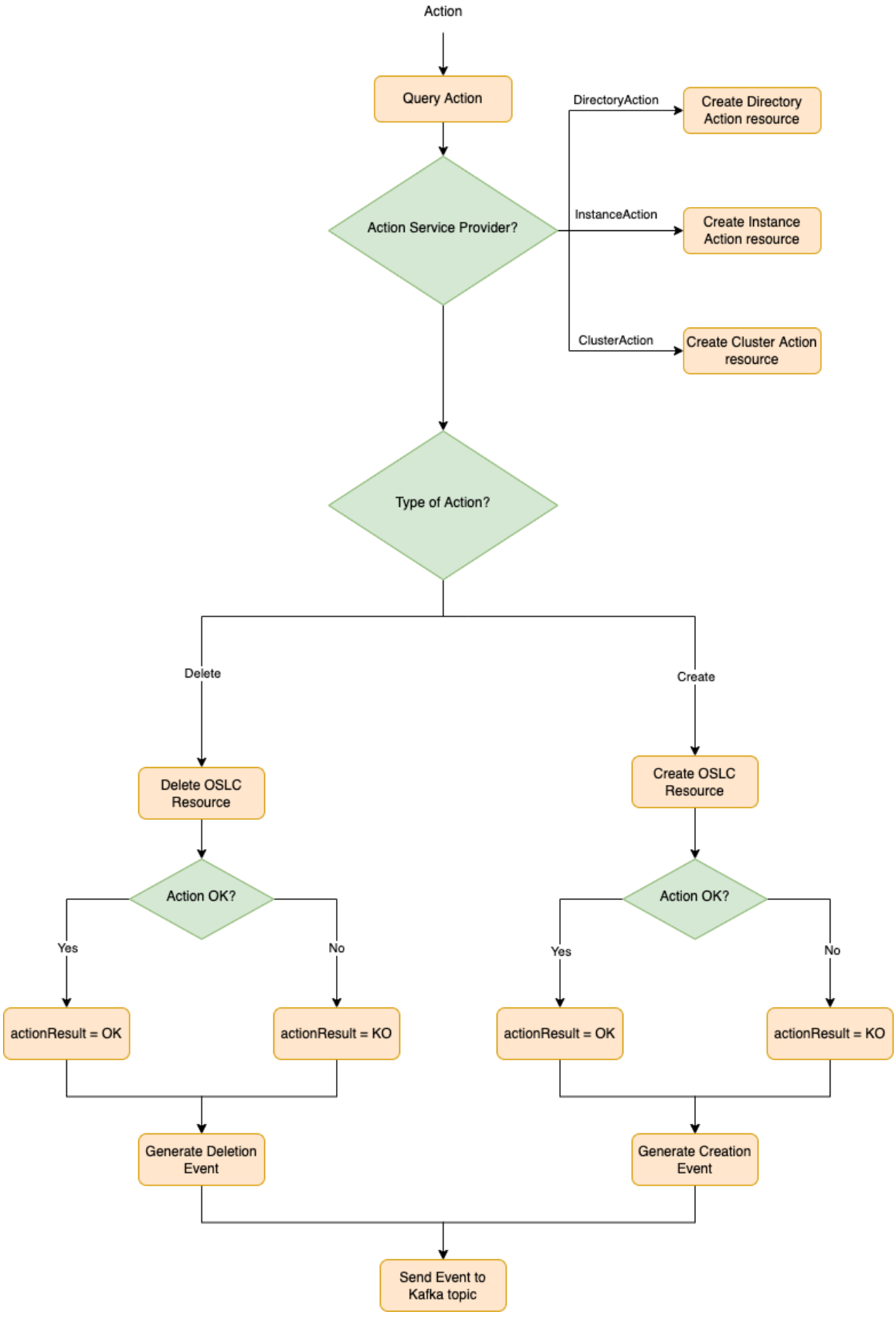


Figure 4.8: Resource management and event generation

field indicating the type of log associated with the active project, which, in this case, is an activity log, done by the "weighty-time-341718" project, which is the actual project used for the development of the project.

Listing 4.5: Cloud Logging Log message example


```
{
  "insertId": "-xaj2frdvey7",
  "logName": "projects/weighty-time-341718/logs/cloudaudit.googleapis.com%2Factivity",
  "protoPayload": {
    "@type": "type.googleapis.com/google.cloud.audit.AuditLog",
    "authenticationInfo": {
      "principalEmail": "oslc.tfm.gcp@gmail.com"
    },
    "authorizationInfo": [
      {
        "granted": true,
        "permission": "storage.buckets.getIamPolicy",
        "resource": "projects/_/buckets/test-adapter2",
        "resourceAttributes": {}
      },
      {
        "granted": true,
        "permission": "storage.buckets.delete",
        "resource": "projects/_/buckets/test-adapter2",
        "resourceAttributes": {}
      }
    ],
    "methodName": "storage.buckets.delete",
    },
    "resourceLocation": {
      "currentLocations": [
        "us"
      ]
    },
    "resourceName": "projects/_/buckets/test-adapter2",
    "serviceName": "storage.googleapis.com",
    "status": {}
  },
  "receiveTimestamp": "2022-05-21T08:22:00.189234166Z",
  "resource": {
    "labels": {
      "bucket_name": "test-adapter2",
      "location": "us",
      "project_id": "weighty-time-341718"
    },
    "type": "gcs_bucket"
  },
  "timestamp": "2022-05-21T08:21:59.056788702Z"
}
```


In addition, the "principalEmail" field is used to indicate the email from which the action is performed. Also, there are some fields which refer to "resource", in this case, shows that an action is being done over "test-adapter2" bucket. To identify which specific action is being done, the "methodName" field is used. In this case, it shows the value "storage.bucket.delete", indicating that a storage object of type bucket is being deleted. By using this field, the adapter can realize the type of action in order to replicate into its internal resources. Other fields, such as "location" or "timestamp", provide information on the location of the resource and the timestamp on which the action is being done.

To read it, since the full message is codified in bytes, the message is decoded from the utf-8 format to obtain the Unicode text. Then, as the data contained in the message are also binary encoded, a Base64 decoding is needed, so a string format of the data is finally obtained.

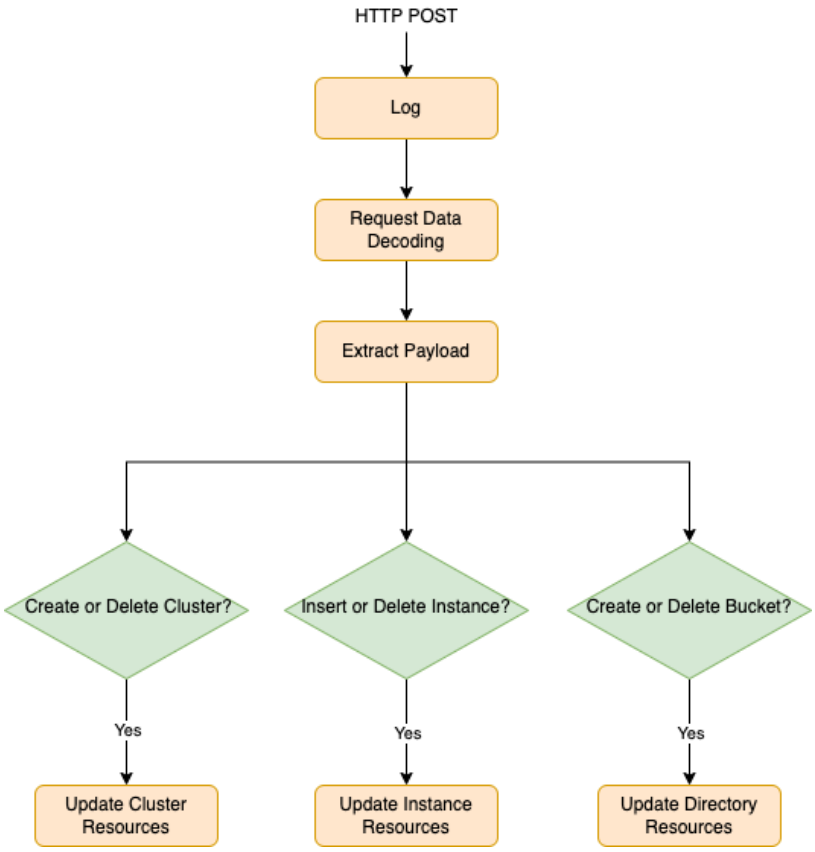


Figure 4.9: *GCPLogs* post function

The workflow of this method can be found in 4.9. After decoding the message, the payload of the message is extracted taking into account the json format of the message of 4.5. The "methodName" field contained in "protoPayload" object is extracted from the message in order to interpret the type of log that GCP has sent.

Depending on the type, whether it is a creation or deletion of buckets, instances, or clusters, an update of that type of resource is triggered by calling the *update_resources()* function, explained in 4.4.1.

Therefore, with this method, resources are always up-to-date and totally synchronized with the real resources on GCP in almost real time, by sending automatically logs when any changes on the platform occur, and the adapter will detect any changes either from actions externally received by other modules or systems, or from logs of the cloud platform. Integration with different cloud vendors will be as easy as routing logs and interpreting

their content similarly as explained in this section.

