UNIVERSITY OF SAAD DAHLAB BLIDA

Faculty of sciences

Department of computer science

Report submitted for the fulfillment of the master degree

In Computer Science

Option : Computer Science and Networking (Système Informatique et Réseaux)

TOPIC :

Unification of communication interfaces for portability in the multi-cloud

Realized by :
RAZALI Ayoub
DERGUIL Hichem

Academic year : 2021/2022

Acknowledgements

We are deeply indebted to our research supervisor, Miss Mancer Yasmin for presenting us with such an interesting master degree topic. Each meeting with her added invaluable aspects to the implementation and broadened our perspective.
She has guided us with her invaluable suggestions, lightened up the way in our darkest times, and encouraged us a lot in our academic life. We have learned to think critically from her, select problems, solve them, and present their solutions.
We will be failing in our duty if we don't acknowledge some of our friends on the campus with whom we have shared our research experiences since it was a joy and enlightenment to us. We would like to address special thanks to the reviewers of our master degree report, for accepting to read and review this report.

Dedication

I dedicate my master degree report work to my family and many friends.

A special feeling of gratitude to my loving mother whose words of encouragement and push for tenacity ring in my ears.

My brothers Mohamed, Hamza, Bilal, and Adem have never left my side and are very special.

I also dedicate this work to my many friends Sidahmed, Lotfi, Habib, Sofiane,youness, Tarak, Mouadh, and Houssam who have supported me throughout the process.

I will always appreciate all they have done, especially Aguenini Aymen and Aymen Brahim for helping me develop my technology skills.

And my partner Derguil Hichem whom I enjoyed working with.

-Razali Ayoub-

Dedication

This work is dedicated to :

For the sake of Allah, my Creator, and my Master.

My great teacher and messenger, Mohammed (May Allah bless and grant him), taught us the purpose of life.

My great parents, who never stop giving of themselves in countless ways.

My beloved brothers Youcef, Billel, and my sister who stood by me when things look bleak.

My beloved niece and nephew Wail and Maram whom I can’t force myself to stop loving.

To all my family, the symbol of love and giving.

My friends who encourage and support me.

All the people in my life who touch my heart.

And my partner Razali Ayoub whom I enjoyed working with.

-Derguil hichem-

Abstract

Cloud computing is an emerging computing paradigm, which provides high service availability, high scalability as well as low usage costs. This has encouraged enterprises and individual users to embrace cloud technology. However, the lack of service interoperability and portability (also known as the vendor lock-in) issue persists.
The vendor lock-in is caused by the cloud service providers who aim to prevent the clients from switching to other clouds or providers, resulting from the diversity of protocols and the absence of a naming convention along with Api's unified structure.
Our goal is to provide a solution for these challenges that face the emergence of the multicloud, the solution will target SOAP and REST APIs and perform the transformation from REST to SOAP and vice versa along with unified web description language.
Our system will play the role of broker because it will be in the middle of the data exchange of microservices from different cloud providers and give them the proper transformation of both format and data wise along with preserving the data sent end received.
This solution will help escape vendor lock-in along with giving cloud users the best experience by giving them the freedom to choose any service provider they want.

Keywords :
SOAP, REST, WSDL, WADL, Semantic web , Cloud , Multicloud, Web description languages.

الملخص

الحوسبة السحابية هي نموذج حوسبة ناشئ، يوفر توافر خدمة عالية وقابلية عالية للتطوير بالإضافة إلى تكاليف استخدام منخفضة. وقد شجع هذا الشركات والمستخدمين الأفراد على تبني التكنولوجيا السحابية. ومع ذلك، لا يزال الافتقار إلى قابلية التشغيل البيني للخدمة وقابليتها للنقل (المعروف أيضًا باسم قضية قفل البائع) مستمرًا.

يحدث قفل البائع بسبب مزودي الخدمة السحابية الذين يهدفون إلى منع العملاء من التحول إلى السحب أو مقدمي الخدمات الآخرين، نتيجة لتنوع البروتوكولات وغياب اتفاقية التسمية جنبًا إلى جنب مع هيكل Api الموحد.

هدفنا هو توفير حل لهذه التحديات التي تواجه ظهور السحب المتعدد، وسيستهدف الحل SOAP و REST APIs وأداء التحول من REST إلى SOAP والعكس صحيح جنبًا إلى جنب مع لغة وصف الويب الموحدة.

سيلعب نظامنا دور البرامج الوسيطة إذا استطعنا قول ذلك لأنه سيكون في منتصف تبادل البيانات للخدمات الدقيقة من مزودي الخدمات السحابية المختلفين ويمنحهم التحويل المناسب لكل من التنسيق والبيانات إلى جانب الحفاظ على البيانات المرسلة الطرف المستلم.

سيساعد هذا الحل في الهروب من قفل البائع جنبًا إلى جنب مع منح مستخدمي السحابة أفضل تجربة من خلال منحهم حرية اختيار أي مزود خدمة يريدونه.

كلمات مفتاحية:

الحوسبة السحابية، الوسيط، الوصف، الاكتشاف، الاختيار، البرمجيات كخدمة، انطولوجيا, التشابه الدلالي.

Résumé

L'informatique en nuage est un paradigme informatique émergent, qui offre un service de haute disponibilité ainsi qu'un un coût d'utilisation minimale, ce qui a encouragé les entreprises et les utilisateurs particuliers à adopter cette technologie. Toutefois, le manque d'interopérabilité et de portabilité (aussi appelé verrouillage du fournisseur) des services créent le problème. Le verrouillage du fournisseur est causé par les fournisseurs des services cloud qui visent à empêcher les clients de basculer vers d'autres clouds ou fournisseurs, résultant de la diversité des protocoles et de l'absence de convention de dénomination avec la structure unifiée d'Api.
Notre objectif est d'apporter une solution à ces défis qui font face à l'émergence du multicloud, la solution ciblera les "API" SOAP et REST et effectuera la transformation de REST en SOAP et vice versa avec un langage de description Web unifié. Notre système jouera le rôle d'un middleware, si on peut le dire, car il sera au milieu de l'échange de données des microservices de différents fournisseurs de cloud et leurs donner la transformation appropriée du format et des données tout en préservant les données envoyées et reçues.
Cette solution aidera à échapper à l'enfermement des fournisseurs tout en offrant aux utilisateurs du cloud la meilleure expérience en leurs donnant la liberté de choisir le fournisseur de services qu'ils souhaitent.

Mots clés :
SOAP, REST, WSDL, WADL, Web sémantique , Cloud , Multicloud, language de description.

Table of contents

Table of Figures

List of Algorithms

List of Tables

List of acronyms and abbreviations

General Introduction 1

I General view on cloud computing and multi-cloud 3

I.1 Introduction : 3

I.2 Cloud computing definition : 3

I.3 Cloud computing architecture : 3

I.3.1 Front End parts : 4

I.3.2 Back End parts : 4

I.4 Cloud computing characteristics : 5

I.4.1 On-demand self-service : 5

I.4.2 Broad network access : 5

I.4.3 Resource pooling : 5

I.4.4 Rapid elasticity : 5

I.4.5 Measured service : 5

I.5 Types of cloud services : 6

I.5.1 Software as a Service ("SaaS") : 6

I.5.2 Platform as a Service ("PaaS") : 6

I.5.3 Infrastructure as a Service ("IaaS") : 7

I.6 Type of cloud computing : 7

I.6.1 Public clouds : 7

I.6.2 Private clouds : 7

I.6.3 Hybrid clouds : 8

I.6.4 Multiclouds : 8
I.6.4.1 Multi-cloud architecture : 8
I.6.4.2 Reasons for adopting the multicloud : 9
I.7 Advantages and disadvantages of cloud computing : 10
I.7.1 Advantages : 10
I.7.2 Disadvantages : 11
I.8 Conclusion 12

II Cloud services and API's deployment 13
II.1 Introduction : 13
II.2 Cloud virtualization : 13
II.2.1 Definition : 14
II.2.2 Type of virtualization : 15
II.2.3 Relation between cloud computing and virtualization : 17
II.2.4 Docker : 18
II.2.4.1 Definition : 18
II.2.4.2 Comparing containers and virtual machines : 18
II.3 Software architecture : 19
II.3.1 Monolithic architecture : 19
II.3.1.1 Definition : 19
II.3.1.2 Advantages of the monolithic application : 20
II.3.1.3 Limitation of the monolithic application : 20
II.3.2 "SOA"(Service Oriented Architecture) : 21
II.3.2.1 Definition : 21
II.3.2.2 How it works : 21
II.3.2.3 Advantages of SOA : 22
II.3.2.4 Challenges of SOA : 22
II.3.3 Microservices : 22
II.3.3.1 Definition : 22
II.3.3.2 Microservice architecture : 23
II.3.3.3 Characteristics of Microservices : 23
II.3.4 Software architecture comparison : 24
II.4 API's (Application Programming Interface) : 25
II.4.1 Definition : 26
II.4.2 Types of API's : 26
II.4.3 Types of API architectures : 27
II.4.3.1 SOAP(Simple Object Access Protocol) : 27
II.4.3.2 REST API's : 30
II.4.3.3 GraphQL : 31

II.4.3.4 Comparative studie : 34
II.5 Communication in cloud : 34
II.5.1 Definition : 34
II.5.2 Communication in cloud : 35
II.5.3 Synchronous communication : 35
II.5.4 Asynchronous communication : 36
II.6 Messaging softwares : 37
II.6.1 Message brokers : 37
II.6.2 Message queues : 37
II.6.3 AMQP (advanced message queue protocol) : 38
II.6.4 "RabbitMQ" : 38
II.6.4.1 RabbitMQ architecture : 39
II.6.4.2 Advantages of RabbitMQ : 39
II.6.4.3 Disadvantages of RabbitMQ : 40
II.6.5 "KafkaMQ" : 40
II.6.5.1 Kafka architecture : 40
II.6.5.2 Kafka advantages : 42
II.6.5.3 Kafka disadvantages : 43
II.6.6 Comparaison between kafkaMQ and RabbitMQ : 43
II.7 Conclusion : 44

III Cloud computing portability **45**
III.1 Introduction : 45
III.2 Interoperability : 45
III.2.1 Definition : 45
III.2.2 Types of interoperability : 45
III.2.3 Benefits of interoperability : 46
III.2.4 Exciting standard : 46
III.2.5 Interoperability related works : 47
III.2.5.1 Solution of Bipin Upadhyaya et al [83] : 48
III.2.5.2 Solution of Bouzerzour and Slimani [25] : 48
III.2.5.3 Solution of Sean Kennedy et al [48] : 49
III.2.5.4 Solution of Majda Elhozmari and Ahmed Ettalbi [57] : 50
III.2.5.5 Solution of William Appleton [6] : 51
III.2.6 Interoperability solutions and standards table : 51
III.2.7 Discussion : 53
III.3 Portability : 53
III.3.1 Definition : 53
III.3.2 Types of portability : 53

III.3.3 benefits of portability : 54
III.3.4 Portability related works : 54
III.3.4.1 Solution of Diego Serrano and Eleni Stroulia [77] : 55
III.3.4.2 Solution of Cassia Trojahn et al [35] : 55
III.3.4.3 Solution of Doina Caragea and Tanveer Syeda-Mahmood [26] : 55
III.3.4.4 Solution of Giuseppina Cretella and Beniamino Di Martino [27] : 55
III.3.4.5 Solution of Yong Jo Lee and Jae Soo Kim [52] : 57
III.3.4.6 Solution of Tanveer Syeda-Mahmood et al [81] : 58
III.3.5 Portability solutions and standards table : 58
III.3.6 Discussion : 60
III.3.7 Conclusion : 60

IV Design of our solution **61**
IV.1 Introduction : 61
IV.2 Needs analysis : 61
IV.2.1 The actors : 61
IV.3 Our proposed solution : 63
IV.3.1 Interoperability solution : 63
IV.3.1.1 General explanation of the solution : 64
IV.3.1.2 Architechture of the solution : 64
IV.3.1.3 Unification of web service description : 66
IV.3.1.4 REST to SOAP transformation : 73
IV.3.1.5 SOAP to REST transformation : 75
IV.3.2 Portability solution : 76
IV.3.2.1 General explanation of the solution : 76
IV.3.2.2 The measure of syntactic similarity Methods : 77
IV.3.2.3 The measure of semantic similarity methods : 78
IV.3.2.4 Result aggregation : 79
IV.3.3 Combining the solutions : 80
IV.4 Conclusion : 81

V Implementation of the proposed solution **82**
V.1 Introduction : 82
V.2 Hardware resources : 82
V.3 Development tools and environment : 82
V.4 Our implementation : 84
V.5 Comparing our work to the related works : 92
V.6 Conclusion : 92

Conclusion et perspectives **93**

Bibliography

Table of Figures

I.1 The Structure of cloud computing architecture [15] 4
I.2 Types of cloud services. [88] 6
I.3 Multicloud architecture. [79] 9

II.1 Virtualization origins. [67] 14
II.2 Types of virtualization. 15
II.3 Software virtual machines.[28] 16
II.4 Hardware virtual machines.[28] 16
II.5 Virtual OS/containers virtual machines. [28] 17
II.6 Monolithic flow on the example of application. 20
II.7 Basic SOA model.[56] 21
II.8 Microservice flow on the example of application. 23
II.9 Types of APIs [30] 26
II.10 SOAP architecture. [49] 28
II.11 SOAP message structure. [40] 29
II.12 Simple architect of GraphQL.[43] 32
II.13 Gateway GraphQL architect.[43] 32
II.14 Hybrid GraphQL architect.[43] 33
II.15 Schramm's Model of communication. [53] 35
II.16 Synchronous communication. [12] 36
II.17 Asynchronous communication. [12] 36
II.18 Message broker.[23] 37
II.19 Message queue.[23] 38
II.20 RabbitMQ (AMQP) architecture.[32] 39
II.21 The typical stream processing pipeline. [51] 41
II.22 Kafka architecture. [32] 42

III.1 REST/SOAP converter architecture . [83] 48
III.2 Interoperability model CIPiMo. [25] 49

III.3 STORHM architect . [48] 50
III.4 The proposed solution for cloud standardization. [57] 51
III.5 Architecture of Giuseppina Cretella and Beniamino Di Martin. [27] 56
III.6 Architecture of Yong Jo Lee and Jae Soo Kim. [52] 57