4.5.3 Helpers

Service API

service_api.py module, contains all of the functions that have an interaction with the Google API. This module is called from the *resourceOSLC.py* file explained above. It is the core module for interacting with the APIs of external service providers. In case any other provider is desired to be added, specific functions that call them respective APIs must be included on this module. All functions can be seen in Figure 4.10.

Regarding the Google Cloud Storage service, it contains *list_bucket()* that returns a list of active buckets in GCP, as well as *get_bucket()*, which returns a Bucket object given a bucket name that needs to be passed as a parameter.

Regarding Google Compute Engine, it contains functions such as *list_instances()* given the project ID, which returns a list of active instances from every zone; *create_instance()* function to create virtual machine instances, given the project ID, instance name, and machine type; and *get_instance()* function, given the project ID, instance name, and zone.

Finally, in relation to Google Kubernetes Engine, it contains a *list_clusters()* function that retrieves a list of active clusters.

In addition, it contains a couple of functions to map the actual attributes of the real resources with OSLC semantic resources, so depending on which service provider and which type of object, it maps one specific attribute or another.

On the one hand, it provides the function *module_to_service_provider()* that maps a module, in this case the type of cloud service, to the attributes of the service provider. Thus, depending on whether the module is a FilesystemService, a VirtualMachineSerive, or a ContainerService, the function takes different attributes to add them to the Service Provider graph. A representation of this function can be found in Figure 4.4.

On the other hand, it provides the *element_to_oslc_resource()* function that directly maps a specific element of a specific cloud service to an OSLC resource. As the Google API usually returns objects of a specific type depending on the case, there is a check before the mapping to verify that the object is of bucket type, for a directory, instance type for a virtual machine, or dictionary type for clusters. A representation of this function can be found in Figure 4.6.

Service Actions

While the above module contained Google API functions to retrieve data from GCP, this module contains functions that take actions on the GCP service, such as creating or deleting resources. Observing the figure 4.8 with respect to the module *service_api.py*, and

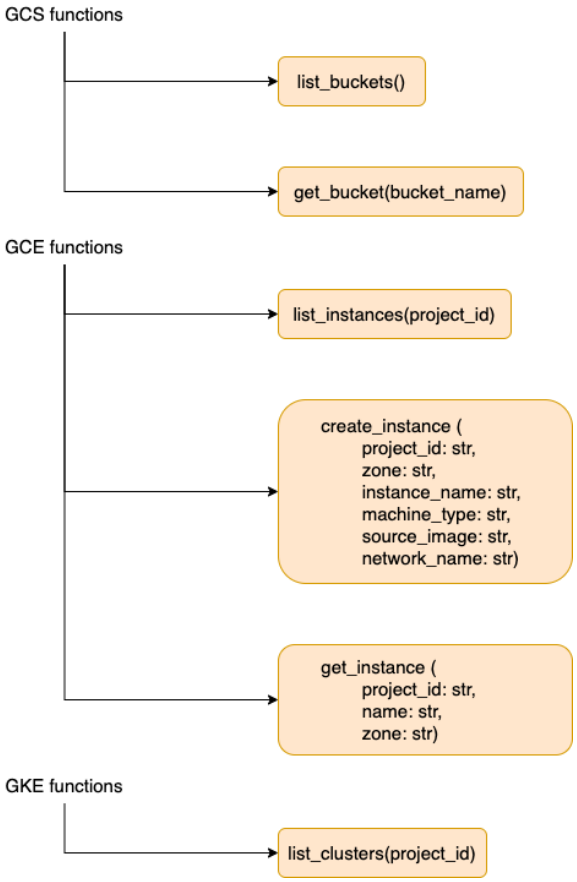


Figure 4.10: *service_api.py* functions

after checking the type of action to be performed, it calls the module *service_actions.py* to perform the creation or deletion of resources.

On the one hand, the function *create_resource()* is used for the creation of GCP resources by extracting useful information from the Action rdf resource that receives the OSLC adapter. Therefore, since the body of the HTTP request received by the adapter is an RDF, it is necessary to query the resource. This query can be found in 4.6 applied to a Directory creation action.

Listing 4.6: SPARQL Query for Directory Creation Action resources

```
query_bucket = """

        PREFIX oslc_gcp: <http://localhost:5001/GCP_OSLC/>

        SELECT ?name ?location ?storage_class

        WHERE {
```

```
            ?s oslc_gcp:directoryName ?name .
            ?s oslc_gcp:directoryLocation ?location .
            ?s oslc_gcp:directoryStorageClass ?storage_class .
        }
"""
```

As can be seen, it is necessary to retrieve useful information, such as the name of the directory, location, or storage class, to create the real resource via the Google API. Later, some logic is needed to check what type of service provider is the action and then retrieve the parameters and call the API to perform the real action in GCP, as can be seen in Figure 4.11.

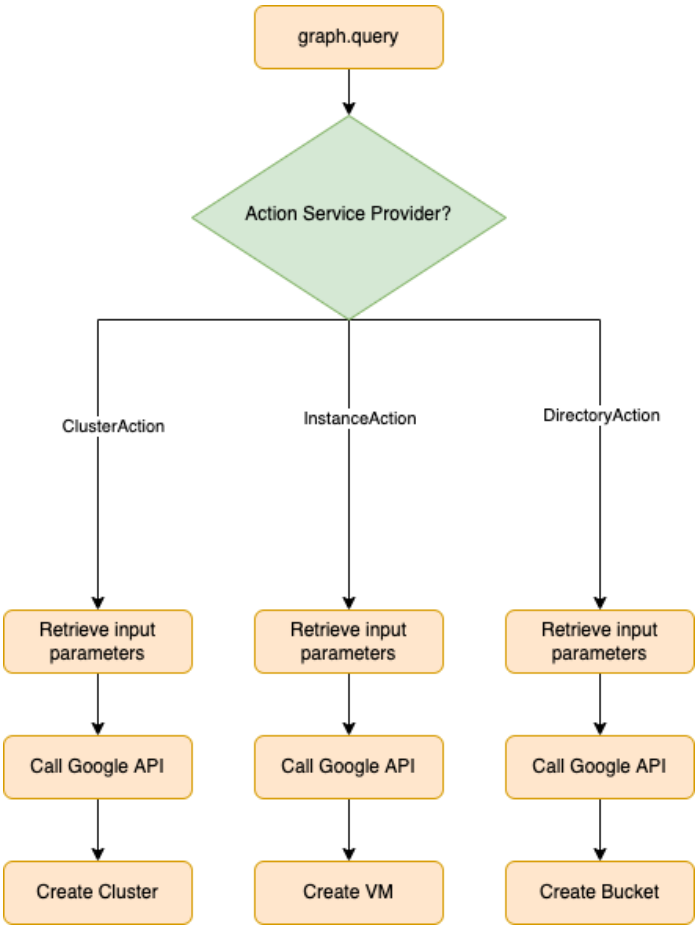


Figure 4.11: *create_resource()* function

On the other hand, the function *delete_resource()* is used for the deletion of resources in GCP when a delete action resource is posted on the OSLC Adapter action endpoint. After retrieving that a deletion Action is requested, again, the input resource is need to be queried. An example of this query, for instance, resources, can be found in the Listing 4.7.

Listing 4.7: SPARQL Query for Instance Deletion Action resources

```
query_instance = """

                PREFIX oslc_gcp: <http://localhost:5001/GCP_OSLC/>

                SELECT ?name ?zone

                WHERE {
                    ?s oslc_gcp:instanceName ?name .
                    ?s oslc_gcp:instanceZone ?zone .
                }
"""
```

Now, it is necessary to check which type of service provider is requested to do the action, since it is necessary to use different functions to call the Google API. After checking it, it performs the required action on GCP.

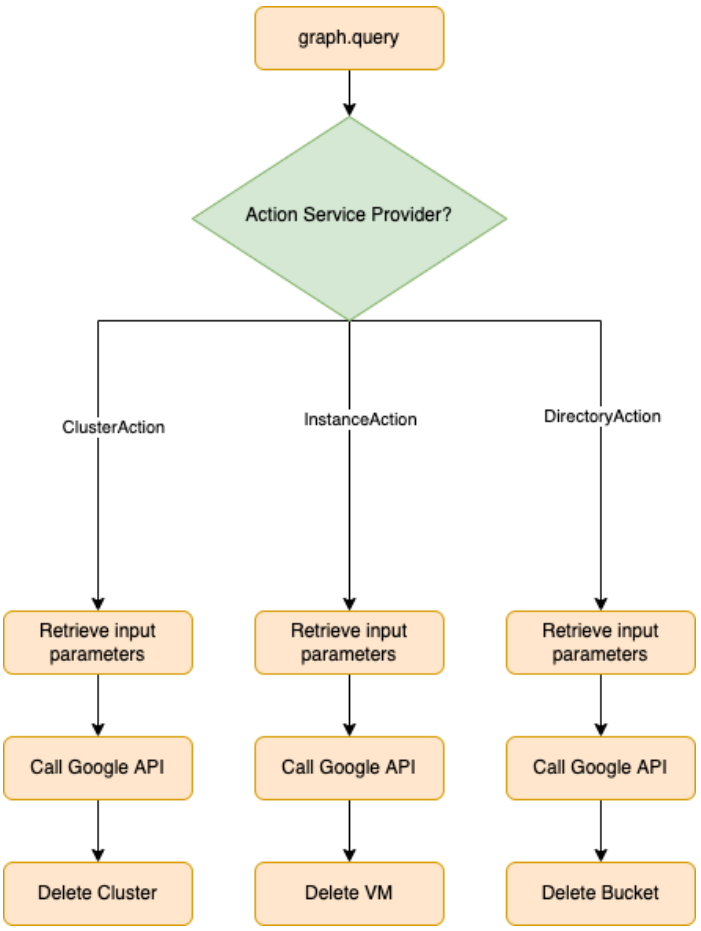


Figure 4.12: *delete_resource()* function

Service Events

This module is used to generate the OSLC Event resources and integrate the events into the TRS. It contains all the functions to perform actions such as *generate_creation_event()*, *generate_modification_event()*, and *generate_deletion_event()*.