IV.1 Use case diagram. 62
IV.2 Solution architecture. 65
IV.3 WADL class diagram. 66
IV.4 WSDL class diagram. 67
IV.5 WSDL entered by the service provider. 69
IV.6 Mapping WSDL file information inside WSDL object. 71
IV.7 Generated WADL object from WSDL. 72
IV.8 Generated WADL object from WSDL. 73
IV.9 REST request for sayHello servive through our converter. 74
IV.10 XML version of REST request for sayHello servive through our converter. 74
IV.11 SOAP version of REST request. 75
IV.12 SOAP response from SOAP service. 75
IV.13 REST version of SOAP response from SOAP service. 76

V.1 System microservice architecture. 84
V.2 API-gateway docker image creation. 85
V.3 Docker image list 85
V.4 Lunching service Registry microservice. 86
V.5 Running containers list. 86
V.6 Eureka registry server dashboard. 87
V.7 Temperature converter WSDL. 88
V.8 Publishing WSDL file. 89
V.9 Generated WADL. 89
V.10 Sending REST GET method. 90
V.11 Sending SOP request. 91
V.12 Going from REST to SOAP. 91
V.13 Going from SOAP to REST. 92

List of Algorithms

1 Transforamtion from WSDL to WADL pseudo algorithm. 69
2 REST to SOAP transformation algorithme 74
3 REST to SOAP transformation algorithm 75
4 Portability algorithm 80

List of Tables

II.1 Virtual machine versus Docker container. 19
II.2 Comparison of different architectures. 24
II.3 Details of services of both SOA and microservices-based applications. 25
II.4 Feature comparison table 44

III.1 Interoperability standards. 47
III.2 Interoperability solutions comparison table. 52
III.3 Portability solutions comparison table. 59

IV.1 Actors and thier roles. 61
IV.2 Textual description of use case. 63
IV.3 Mapping rules from WSDL to WADL. 68
IV.4 Dice method example. 77
IV.5 Jaccard Index method example. 78

V.1 Evaluation of our solution. 92

List of acronyms and abbreviations

Abbreviation	**Definition**
ANSI	American National Standards Institute
AWS	Amazon Web Services
API	Application Programming Interface
ActiveMQ	Active Message Queuing
AMR	Advanced Market Research
AMQP	Advanced Message Queue Protocol
ACK	Acknowledge
CSP	Communications Service Providers
CPU	Central Processing Unit
CMfg	Cloud Manufacturing
DNS	Domain Name System
ESBs	Enterprise Service Buses
GB	Gigabyte
GraphQL	Graph Query Language
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HDFS	Hadoop Distributed File System
IBM	International Business Machines
IaaS	Infrastructure as a Service
IT	Information technology
ISO	International Organization for Standardization
IDE	Integrated development environment
JSON	JavaScript Object Notation
KafkaMQ	KafKa Message Queue
LAN	Local Area Network
MAC	Medium Access Control

MBs	Megabits per second
MIaaS	Monitoring- Integration-As-A-Service
MPaaS	Mobile Platform as a Service
MSaaS	Managed software as a service
MQTT	Message Queuing Telemetry Transport
NoSQL	Not Only SQL
OS	Operating System
OAuth	Open Authorization
P2P	Peer-to-peer
PaaS	Blockchain-based service composition
QEMU	Quick Emulator
QoS	Quality of Service
RAM	Random Access Memory
REST	Representational State Transfer
RPC	Remote Procedure Call
RabbitMQ	Rabbit Message Queue
REDIS	Remote Dictionary Server
SSH	Secure Shell
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language
STOMP	Simple Text Oriented Message Protocol
SaaS	Software as a Service
StoRHm	"SOAP" to RESTful "HTTP" mapping
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
UDDI	Universal description identifiers
VPN	Virtual private network
VPS	Virtual Private Server
VMM	Virtual Machine Monitor
VMS	Vehicle Management System
VM	Virtual Machine
WAR	Web Application Resource
WSDL	Web Services Description Language
WADL	Web Application Description Language
WS	Web Services
WAN	Wide Area Network
XML	Extensible Markup Language

General Introduction

The influence of cloud computing on business and end users cannot be overstated : the ubiquitous of software that runs on cloud networks has altered many aspects of daily life. By embracing cloud computing, entrepreneurs and organizations may save expenses and expand their products without having to purchase and manage all of the gear and software. Independent developers now have the ability to establish internationally accessible apps and online businesses. Researchers may now exchange and analyze data on previously unimaginable scales. Furthermore, internet users may easily access software and storage in order to produce, distribute, and save digital media in numbers that much exceed the computational power of their own devices.[33]

Problematic :

User demands for increasingly complex cloud services are driving service providers to work together to deliver effective services. However, cloud services from different providers are delivered with technologies and APIs different. Some, for example, use "REST" APIs while others use APIs SOAP.
The problem of this work consists in defining a broker-based solution for unification communication interfaces in the multicloud.

Work objectives :

In this section, we will present our work objectives : Study of communication and APIs in the multicloud then do comparative study of existing solutions for communication and APIs in the multi-cloud, after that we will define a broker-based solution for unifying communication interfaces in the multicloud, finally, we will validate the proposed solution through experimentation through a Java EE application.

Organization of the report :

This word done in this report is divided into five chapters :

The first chapter :

In this chapter, we introduced what is cloud computing along with its types and architecture from a software engineer point of view and the deployment model available then go through their characteristics and a brief view of Multicloud.

The second chapter :

This chapter is designed to pave the way and provide additional information needed to understand this work and all Tribal gains to keep up with the cloud environment along with API models and comparative study between them.

The third chapter :

Is dedicated to mentioning all the related works concerning our work with comparative study and discussion about all of them.

The fourth chapter :

In this chapter, we introduce our solution concerning our problem.

The fifth chapter :

His is the chapter in which we introduce our implementation of the proposed solutions that we mention in the fourth chapter.

In the end, we end this work with the conclusion of this work along with our perspective and future works .

Chapter I

General view on cloud computing and multi-cloud

I.1 Introduction :

The use of Internet and new technologies nowadays, for business and for the current users, is already part of everyday life. Any information is available anywhere in the world at any time. That was not possible few years ago. Nowadays it have arisen a lot of possibilities of access to public and private information like internet speed access or the deployment of mobile dispositive that allow the connection to Internet from almost everywhere.
In this chapter, we will give an overview on cloud computing including it's architecture , deployment mode and types .

I.2 Cloud computing definition :

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models .[61]

I.3 Cloud computing architecture :

This section describes the architectural, commercial and various models of operation- cloud computing.
The architecture of cloud computing is divided into two parts : **Front End Architecture** and **Back End Architecture** as shown in Figure I.1 , separated but dependent on each other and linked together through the network that is generally the internet.

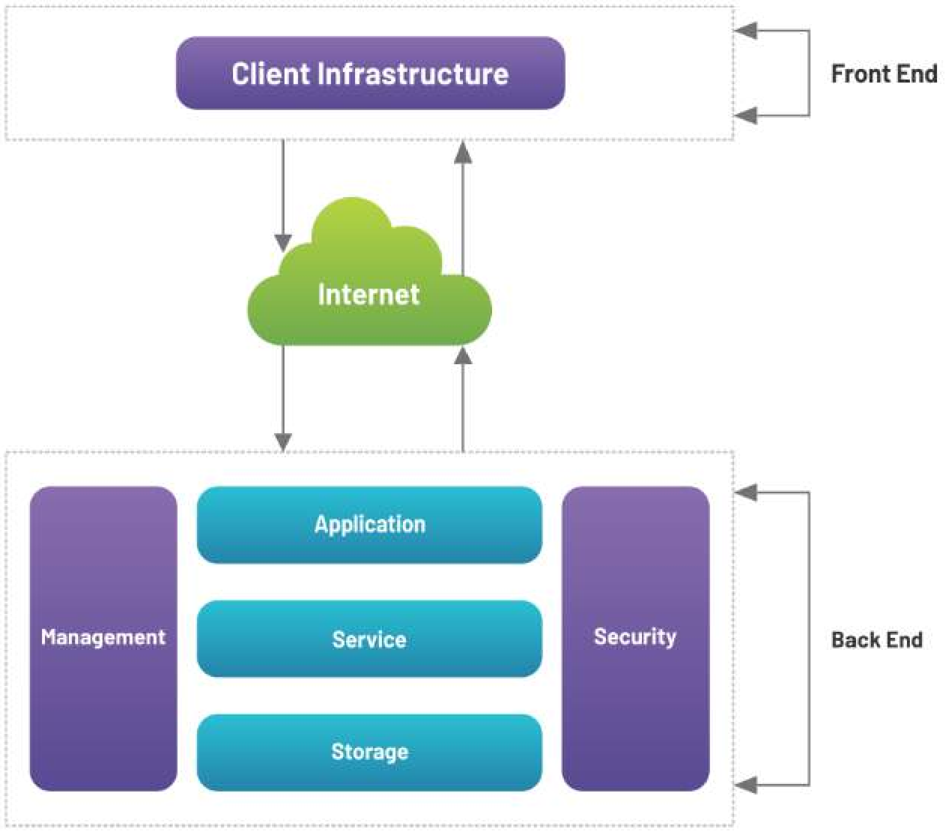


FIGURE I.1 – The Structure of cloud computing architecture [15]

I.3.1 Front End parts :

Front end architecture is a term used to denote any user-facing part of cloud computing architecture. This is the part that the end user interacts with, and it is comprised of subcomponents that make up the user experience. Front end architecture generally takes the form of a user interface and is an integral part of how the user interacts with cloud computing software.
Most working IT professionals today will mainly interact with front-end cloud software architecture. Examples of front-end architecture include web browsers, local networks and common web apps. Gmail, a popular cloud service used by millions daily, has its front-end architecture in the form of the web application. The interface allows users to access the services offered by the Gmail architecture as a whole.[15]

I.3.2 Back End parts :

Back end architecture is the part of the cloud computing architecture that powers the front-end architecture. This includes the core components of the system such as hardware and storage, and is generally located in a server farm in a geographically distant location. Back end architecture is taken care of by the cloud service provider offering cloud software. Prominent cloud providers such as "AWS" generally have robust backend architecture to ensure continuous uptime with low latency. In addition to this, powerful front-end cloud architecture plays a role in creating a dependable and easy-to use product.[15]

I.4 Cloud computing characteristics :

Cloud computing has many characteristics we will mention a few of them in this section.

I.4.1 On-demand self-service :

A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.[61]

I.4.2 Broad network access :

Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g.,mobile phones, tablets, laptops, and workstations). [61]

I.4.3 Resource pooling :

The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth.[61]

I.4.4 Rapid elasticity :

Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.[61]

I.4.5 Measured service :

Cloud systems automatically control and optimize resource use by leveraging a metering capability1 at some level of abstraction appropriate to the type of service (e.g.,storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.[61]

I.5 Types of cloud services :

Cloud computing it's based in the offer of services, we found 3 different kinds of services, as the Figure shows I.2 :

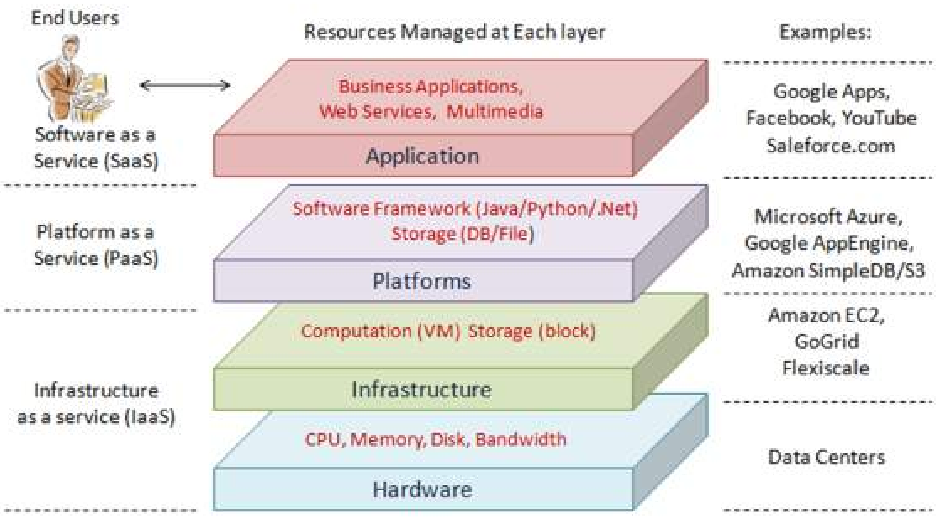

End Users
Software as a Service (SaaS)
Platform as a Service (PaaS)
Infrastructure as a service (IaaS)
Resources Managed at Each layer
Business Applications, Web Services, Multimedia
Application
Software Framework (Java/Python/.Net) Storage (DB/File)
Platforms
Computation (VM) Storage (block)
Infrastructure
CPU, Memory, Disk, Bandwidth
Hardware
Examples:
Google Apps, Facebook, YouTube Saleforce.com
Microsoft Azure, Google AppEngine, Amazon SimpleDB/S3
Amazon EC2, GoGrid Flexiscale
Data Centers

FIGURE I.2 – Types of cloud services. [88]

I.5.1 Software as a Service ("SaaS") :

These services are applications over Internet. Normally the user can run these applications using a web-browser. User abstract totally about the hardware and software that is using and simply access to a interface with a web browser and from there he have access to some information and functionalities. It's dedicated to current users ; an example to this kind of services may be Google Docs .[88]

I.5.2 Platform as a Service ("PaaS") :

These services are focused on the deployment of applications or services online letting to the developer manage the hardware or software necessary, including also a solution stack. This service includes all the life-cycle of the deployment of application/ service such as design, implementation, testing, deployment, integrity with databases, etc. [88] There are three characteristic points in this service :
• **Services for deployment, testing and maintenance of applications.**
• **Multi-user architecture, in other words, scalability.**
• **Collaborative tools.**
An example of these services is Google App engine.

I.5.3 Infrastructure as a Service ("IaaS") :

These services are focused to offer a computer infrastructure. All the servers, connections, software and other resources are offered by the providers. And the users see it like an entire infrastructure hosted in the same organization .[88]

I.6 Type of cloud computing :

According to redhat[1] there are 4 main types of cloud computing : private clouds, public clouds, hybrid clouds, and multi-clouds, we will talk briefly about them :

I.6.1 Public clouds :

Are cloud environments typically created from IT infrastructure not owned by the end user. Some of the largest public cloud providers include Alibaba Cloud, Amazon Web Services (AWS), Google Cloud, "IBM" Cloud, and Microsoft Azure.
Traditional public clouds always ran off-premises, but today's public cloud providers have started offering cloud services on clients' on-premise data centers. This has made location and ownership distinctions obsolete.
All clouds become public clouds when the environments are partitioned and redistributed to multiple tenants. Fee structures aren't necessary characteristics of public clouds anymore, since some cloud providers (like the Massachusettes Open Cloud) allow tenants to use their clouds for free. The bare-metal IT infrastructure used by public cloud providers can also be abstracted and sold as IaaS, or it can be developed into a cloud platform sold as PaaS .