When a creation event is performed, the RDF resource is passed as parameter and the *generate_creation_event()* function is called with the "Creation" action type as input. This function integrates with the TRS this action. Then, a graph is created with an OSLC Event resource along with the type of the event, in this case, the "Creation Event", and it is returned.

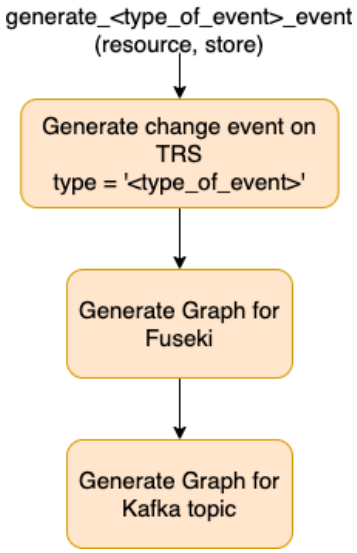


Figure 4.13: Events function

The modification or deletion event performs actions similar to the creation event function. First, integration of the event with the TRS is performed and then a graph is generated and returned with the type of event.

In addition, apart from the generation of the Events resources locally on the adapter, as Apache Jena Fuseki server is used in order to query these events, RDF Graphs are also sent to this server endpoint. In order to send these resources correctly to the Fuseki endpoint, some parameters need to be set.

First, it is necessary to create a *SPARQLUpdateStore* object, introducing a proper *auth* variable containing the correct user and password of the Fuseki service, already configured with these credentials. Then, it is necessary to define a *query_endpoint* with a URL "http://⟨endpoint_url⟩/⟨dataset⟩/query", and a *update_endpoint* with an URL "http://⟨endpoint_url⟩/⟨data and call the *SPARQLUpdateStore.open()* function that introduces both parameters. Finally, when creating the Graph resource, it is necessary to introduce, as a parameter, the previous

SPARQLUpdateStore object and add to that graph any desired RDF triples.

Therefore, all triples added to that store will be sent to the Fuseki server and will be stored, so it will be possible to make queries in the Fuseki user interface to retrieve them.

4.6 Logs routing

In this section, the implementation of GCP log routing will be explained. As the OSLC Adapter needed to be aware of any changes on its resources but carried out on GCP, there was a necessity of establish a communication between GCP and the adapter by any means.

After researching different possibilities, Google Cloud Logging has been the best solution. This service allows one to store all of the cloud platform logs, which shows any changes on its resources, on buckets of Google Cloud Storage. Also, it provides a user interface to query all of the logs, so there is the possibility of retrieving only a set of logs that meets some conditions.

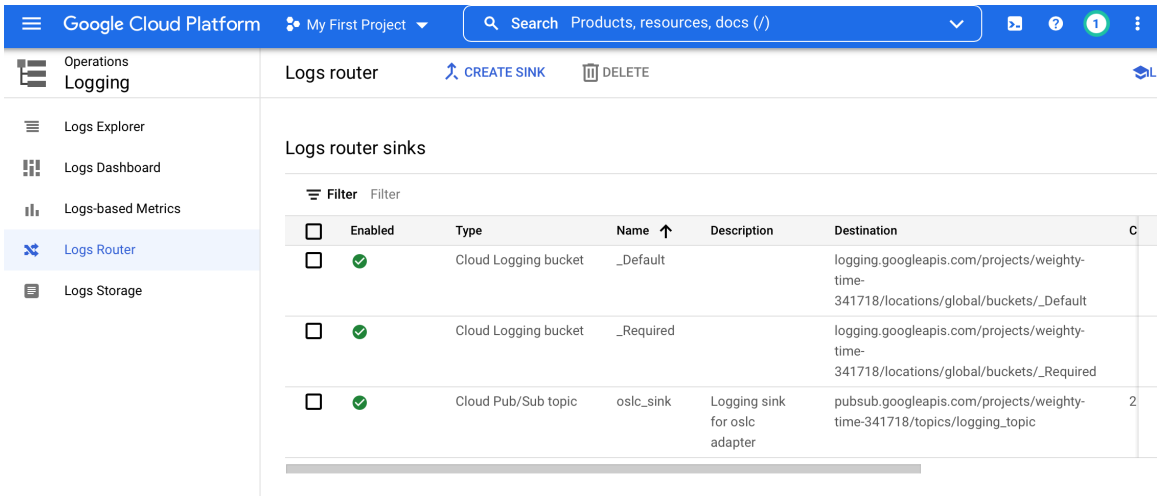


Figure 4.14: Cloud Logging Sink

To route all logs to an endpoint, a log sink was used. The sink type that has been configured is a Cloud Pub/Sub topic. This means that all of the logs will be routed into a Pub/Sub topic, so it will receive and store all of them. As a messaging service, Pub / Sub must have a subscriber to the topic, so a *logging_topic_sub* subscription has been created.

At this point, there were two possibilities to create the subscription. On the one hand, there is the possibility to create a pull subscription, which means that the messages of the topic need to be pulled every time an update on the resources is needed. This implies creating a thread function to retrieve the logs synchronously.

On the other hand, with the actual solution that has been carried out, it is possible to configure a push subscription that means that Pub/Sub automatically sends HTTP POST

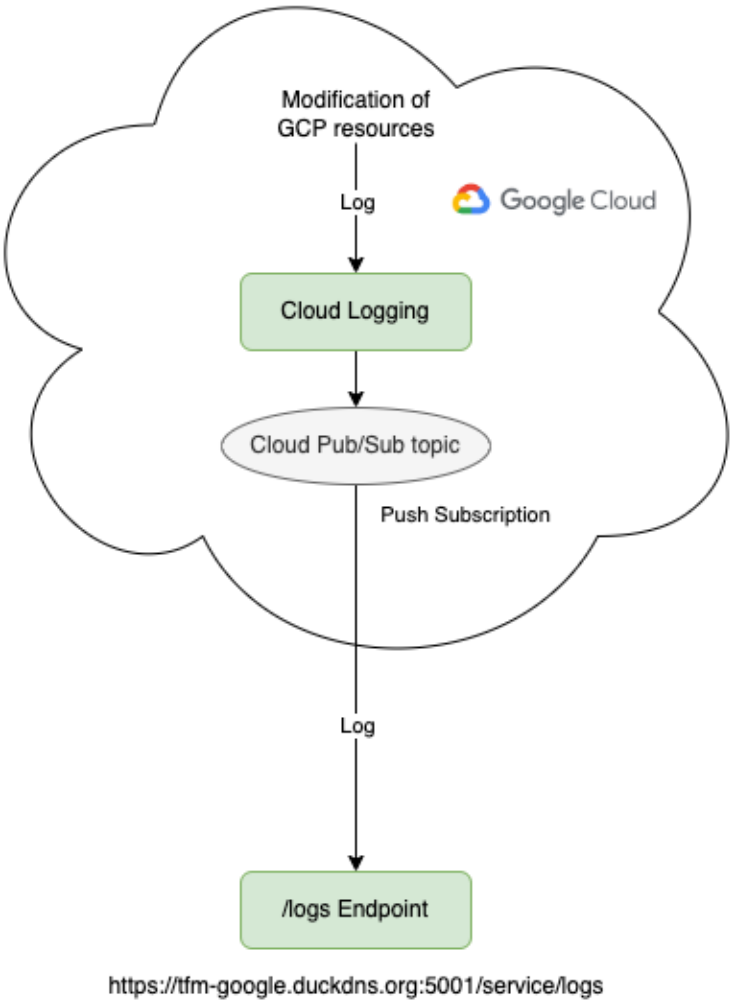


Figure 4.15: Logs routing process

requests to the endpoint of your choice. A problem occurred here with the communication of GCP and the adapter, as, first, it was developed locally and over HTTP, instead of HTTPS. Therefore, the adapter was migrated and executed on a GCE virtual machine to make it publicly accessible via HTTPS, performing several actions that will be explained in the next section 4.7.

4.7 SSL Certificates Generation

At first, as the development of the adapter was carried out locally, there was no issue in working over http and not over https, as there was no necessity to secure the endpoint when performing simple HTTP requests from a personal laptop connected to a home network.

The main issue occurred when trying to integrate the adapter with Google Cloud logging services. As mentioned above, Pub/Sub was not able to have a push subscription to an endpoint URL that does not have a publicly accessible HTTPS address, so the server for the push endpoint must have a valid SSL certificate signed by a certificate authority. To accomplish this task, the following process has been followed.