I.6.2 Private clouds :

Are loosely defined as cloud environments solely dedicated to a single end user or group, where the environment usually runs behind that user or group's firewall. All clouds become private clouds when the underlying IT infrastructure is dedicated to a single customer with completely isolated access.
But private clouds no longer have to be sourced from on-prem IT infrastructure. Organizations are now building private clouds on rented, vendor-owned data centers located off-premises, which makes any location and ownership rules obsolete .

1. https : //www.redhat.com/en/topics/cloud-computing/ public-cloud-vs-private-cloud-and-hybrid-cloud

I.6.3 Hybrid clouds :

A hybrid cloud is a seemingly single IT environment created from multiple environments connected through local area networks (LANs), wide area networks ("WANs"), virtual private networks ("VPNs"), and/or APIs.
The characteristics of hybrid clouds are complex and the requirements can differ, depending on whom you ask. For example, a hybrid cloud may need to include : **At least 1 private cloud and at least 1 public cloud, 2 or more private clouds, 2 or more public clouds.**
A bare-metal or virtual environment connected to at least 1 public cloud or private cloud But every IT system becomes a hybrid cloud when apps can move in and out of multiple separate—yet connected—environments. At least a few of those environments need to be sourced from consolidated IT resources that can scale on demand. And all those environments need to be managed as a single environment using an integrated management and orchestration platform.

I.6.4 Multiclouds :

Multiclouds are a cloud approach made up of more than 1 cloud service, from more than 1 cloud vendor—public or private. All hybrid clouds are multiclouds, but not all multiclouds are hybrid clouds. Multiclouds become hybrid clouds when multiple clouds are connected by some form of integration or orchestration.
A multicloud environment might exist on purpose (to better control sensitive data or as redundant storage space for improved disaster recovery) or by accident (usually the result of shadow IT). Either way, having multiple clouds is becoming more common across enterprises that seek to improve security and performance through an expanded portfolio of environments.[75]

I.6.4.1 Multi-cloud architecture :

To help deal with some of the issues mentioned above, the next logical progression in cloud computing is in multi-cloud computing or cloud systems that utilize numerous cloud networks and services simultaneously . In short, multi-cloud systems simply use more than one "CSP", though they come in many subcategories that will be described below. This is similar to the definition of cross-cloud architectures, defined as systems that span across multiple provisioning boundaries. Though the exact distinction between multi-clouds and cross clouds is lacking (as described in further detail below), it can be understood that in multi-clouds, the user or business in question utilizes different cloud services for different applications in their business. For example, they may store data on an AWS cloud, share documents on the Google Cloud platform, and perform data analysis on yet another cloud. On the other hand, cross-cloud architectures are designed to make the transfer of data and utilization of apps across the clouds more streamlined and cohesive.[42]
Multicloud refers to the presence of more than 1 cloud deployment of the same type (public or private), sourced from different vendors. Hybrid cloud refers to the presence of multiple

deployment types (public or private) with some form of integration or orchestration between them. [75]
Multi-cloud architecture is shown in Figure I.3. Whenever, a client requests for a service, it is parsed by the client side interface and passed on to CSP side interface after processing by the middleware. The service request will be granted by one of the clouds based on the availability, underlying algorithms and request criteria. Selection of proper cloud on the basis of service request requires sophisticated technologies[79].

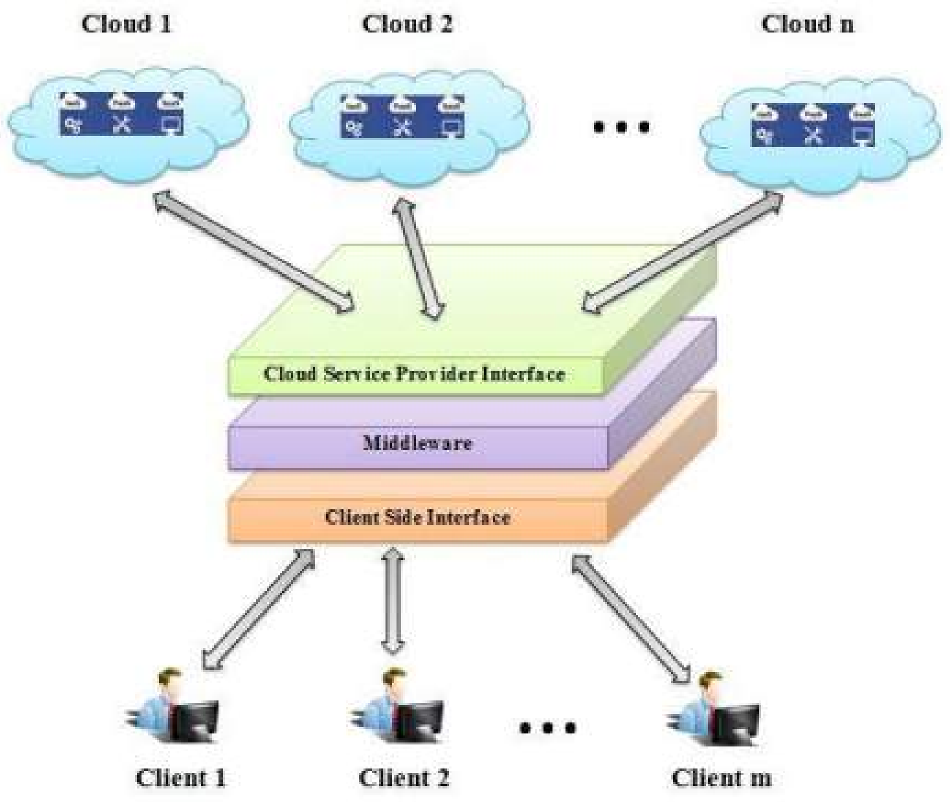


FIGURE I.3 – Multicloud architecture. [79]

I.6.4.2 Reasons for adopting the multicloud :

There are various rationales for using multi-clouds over a single cloud approach nowadays, we will mention some of them in this section :
•**Shadow IT :** Shadow IT is becoming a reality that contributes to multiclouds. Hardware or software deployed independently from the central IT team may become large enough to warrant more oversight. At that point, migrating the infrastructure and data to a preferred system (let's pretend we're talking about public clouds here) might be out of the question. That shadow IT deployment is simply aggregated as part of the enterprise's existing clouds—thereby creating a multicloud.[75]
•**Flexibility :** You might find the perfect cloud solution for 1 aspect of your enterprise—a pro-

prietary cloud fine-tuned for hosting a proprietary app, an affordable cloud perfect for archiving public records, a cloud that scales broadly for hosting systems with highly variable use rates—but no single cloud can do everything. (Or, rather, no single cloud can do everything well.)[75]
•**Proximity :** To reduce poor response times for cloud users thousands of miles away from a company's headquarters, some workloads could be hosted by regional cloud providers that operate closer to where the users are. This solution lets the enterprise maintain high availability and adhere to data sovereignty laws—protocols that subject data to the regulations of the country in which that data is located.[75]
•**Failover :** Multicloud environments help protect enterprises from outages. As a failover solution, multicloud allows enterprises to have an available, highly scalable backup for data, workflows, and systems if—or perhaps when, as Murphy's Law suggests—your primary cloud goes dark.[75]

I.7 Advantages and disadvantages of cloud computing :