The DuckDNS domain needs to first be configured with a new domain linked to the external IP of the GCE virtual machine. Then, in order to avoid an update on the IP, and therefore, the domain points to a wrong IP address, a bash script is needed on the virtual machine to auto-update this new IP address with the already created domain. In this case, the domain "https://tfm-google.duckdns.org" has been created.

The bash script contains a simple line that calls the DuckDNS URL to update the IP address to which the domain points and is configured to run as a crontab every five minutes. Additionally, a firewall configuration on the virtual machine is needed to allow Internet traffic into the actual OSLC Adapter server running as shown in Figure 4.16.

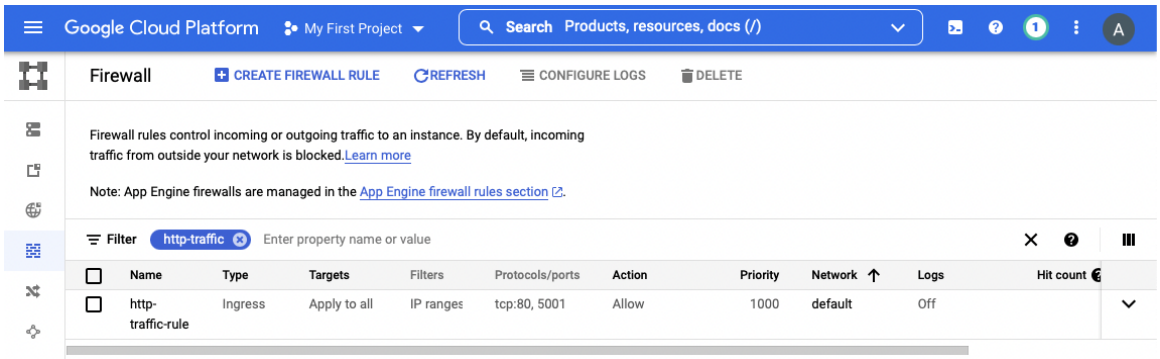


Figure 4.16: Firewall configuration

In conclusion, a certificate is generated with the files *cert.pem* and *privkey.pem*, which are the files that will be included in *docker-compose.yml* to run the Flask server over HTTPS, and therefore enables communication between GCP and the server through Pub/Sub push subscription.

CHAPTER 5

Use Cases

This chapter will describe the different use cases of the OSLC Adapter along with a demonstration scenario in order to test the adapter capabilities

5.1 Introduction

After explaining the architecture of the adapter, the global project on which it will be integrated, and all of the internal modules which conform to the adapter, in this chapter it will be explained the different use cases that accomplish all of main objectives of the project and explain the functioning of the system.

Three use cases will be explained. The first will explain a normal scenario in which an action is executed from the outside of the adapter, but directly following it. The second one will explain a situation in which an action is done on the Google Cloud Platform, and the adapter will notice every change and execute a certain action and generate an event internally. Finally, a demo scenario will be introduced that exposes a real situation in which the adapter can deploy multiple resources of interest.

5.2 Use Case: Normal Scenario

As a normal scenario of use, as shown in Figure 3.2, the interaction within the adapter will be carried out through the distributed Event-based Messaging Platform which, as explained before, will be formed by Kafka. In this case, a message will be published on a Kafka topic and will be received by the adapter to perform any action. Instead, as a matter of testing, a user will send an action from the outside of the system to the adapter.

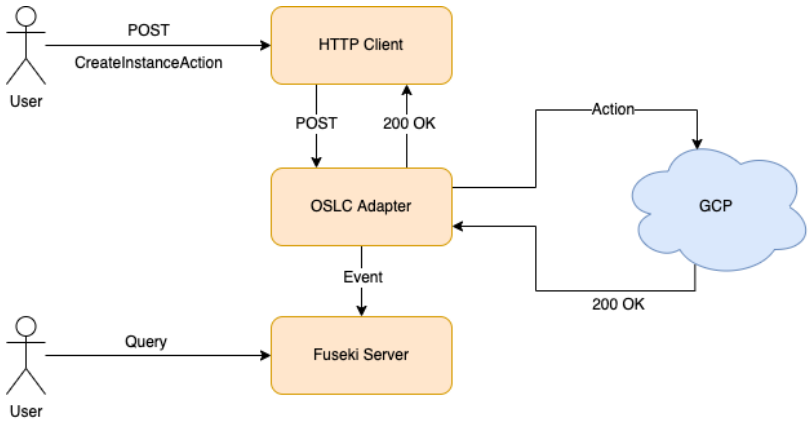


Figure 5.1: Use Case 1

Therefore, an Action resource will be sent by a user through an HTTP client, as is Insomnia. This resource will be sent to the action endpoint, pointing to the server, with the URL "https://tfm-google.duckdns.org:5001/service/action". This resource will be an RDF Graph in rdf/xml format, which will contain an Action resource with the type of action and the necessary parameters to perform that action. In this case, it will be sent in the first

case, a CreateInstanceAction, as can be seen in Figure 5.2.

POST https://tfm-google.duckdns.org:5001/service/action Send

Other Auth Query Header 2 Docs

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:oslc="http://open-services.net/ns/core#"
  xmlns:oslc_gcp="http://localhost:5001/GCP_OSLC/"
  xmlns:action="http://open-services.net/ns/actions#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
  <rdf:Description>
  <rdf:type rdf:resource="http://localhost:5001/GCP_OSLC/CreateInstanceAction"/>
    <oslc_gcp:instanceName>test-instance-oslc-adapter</oslc_gcp:instanceName>
    <oslc_gcp:instanceZone>us-central1-c</oslc_gcp:instanceZone>
  </rdf:Description>
</rdf:RDF>
```

Figure 5.2: CreateInstanceAction resource

Now, after sending the HTTP POST petition, the adapter receives the resource and will process it. Then, it will respond with the created resource again, in rdf / xml format, as can be seen in Figure 5.4.

200 OK 8.82 s 1008 B Just Now

Preview Header 6 Cookie Timeline

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:ns1="http://open-services.net/ns/core#"
  xmlns:ns2="http://localhost:5001/GCP_OSLC/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
  <rdf:Description
rdf:about="http://localhost:5001/GCP_OSLC/service/serviceProviders/ComputeEngine/instance/409782570872970694">
    <rdf:type rdf:resource="http://localhost:5001/GCP_OSLC/Instance"/>
    <rdf:type rdf:resource="http://localhost:5001/GCP_OSLC/CreateInstanceAction"/>
    <ns1:serviceProvider rdf:resource="http://localhost:5001/GCP_OSLC/service/serviceProviders/2"/>
    <ns2:instanceName>test-instance-oslc-adapter</ns2:instanceName>
    <ns2:instanceZone>https://www.googleapis.com/compute/v1/projects/weighty-time-341718/zones/us-central1-
c</ns2:instanceZone>
    <ns2:instanceZone>us-central1-c</ns2:instanceZone>
    <ns2:instanceCreationTimestamp>2022-06-15T11:21:29.876-07:00</ns2:instanceCreationTimestamp>
    <ns1:details>RUNNING</ns1:details>
  </rdf:Description>
</rdf:RDF>
```

Figure 5.3: CreateInstanceAction response

The resource can then be observed on Google Cloud Platform after posting the resource. The virtual machine named "test-instance-oslc-adapter" is deployed and running properly on the platform, as can be seen in Figure 5.4.

Finally, the event is sent to the Kafka topic and the Apache Jena Fuseki server, so

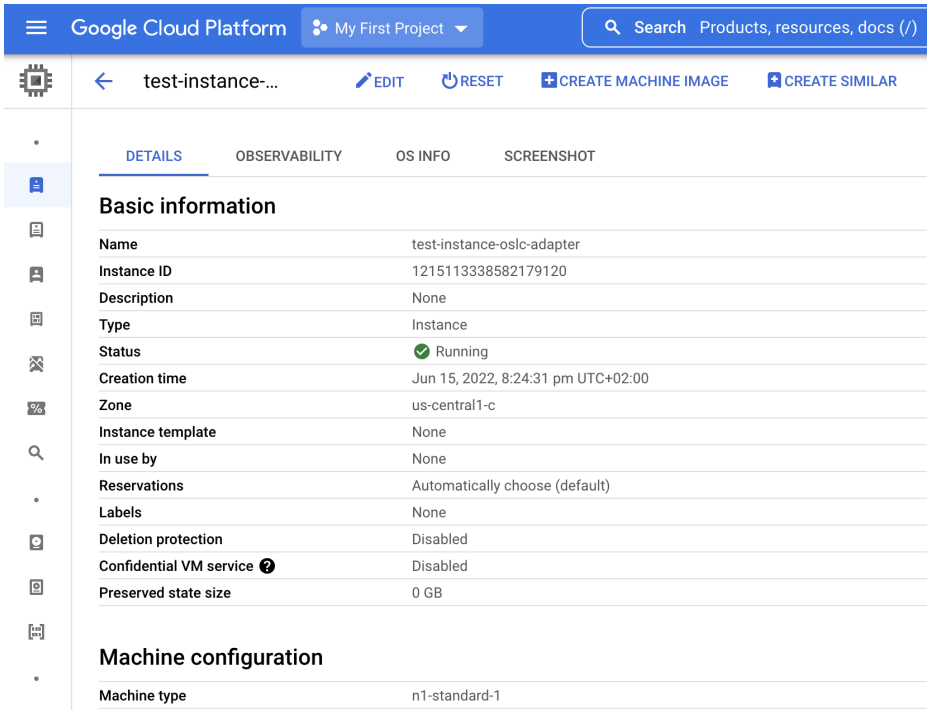


Figure 5.4: Instance created on GCP

therefore the event resource can be queried on the Fuseki server, as can be seen in Figure 5.5.

Here, it can be seen that the instance with id 1215113338582179120, is generated, and can be checked by observing that a "Creation Event" appears associated with that resource.

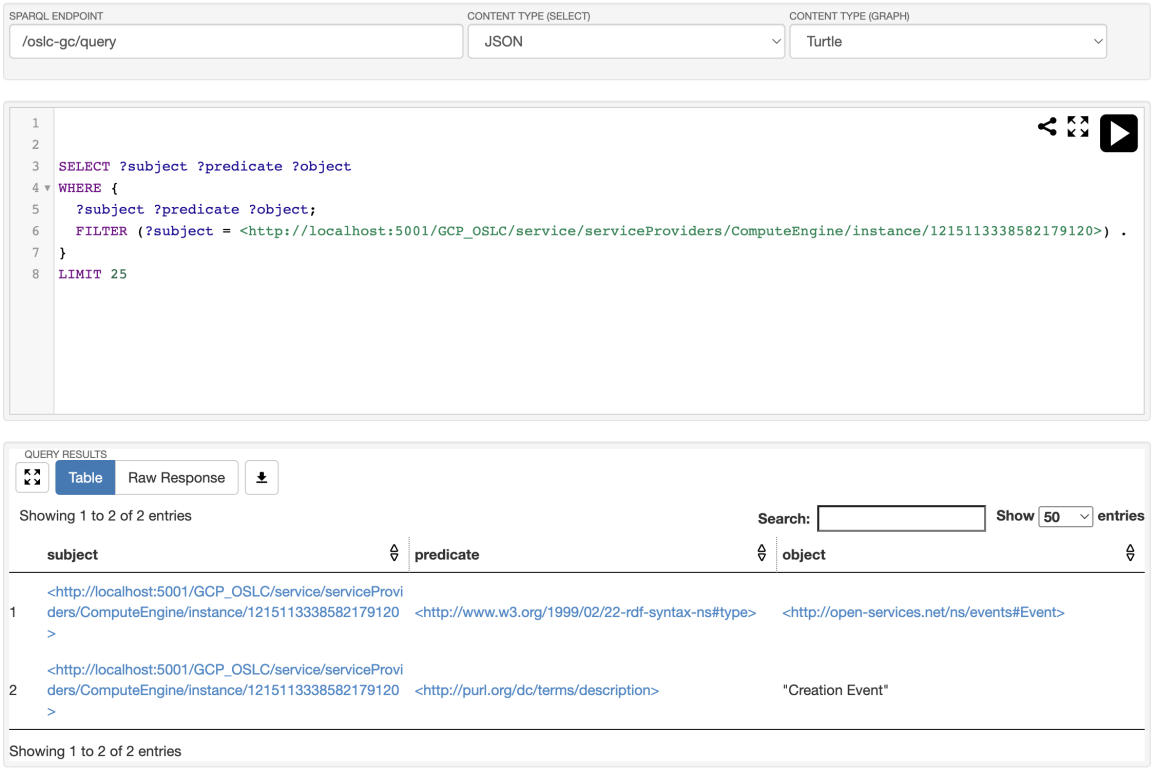


Figure 5.5: Creation Event on Fuseki server

5.3 Use Case: Action in Google Cloud

The second use case involves an action done directly in Google Cloud Platform, without interacting with the adapter. Therefore, a user performs an action on the platform, as it can be the creation of a Google Cloud Storage Bucket to store files. Initially, when the adapter is run for the first time, as some directories have not yet been created, they can be seen to be active on the adapter, as can be seen in Figure 5.7.

As can be seen here, there is only one active directory already created on Google Cloud, and therefore only this resource is created on the adapter. Then, when an action is done directly in GCP, such as the creation of another directory with the name "testing-use-case2", as can be seen in Figure 5.8, the adapter is aware of this and detects the action, adding the resource internally.

In Figure 5.9, the active directories in the adapter can be seen, before the action carried out in GCP.

Here, it can be seen that the directory "testing-use-case2" is active and is created directly in the adapter without any interaction with it and only performing an action on GCP.