In the following section we are presenting the main advantages and disadvantages of Cloud Computing.[20]

I.7.1 Advantages :

The most valuable points of cloud computing are [20] :
• **Cost efficiency :**
Cloud computing is probably the most cost efficient method to use, maintain and upgrade. Traditional desktop software costs companies a lot, in terms of finance. Adding up the licensing fees for multiple users can prove to be very expensive for the establishment concerned. The cloud, on the other hand, is available at much cheaper rates and hence, can significantly lower the company's IT expenses. Besides, there are many one(time(payment, pay(as(you(go and other scalable options available, which makes it very reasonable for the company in question.[20]
• **Backup and Recovery :**
Since all the data is stored in the cloud, backing it up and restoring the same is relatively much easier than storing the same on a physical device. Furthermore, most cloud service providers are usually competent enough to handle recovery of information. Hence, this makes the entire process of backup and recovery much simpler than other traditional methods of data storage.[20]
• **Easy access to information :**
Once the users register in the cloud, they can access the information from anywhere, where there is an Internet connection. This convenient feature lets users move beyond time zone and geographic location issues.[20]
• **Almost Unlimited Storage :**

Storing information in the cloud gives you almost unlimited storage capacity.[20]
• Easier scale of services :
It makes it easier for enterprises to scale their service according to the demand of clients.[20]
•Deliver new services :
It makes possible new classes of applications and deliveries of new services that are interactive in nature.[20]

I.7.2 Disadvantages :

Experts about cloud computing identifies next points as possible problems about the use of cloud computing [20] :
• Technical issues :
Though it is true that information and data on the Cloud can be accessed any time and from anywhere, there are moments when the system can have some serious malfunction. Businesses should be aware of the fact that this technology is always prone to outages and other technical issues. Even the best Cloud service providers run into this kind of trouble, in spite of keeping up high standards of maintenance.[20]
•Security in the cloud :
The other major issue of Cloud is represented by security. Before adopting this technology, beneficiaries should know that they will be surrendering all their company's sensitive information to a third(party cloud service provider. This could potentially impose a great risk to the company.
Hence, businesse need to make sure that they choose the most reliable service provider, who will keep their information totally secure. Switching to the cloud can actually improve security for a small business, as mentioned by Michael Redding, managing director of Accenture Technology Labs. "Because large cloud computing companies have more resources, he says, they are often able to offer levels of security an average small business may not be able to afford implementing on its own servers" (Outsource IT Headaches to the Cloud (The Globe and Mail)).[20]
• Cost :
At first glance, a cloud computing application may appear to be a lot cheaper than a particular software solution installed and run in(house. Still, the companies need to ensure that the cloud applications have all the features that the software does and if not, to identify which are the missing features important to them.
A total cost comparison is also required. While many cloud computer vendors present themselves as utility(based providers, claiming that they only charge for what customers use, Gartner says that this isn't true ; in most cases, a company must commit to a predetermined contract independent of actual use. Companies need to look closely at the pricing plans and details for each application.[20]

I.8 Conclusion

Cloud computing is a blossoming technology with numerous applications in many industries, including remote computing and storage. Though vendor lock-in and cyber security are major concerns hybrid clouds, multi-clouds, and federation clouds may address some of these problems by providing users with alternatives in the case of scheduled maintenance, breaches, or shut-downs, though each has their own benefits and disadvantages. Hybrid systems are easily customised to a given application, however less transferable, and often used for only one task whereas multi-clouds and federated clouds are suited for businesses that require multiple tasks or services. Future work should incorporate multi-cloud paradigms and combine them with other technologies such as machine learning and big data, as these technologies can either be used to solve some of the current issues with cloud computing, such as increasing security, or can be used for entirely new methods of analysis.

Chapter II

Cloud services and API's deployment

II.1 Introduction :

Cloud computing has become one of the key considerations both in academia and industry. Cheap, seemingly unlimited computing resources that can be allocated almost instantaneously and pay-as-you-go pricing schemes are some of the reasons for the success of Cloud computing industry.[74]
The Cloud Computing environment, however, is plagued by many issues hindering adoption. One such issue is vendor lock-in, forcing the Cloud users to adhere to one service provider in terms of data and application logic. Semantic Web has been an important research area that has seen significant attention from both academic and industrial researchers. One key property of Semantic Web is the notion of interoperability and portability through high-level models. Significant work has been done in the areas of data modeling, matching, and transformations.[74]
The issues the Cloud computing community is facing now with respect to portability of data and application logic are the same issue the Semantic Web community has been trying to address for some time.[74]
This chapter focus on all basic concepts of cloud interface portability and the technologies that forced its appearance.

II.2 Cloud virtualization :

By the 1970s, mainframe users saw the first implementation of virtualization and symmetric multiprocessing, where different users could use the resources of a single machine to execute different processes concurrently. But as the Internet continued to expand, organizations, researchers and academicians began to grapple with concerns relating to the high cost of computers at the time and only sparsely intermittent computational needs; they could not justify investing at high costs in a computer that would be idle for most of the time. To assuage

theseconcerns, entrepreneurs came up with the idea of “renting” time, making it possible for organizations and users to either own or subscribe to computing resources at much lower costs.[29]
It now became possible for users to access large-scale mainframe computer systems from thin clients/terminal machines, often referred to as “static terminals” because they were used mainly for communications but had no internal processing capabilities.
This idea helped to enhance the efficiency of expensive mainframe systems and reduce idle periodsby allowing multiple users to share both the physical access to the computer from multiple terminals as well as the processing resources ("CPU" time) ; thus, allowing for greater returns on investment for companies that practiced such. It is this technology that evolved through various nomenclatures – such as Remote Job Entry in the 1950s, Shared and Dedicated Web Hosting (which are forms of Virtual Web Hosting) around 1995 to 1997, Virtual Private Server ("VPS") Hosting around 1998, Grid/Utility Computing – to become Cloud Computing (CC) about three decades later.[68]
Figure II.1 below shows a simple view of Cloud virtualization :

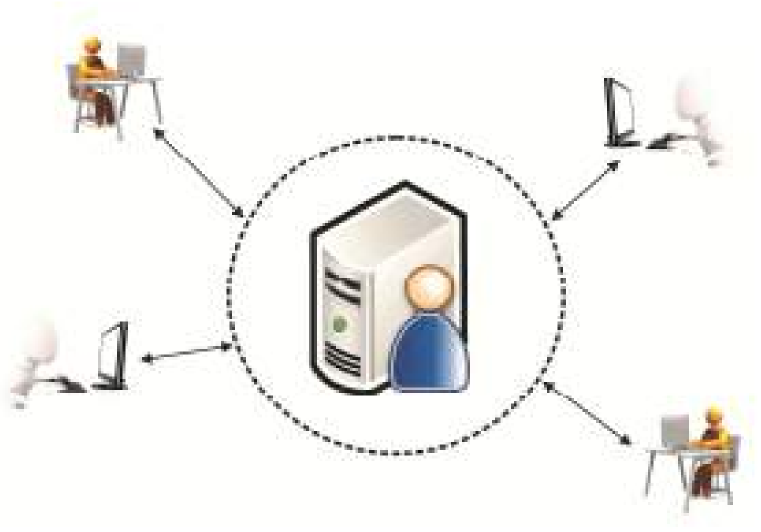


FIGURE II.1 – Virtualization origins. [67]

II.2.1 Definition :

Virtualization, in the simplest and easiest words, is the abstraction of computer resources. virtualization is a component within cloud computing that allows the separation of the operating system from the hardware on which it is working. virtualization has the ability to allow a single physical resource server as multiple virtual resources and even make multiple physical resources function as a single virtual resource. virtualization has brought a great change in the working of IT organizations in all ways.[44]
Goldberg R. P defined Virtual machines as :"A system...which...is a hardware-software duplicate

of a real existing machine, in which a non-trivial subset of the virtual machine's instructions execute directly on the host machine..."[58]

II.2.2 Type of virtualization :

Virtual machines are implemented in various forms. Mainframe, open source, para virtualization, and custom approaches to virtual machines have been designed over the years [19]. Complexity in chip technology and approaches to solving the x86 limitations of virtualization have led to three different variants of virtual machines as it is shown in the Figure II.2 :

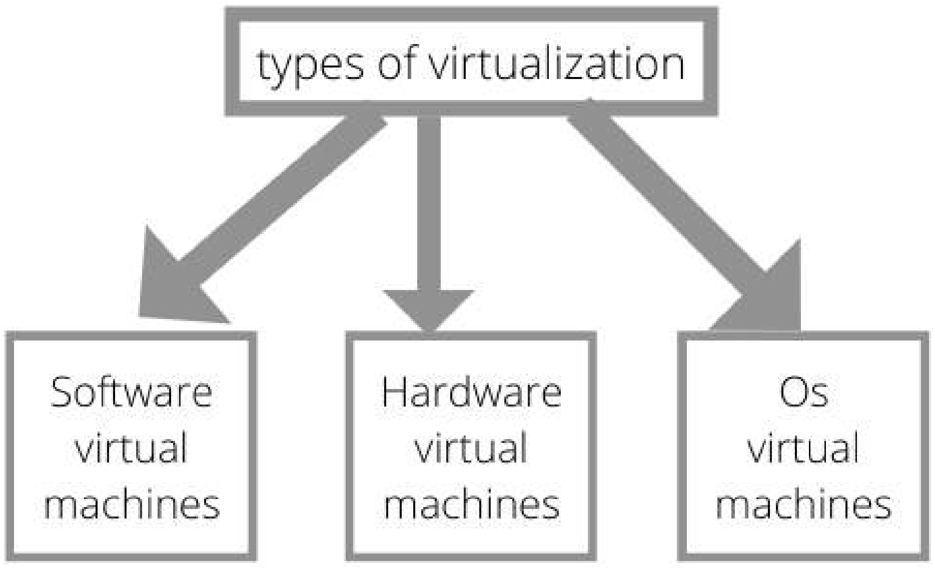


FIGURE II.2 – Types of virtualization.

— Software virtual machines (See Figure II.3), which manage interactions between the host operating system and guest operating system (e.g., Microsoft Virtual Server 2005).[28]

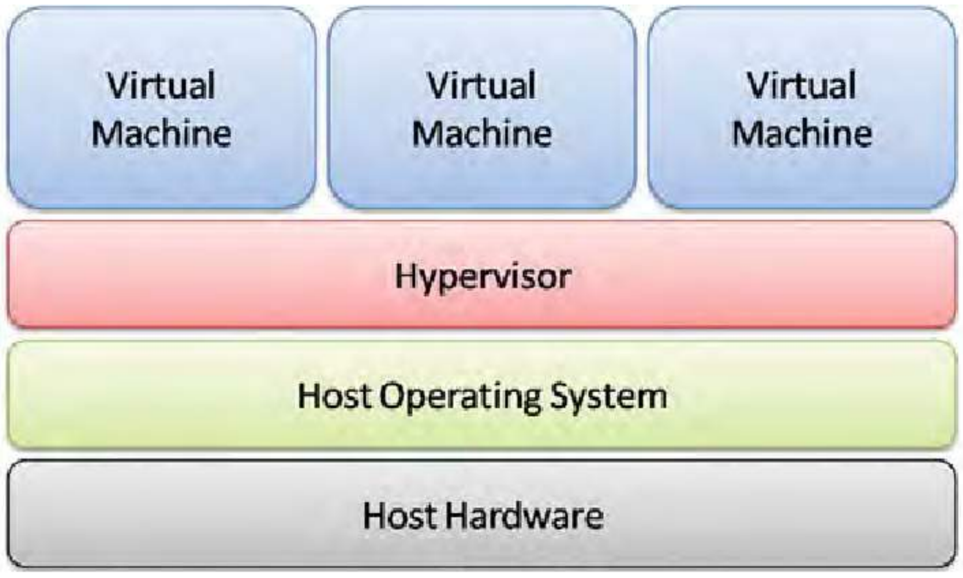


FIGURE II.3 – Software virtual machines.[28]

— Hardware virtual machines (See Figure II.4), in which virtualization technology sits directly on host hardware (bare metal) using hypervisors, modified code, or APIs to facilitate faster transactions with hardware devices (e.g., VMWare ESX).[28]

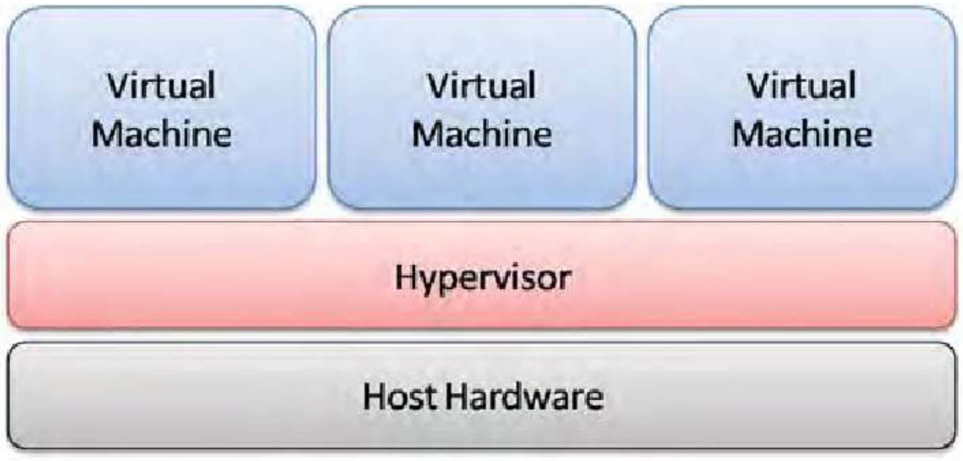


FIGURE II.4 – Hardware virtual machines.[28]

— Virtual OS/containers (See Figure II.5), in which the host operating system is partitioned into containers or zones (e.g., Solaris Zones, BSD Jail).[28]