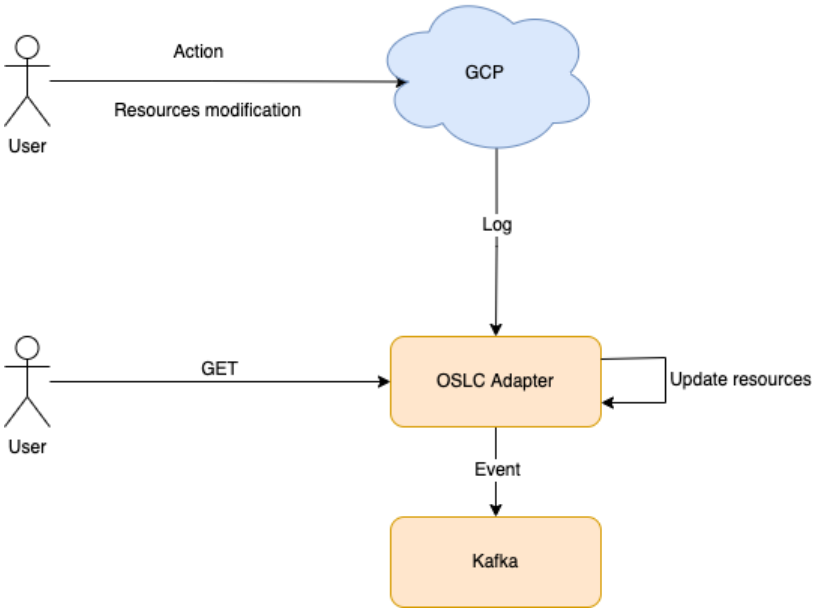


Figure 5.6: Use Case 2

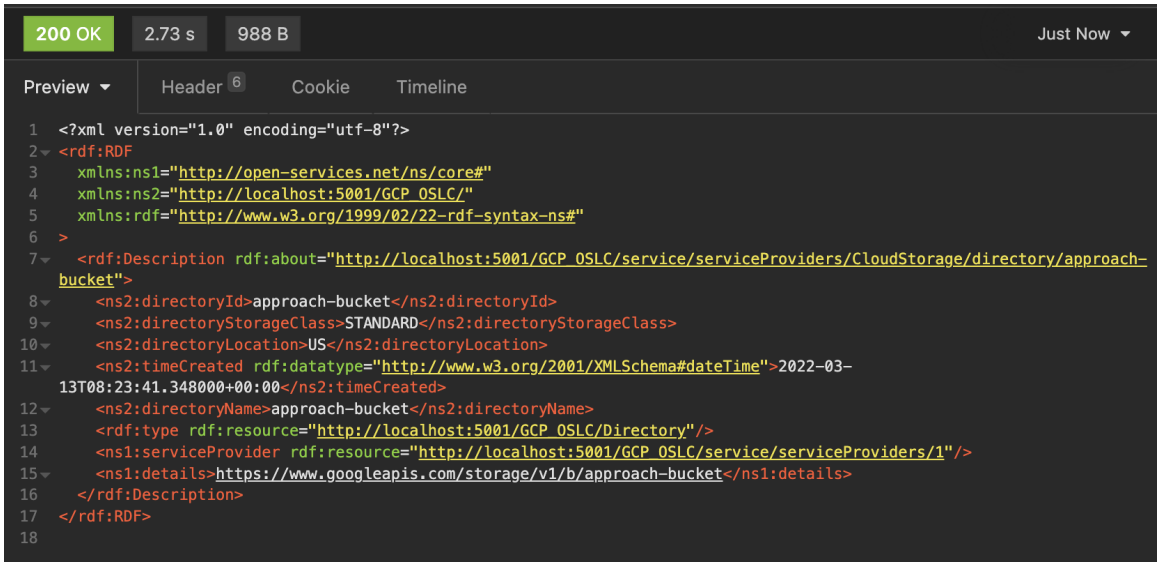


Figure 5.7: Initial active directories of the adapter

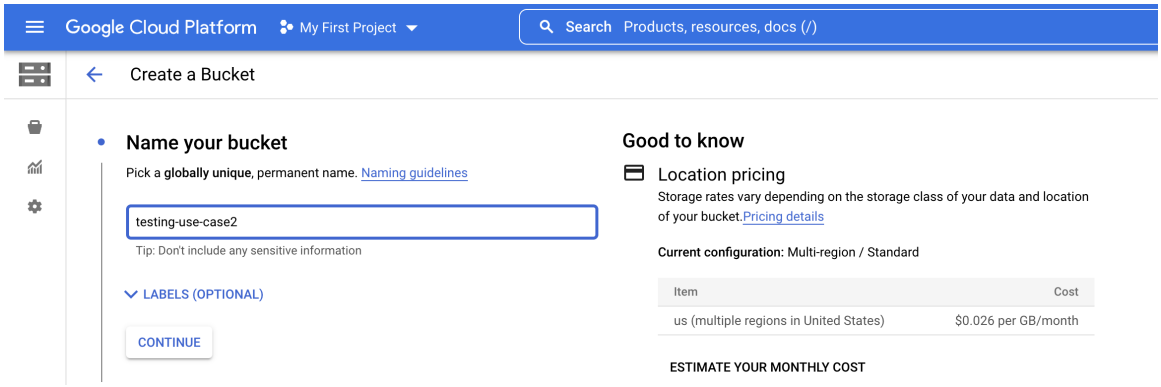


Figure 5.8: Creation of a directory in GCP UI

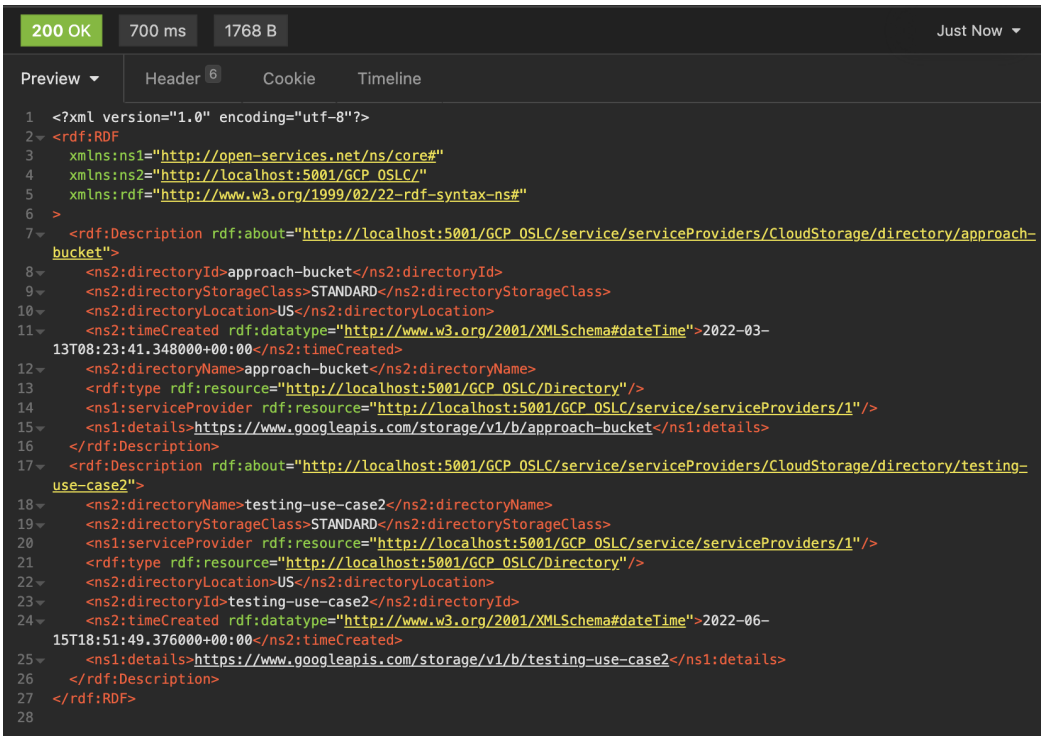


Figure 5.9: Active directories in the adapter after the creation of a directory in GCP

5.4 Use Case: Demo Scenario

Finally, a demo scenario was developed so that adapter capabilities could be tested in an almost real environment. In order to introduce the demo scenario, context needs to be added. For this use case, the GitHub platform will be involved; as being an open source repository of code, it is possible to track any updates on a specific project.

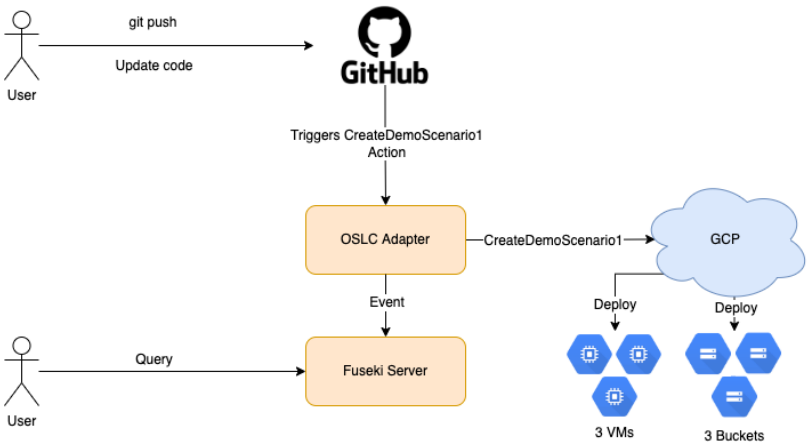


Figure 5.10: Use Case 3

Therefore, the use case starts when the code of a specific repository is updated and, as occurs when an action is done on GCP as explained in Section 5.3, a complementary system needs to be implemented in order to track GitHub changes, and then integrated with the whole project. This system is not implemented since it is a theoretical situation used for the explanation of a real use case on which the OSLC adapter takes action.

Once the update is completed, an event is sent to the global system and the adapter is aware of this event, so it will process it. Then, a "Demo Scenario" action will be triggered in order to deploy some architecture and resources on Google Cloud. To simulate this event, the "CreateDemoScenario1" action will be triggered directly on the adapter action endpoint by sending the RDF containing this resource type.

By sending this type of action resource, the adapter identifies it and deploys a total of three virtual machines with the updated code of the system, as well as a file system service for each of them, that is, a bucket for each of them.

On the one hand, by the post of this action resource of the scenario, three virtual machines are deployed containing the update code with completely updated functionality. This could be a newer version of a web application, a newer machine learning model, or a newer version of a natural language bot. These resources can be found in Figure 5.12.

POST https://tfm-google.duckdns.org:5001/service/action Send

Other Auth Query Header 2 Docs

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <rdf:RDF
3    xmlns:oslc="http://open-services.net/ns/core#"
4    xmlns:oslc_gcp="http://localhost:5001/GCP_OSLC/"
5    xmlns:action="http://open-services.net/ns/actions#"
6    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
7
8  >
9    <rdf:Description>
10     <rdf:type rdf:resource="http://localhost:5001/GCP_OSLC/CreateDemoScenario1"/>
11   </rdf:Description>
12 </rdf:RDF>
```

Figure 5.11: CreateDemoScenario1 resource

tfm-oslc	us-west4-b	10.182.0.5 (nic0)	34.125.211.108 (nic0)	SSH		
vm1-scenario1	us-central1-c	10.128.0.40 (nic0)		SSH		
vm2-scenario1	us-central1-c	10.128.0.41 (nic0)		SSH		
vm3-scenario1	us-central1-c	10.128.0.42 (nic0)		SSH		

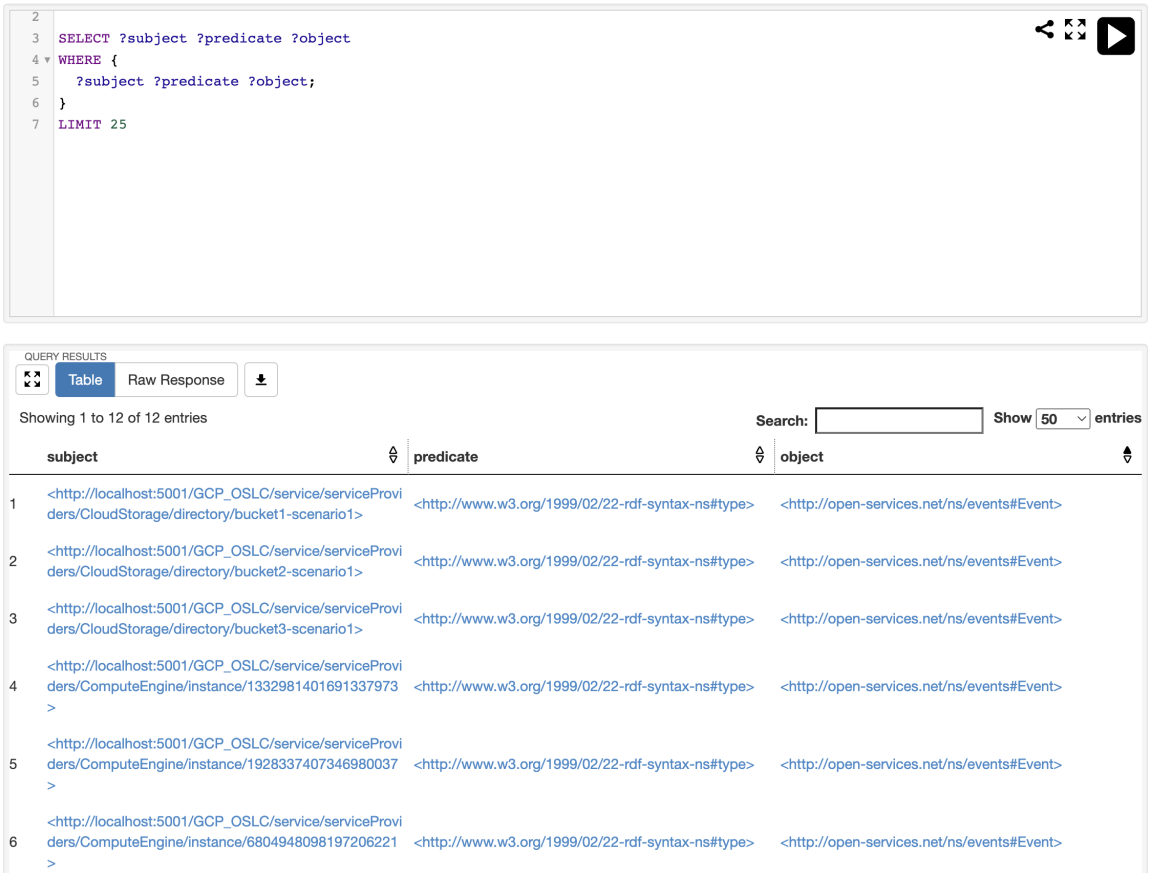
Figure 5.12: Virtual Machines created with the Demo Scenario action resource

On the other hand, a file system service is also deployed for each of the VMs to provide storage capability for different necessary files for a specific application. This Buckets can be found in Figure 5.13.

bucket1-scenario1	18 Jun 2022, 19:24:01	Multi-region	us	Standard	18 Jun 2022, 19:24:01	Subject to object ACLs	Fine-grained
bucket2-scenario1	18 Jun 2022, 19:24:03	Multi-region	us	Standard	18 Jun 2022, 19:24:03	Subject to object ACLs	Fine-grained
bucket3-scenario1	18 Jun 2022, 19:24:04	Multi-region	us	Standard	18 Jun 2022, 19:24:04	Subject to object ACLs	Fine-grained

Figure 5.13: Buckets created with the Demo Scenario action resource

Finally, as part of the functioning of the adapter, an event per created resource is generated and published into a Kafka topic, as well as sent to the Apache Jena Fuseki server, so it is possible to query the event resources and confirm the creation of the different items of the demo. The complete list of the resources after executing the Demo Scenario, can be seen in the Fuseki server in Figure 5.14.



```
SELECT ?subject ?predicate ?object
WHERE {
  ?subject ?predicate ?object;
}
LIMIT 25
```

QUERY RESULTS

Table | Raw Response

Showing 1 to 12 of 12 entries

Search: Show 50 entries

	subject	predicate	object
1	<http://localhost:5001/GCP_OSLC/service/serviceProviders/CloudStorage/directory/bucket1-scenario1>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://open-services.net/ns/events#Event>
2	<http://localhost:5001/GCP_OSLC/service/serviceProviders/CloudStorage/directory/bucket2-scenario1>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://open-services.net/ns/events#Event>
3	<http://localhost:5001/GCP_OSLC/service/serviceProviders/CloudStorage/directory/bucket3-scenario1>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://open-services.net/ns/events#Event>
4	<http://localhost:5001/GCP_OSLC/service/serviceProviders/ComputeEngine/instance/1332981401691337973>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://open-services.net/ns/events#Event>
5	<http://localhost:5001/GCP_OSLC/service/serviceProviders/ComputeEngine/instance/1928337407346980037>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://open-services.net/ns/events#Event>
6	<http://localhost:5001/GCP_OSLC/service/serviceProviders/ComputeEngine/instance/6804948098197206221>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://open-services.net/ns/events#Event>

Figure 5.14: Demo Scenario created resources in Fuseki

As a conclusion, the demo scenario triggers a total of six resources which can be listed on the Fuseki server and corroborates the adapter capabilities in a real scenario use case, so it is possible to create any type of custom action in order to deploy a specific scenario for a specific use case.

CHAPTER 6

Conclusions and Future Work

This chapter will describe the conclusions of the master thesis along with the achieved goals and a discussion of future work.

6.1 Conclusion

In this project, deep research has been carried out on OSLC, a standard specification that provides full flexibility and integration of a diversity of tools that enables the interaction with resources by using the same standard language. The study has concluded with a seamless integration of Google Cloud Platform as a cloud service provider, establishing a standard resource definition and classification, by creating an extension of the OSLC ontology and creating a cloud semantic model.

This model establishes the basis for allowing the integration of other cloud providers and all of its resources and services, in an agile and fast way. The addition of cloud providers to the model would be as easy as importing the model and adding its resources into its different categories, so that every possible tool could be defined and integrated with the system.

The development of the project has been carried out taking into account the global scope and architecture of the SmartDevOps project, so that the developed system could be fully compatible with the global solution. In that way, an OSLC Adapter has been developed that interacts with standard Actions and generates Events, so that it enables the interaction with Big Data environments by retrieving and modifying its resources, whereas generating Events in response of actions, allowing compatibility with an event automation service.

In conclusion, this project represents a major step forward in standardizing cloud tools and services and implements a solution to avoid one of the biggest problems in Big Data applications, that is, vendor lock-in. It provides a solution that benefits the development of standard applications and unlocks the full potential of Big Data tools by integrating automation and standardization of a multi-cloud environment.

6.2 Achieved Goals

The goals achieved for this project are the following:

- **Research and study of semantic concepts and OSLC standard.** Semantic vocabulary and definitions have been studied as well as OSLC concepts in order to gain knowledge of this field.
- **Definition of a cloud semantic model following OSLC core concepts.** A cloud semantic model has been defined that allows the integration of multiple cloud service providers and its multiple services, as it has been developed following standard concepts and taking into account global resource definitions for cloud resources.

- **Design and development of an adapter to interact with Google APIs.** An OSLC Adapter has been developed taking into account the scope and architecture of the global project, fulfilling all the requirements to deal with Actions and Events resources, and allowing seamless integration with Big Data environments and DevOps tools, as the ones that Google Cloud provides.

- **Demonstration of the adapter capabilities by interacting with the resources.** A demonstration of the functionality of the adapter has been performed by deploying a real scenario of multiple resources and dealing with them in real time.

6.3 Future Work

The development of the project has covered multiple topics and achieved all original objectives. However, there are several improvement points that should be considered in future developments, to enhance the global scope of the project and extend the reach to a larger number of tools.

- **Integration with more cloud vendors.** As this project has been developed to interact with the Google Cloud Platform, there are several functions that are specific to Google, as it is necessary to use Google APIs to retrieve information and interact with its resources. In that way, if another cloud service provider is integrated on the system, it is needed to develop newer functions that complement the actual ones, for using it specific APIs for managing its resources. Therefore, all functions can coexist on the same code, avoiding the creation of a specific project for each cloud vendor.

- **Extension of the cloud semantic model.** Due to the large number of tools and services that cloud provider offers, a semantic model has been developed that covers the most important ones, but, since there are many of them, the actual model can be extended to cover all of them in newer categories. The designed semantic model provides the basis for allowing the integration of all existing cloud services following the principles of an open collaboration model that can be improved by continuous iterations.

- **On premise tools integration.** Despite of the numerous benefits that cloud applications have, it would be a key enabler for enterprises to integrate their own on-premise tools with other cloud services, so that they can easily create a hybrid multi-cloud environment, without taking care of incompatibilities and taking advantage of standard resources.

- **Provide full automation.** As a future objective of the actual project and the SmartDevOps project, any tool could be integrated through a semantic automation module that meets the OSLC standard and therefore could extend the use of OSLC. In consequence, the global project could cover all the necessary tools to run a business by using standard concepts and resources.

APPENDIX A

Code

All the code of the project can be found in the following GitHub repository: `https://github.com/AlexVaPe/pyOSLC_GCP`. There it can be found a README.md file with different steps to run the adapter. In the following, can be found an extract of this file.

A.1 README.md file

GCP Python OSLC Adapter

Python implementation of an OSLC Adapter for Google Cloud Platform

How to run locally the adapter:

1. Clone the repository.
2. Create a Google Cloud account or use an existing one.
3. Go to the API Console: https://console.developers.google.com/.
4. From the projects list, select a project or create a new one, and choose Credentials from the left panel.
5. Click Create credentials and then select API key → it will generate a .json file that you need to store at the root of the project.

6. Then, you need to modify the PROJECT_ID variable of the code with the id of your GCP project.

7. Modify the "command" of the docker-compose.yml to execute flask run command without –cert and –key options. This will run a flask server over http on http://localhost:5001 url.

8. For running the adapter, you need to execute: "docker-compose up"

Then, the server will be up and running and you will be able to make http post requests to the Action Endpoint in order to create/delete GCP resources. For this, you can use an API client as Insomnia (https://insomnia.rest).

However, the server will not be able to receive logs from GCP as it is running locally. In order to make it publicly accessible, it is needed to follow the process that will be explained on the next section.

Action Endpoint

Endpoint URL: http://0.0.0.0:5001/service/action

Headers of the request:

- Content-Type: application/rdf+xml
- Accept: application/rdf+xml

Example of a GCS Bucket creation

Body:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:ns1="http://open-services.net/ns/core#"
  xmlns:ns2="http://localhost:5001/GCP_OSLC/"
  xmlns:action="http://open-services.net/ns/actions#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

>
  <rdf:Description>
    <rdf:type rdf:resource="http://localhost:5001/GCP_OSLC/
        CreateDirectoryAction"/>
    <ns2:directoryName>test-adapter</ns2:directoryName>
    <ns2:directoryStorageClass>STANDARD</ns2:directoryStorageClass>
    <ns2:directoryLocation>US</ns2:directoryLocation>
  </rdf:Description>
</rdf:RDF>
```

Example of a GCE Instance creation

Body:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:ns1="http://open-services.net/ns/core#"
  xmlns:ns2="http://localhost:5001/GCP_OSLC/"
  xmlns:action="http://open-services.net/ns/actions#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
  <rdf:Description>
  <rdf:type rdf:resource="http://localhost:5001/GCP_OSLC/
      CreateInstanceAction"/>
    <ns2:instanceName>test-instance-oslc</ns2:instanceName>
    <ns2:instanceZone>us-central1-c</ns2:instanceZone>
  </rdf:Description>
</rdf:RDF>
```

How to make the adapter publicly accessible

1. First, it is recommended to run a GCE Instance on GCP and note the external ip of the machine.

2. Then, you need to follow the instructions of the following link: https://www.splitbrain.org/blog/2017-08/10-homeassistant_duckdns_letsencrypt, in order to get a domain and generate certificates for that domain and store them on the machine.

3. After that, you need to set up a Cloud Logging sink pointing to a Google Pub/Sub topic. You can follow Google official website: https://cloud.google.com/logging/docs/export/configur

4. Then, you need to create a push subscription on Google Pub/Sub (https://cloud.google.com/pubsub/docs/p pointing to the domain that you already created followed by the logs endpoint: "https://your$_{d}omain-duckdns.org:5001/service/logs"Finally, youneedtoincludetothe"command"fieldofthedocker-compose.ymlfilethe--certand--keyoptionsthatwereoriginallyincluded:"flaskrun-h0.0.0.0--cert=/code/certs/cert.pem--key=/code/certs/privkey.pem"

   Now, the interaction with the server is equal as before, but the url needs to be changed to the new one: "https:your_domain-duckdns.org:5001/service/action".

APPENDIX B

Project Impact

B.1 Social Impact

This project could have a positive impact on society as it leads to the improvement in the end of Big Data applications as it will facilitate their development. Therefore, since Big Data can be applied to any sector, it could lead to a better healthcare, transportation, or security sector.

B.2 Economic Impact

The economic impact of this project can be linked with one of the main objectives of it, that is the avoidance of vendor lock-in. Regarding economic aspects, with the actual situation, there exist only three cloud providers, and this can lead to a monopoly of these big tech companies, Amazon, Google, and Microsoft. As most of the cloud market is dominated by these three, it slows the rise of new vendors on the market.

With the proposed solution, this economic aspect will be impacted in a better way, in the sense of allowing the interaction with different companies and avoiding the lock on a single cloud vendor.

B.3 Environmental Impact

Big data on-premise applications often have an enormous carbon footprint due to the real state needed to house and store the server. By providing more facilities in order to migrate actual solutions into the Cloud, could lead to a high percentage of decrease of the energy used by data centers as cloud computing takes care of two vital elements for approaching a green IT infrastructure, that is, energy efficiency and resource efficiency.

B.4 Ethical Impact

Ethic implications are partially linked with economic impact, as a vendor lock-in may lead to monopoly of companies regarding the cloud services market. Therefore, with the proposed solution, this can be avoided, as it opens the possibility of participation of multiple companies that want to offer cloud services, so that they will not be affected by this monopoly.

APPENDIX C

Project Budget

C.1 Human Resources

This project has been developed within the GSI UPM department with an hourly net salary of 10€. This Final Master's Thesis within the Master's Degree of Telecommunications Engineering (MUIT) consists of 30 ECTS, being 1 ECTS, 30 hours of work. Therefore, this thesis requires at least a total of 900 hours. In conclusion, the associated cost for one person is 9.000 €.

C.2 Material Resources

For the development of this project, two machines have been used. First, a personal computer with the following characteristics has been used:

- **Model:** Apple Macbook Pro 14" 2021.
- **CPU:** Apple M1 Pro chip, 8 core CPU with 6 performance cores and 2 efficiency cores, 14 core GPU, 16 core neural engine and 200GB/s memory bandwidth.
- **RAM:** 16GB unified memory.
- **Disk:** 512GB SSD.

- **Operative System:** macOS Monterey 12.3.1.
- **Price:** 2,249€.

After, a virtual machine instance of Google Compute Engine have been used, with the following characteristics:

- **Machine type:** e2-medium.
- **CPU platform:** Intel Broadwell.
- **Memory:** 10GB.
- **Operative System:** Debian 10.
- **Price:** $0,046 hourly.

The first machine has been used for the development of the project, while the second has been used for testing the real performance of the developed code.

C.3 Licenses

Every tool used to develop the project follows an open source philosophy. However, the use of Google Cloud Platform requires a paid account, and, although it is a pay-per-use service, a free account has been used until credits have run out.

C.4 Total Costs

With all the information explained above, the total cost for the development of the project is approximately 11,249€.

Bibliography

[1] Bruno Almeida. Storage Options in Google Cloud: Block, Network File, and Object Storage. https://cloud.netapp.com/blog/object-storage-block-and-shared-file-storage-in-google-cloud, 2020.

[2] Apache. Introduction to Kafka. https://kafka.apache.org/intro. Accessed on: 15-06-2022.

[3] Microsoft Azure. What are public, private, and hybrid clouds? https://azure.microsoft.com/en-us/overview/what-are-private-public-hybrid-clouds/#public-cloud.

[4] Stephen J. Bigelow. Google Cloud. https://www.techtarget.com/searchcloudcomputing/definition/Google-Cloud-Platform, 2022.

[5] Fabian Calvo. DevOps and Cloud: A Perfect Symbiosis. https://www.teldat.com/blog/devops-cloud-software-development/, 2020.

[6] Ignacio Corcuera-Platas. Development of a Deep Learning Based Sentiment Analysis and Evaluation Service. Master thesis, ETSI Telecomunicación, Madrid, January 2018.

[7] Docker. Use containers to Build, Share and Run your applications. https://www.docker.com/resources/what-container/.

[8] IBM Cloud Education. Private Cloud. https://www.ibm.com/cloud/learn/introduction-to-private-cloud.

[9] Let's Encrypt. Getting Started. https://letsencrypt.org/getting-started/.

[10] Open Services for Lifecycle Collaboration. OSLC Specifications. https://open-services.net/specifications/.

[11] Open Services for Lifecycle Collaboration. OSLC Automation Specification Version 2.0. https://archive.open-services.net/wiki/automation/OSLC-Automation-Specification-Version-2.0/index.html, 2014.

[12] Open Services for Lifecycle Collaboration. OSLC Core Version 3.0. Part 2: Discovery. https://docs.oasis-open-projects.org/oslc-op/core/v3.0/os/discovery.html, 2021.

[13] Open Services for Lifecycle Collaboration. OSLC Core Version 3.0. Part 3: Resource Preview. https://docs.oasis-open-projects.org/oslc-op/core/v3.0/os/resource-preview.html#introduction, 2021.

[14] Open Services for Lifecycle Collaboration. OSLC Tracked Resource Set Version 3.0. Part 1: Specification. https://docs.oasis-open-projects.org/oslc-op/trs/v3.0/ps01/tracked-resource-set.html, 2022.

[15] Jerry Franklin. Apache Kafka Use Cases: When To Use It When Not To. https://www.upsolver.com/blog/apache-kafka-use-cases-when-to-use-not, 2022.

[16] Google. Cloud Logging. https://cloud.google.com/logging.

[17] Google. Google Cloud. https://cloud.google.com.

[18] Google. What is cloud computing? https://cloud.google.com/learn/what-is-cloud-computing.

[19] Google. What is Pub/Sub? https://cloud.google.com/pubsub/docs/overview, 2022.

[20] IBM. Linked data and OSLC integrations. https://www.ibm.com/docs/en/elm/6.0.1?topic=integrations-linked-data, 2021.

[21] IBM. Open Services for Lifecycle Collaboration integrations. https://www.ibm.com/docs/en/elm/6.0.1?topic=integrating-oslc-integrations, 2021.

[22] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. A survey of devops concepts and challenges. *ACM Comput. Surv.*, 52(6), nov 2019.

[23] Ekaterina Novoseltseva. Top 10 Benefits of Docker. https://dzone.com/articles/top-10-benefits-of-using-docker, 2017.

[24] OSLC Open Project. Open Services for Lifecycle Collaboration integrations. https://www.ibm.com/docs/en/elm/6.0.1?topic=integrating-oslc-integrations, 2021.

[25] Pythonbasics. What is Flask Python. https://pythonbasics.org/what-is-flask-python/.

[26] Stephen Watts Muhammad Raza. SaaS vs PaaS vs IaaS: What's The Difference How To Choose. https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/, 2019.

[27] Pablo Galindo Salgado. What's New In Python 3.10. https://docs.python.org/3/whatsnew/3.10.html, 2022.

[28] Meenal Sarda. Google Cloud Compute Services For Beginners. https://k21academy.com/google-cloud/google-cloud-compute-services/, 2021.

[29] Lukas Schauer. Dehydrated. https://github.com/dehydrated-io/dehydrated.

[30] Sofija Simic. What is Docker Compose. https://phoenixnap.com/kb/docker-compose, 2021.

[31] Stardog Union. RDF Graph Data Model. https://docs.stardog.com/tutorials/rdf-graph-data-model#rdf-graphs.

[32] Sai Vennam. Hybrid Cloud. https://www.ibm.com/cloud/learn/hybrid-cloud, 2021.