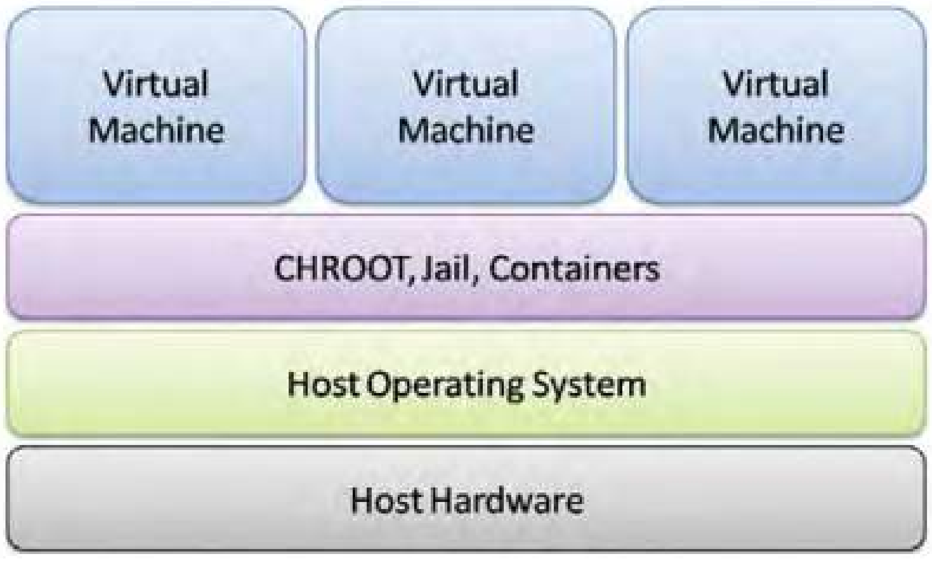


FIGURE II.5 – Virtual OS/containers virtual machines. [28]

II.2.3 Relation between cloud computing and virtualization :

When an end-user uses a computer, what he/she directly interacts with is a view provided by the OS which abstracts all the physical components available. The user gets information that he is interested in from the view.
It can be described that what virtualization does is build up a few different logic views from a physical machine, each of which can be used to interact with a user simultaneously.
Cloud computing requirements of dynamic cutting and distribution of computing resources are owing to virtualization.
Virtualization involves the construction of an isomorphism that maps a virtual guest system to a real host. This isomorphism maps the guest state to the host state. From the angle of resource, virtualization creates plural subsets logically from the complete set of machines.
Each logic view may possess a similar architecture a physical view. When a user interacts with certain resources via a view, he/she doesn't need to know if the view is a drawing of physical resources or just a graph of a logic set of resources. The end users do not see the details of resources, but just pay attention and interact with the logic view provided by the virtualization layer, or "VMM" (Virtual Machine Monitor). These subsets that behave the same as real machines are called "VMs" (Virtual Machines).
Therefore, on the resource virtualization layer storage must be organized as a single logical pool of resources available to users, and on the virtual resource layer, that software must package a set of computing resources and behaviors and present it as an available computing environment

is at the core of what it means to create a virtual machine. The service layer is built upon virtual organizations where the nodes are fully decentralized just like "P2P" systems.[34]

II.2.4 Docker :

As we mentioned there are many types of virtualization but the most suitable one for cloud computing is os/container virtualization thus that is Lightweight compared to the old virtualization technics as was mentioned by Stephen Soltesz et al[80] :" Our experiments indicate that container-based systems provide up to 2x the performance of hypervisor-based systems for server-type workloads and scale further while preserving performance".

II.2.4.1 Definition :

Docker [1] is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.
Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allows you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host. You can easily share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

II.2.4.2 Comparing containers and virtual machines :

As we mentioned before cloud computing requires us to use container-based virtualization. In the following Table II.1, we will mention some of the differences between the VMs and Container-based virtualization :

1. https ://docs.docker.com/ get-started/overview/

	Virtual Machines	**Docker Containers**
Isolation Process Level	Hardware	Operating System
Operating System	Separated	Shared
Boot up time	Long	Short
Resources usage	More	Less
Pre-built Images	Hard to find and manage	Already available for home server
Customised preconfigured images	Hard to build	Easy to build
Size	Bigger because they contain whole OS undemeath	Smaller with only docker engines over the host OS
Mobility	Easy to move to a new host OS	Destroyed and recreated instead of moving.
Creation time	Longer	Within seconds

TABLE II.1 – Virtual machine versus Docker container.

As was stated by Potdar ET al[71] in their paper they have done a comparative performance study between docker container and virtual machine, in this study they used standard benchmark tools such as Sysbench, Phoronix, and Apache benchmark, which include CPU performance, Memory throughput, Storage read/write performance, load test, and operation speed measurement. Before they started their tests they pointed out the differences between their architectures in the table below. as a result Potdar ET al said "It is observed that Docker containers perform better over "VM" in every test, as the presence of "QEMU" layer in the virtual machine makes it less efficient than Docker containers."

II.3 Software architecture :

In the last years, cloud-native architectures have emerged as a target platform for the deployment of microservice architectures. The migration of existing monoliths into cloud-native applications is still in the early phase. we decided in this section to introduce a brief overview of Microservices, but before that, we are going to have a quick overview of monolithic applications and the history that led to Microservices.

II.3.1 Monolithic architecture :

Traditional application design is often called monolithic application, in this section, we will discuss it briefly.

II.3.1.1 Definition :

Monolithic applications will be deployed as a huge WAR file that wraps all the code from the different teams and all the external dependencies of the project, such as frameworks and libraries, as illustrated in the Figure II.6 . To deliver this application to the production team, the project manager has to be sure that all the software teams deliver their code on time. If one of the teams has a delay for any reason, the application will be delayed for sure, because it cannot be delivered with unfinished parts. [50]

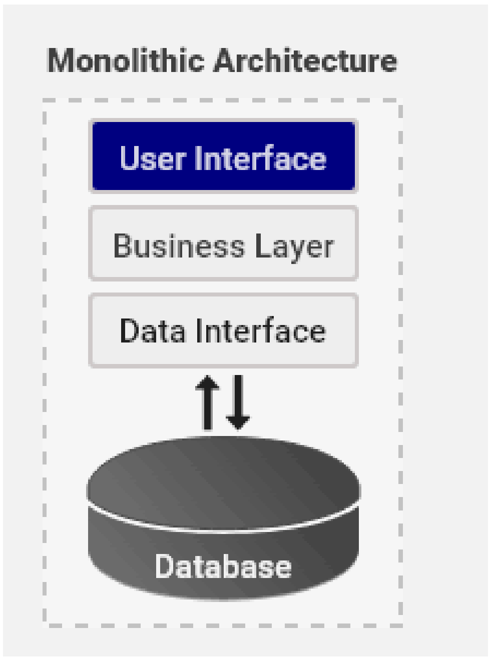


FIGURE II.6 – Monolithic flow on the example of application.

II.3.1.2 Advantages of the monolithic application :

There are many significant advantages to having all of your code in the same repository. Perhaps the most compelling one is the ability to tightly couple interdependent changes. That is, the ability to atomically update both an interface and all the users of that interface.

According to [65] here is some of the advantages of monolithic application :

— Simplicity of development.

— Simplicity of debugging.

— Simplicity of deployment.

— Simplicity of application evolution.

— Low cost in the early stages of the application.

II.3.1.3 Limitation of the monolithic application :

Our old monolith applications were good a decade ago. Now, they are obsolete and not useful anymore with the current demand. This architecture has the following limitations or drawbacks according (Justas Kazanavičius and Dalius Mažeika)[47] to :

— Monolith became too big and complex to maintain or extend.

— Modularity and decentralization are an important aspect.

— Less Scalable.

— More deployment and restart times.

II.3.2 "SOA"(Service Oriented Architecture) :

SOA is an architectural technique that involves the interaction between loosely coupled services that function independently, whereas a service can be combined to provide the functionality of a large software application.[82]

II.3.2.1 Definition :

There is no authoritative definition for "Service Oriented Architecture (SOA)" yet. Different resources explain SOA in different ways. One typical definition can be found in [56] Service-oriented architecture (SOA) is a type of software design that makes software components reusable using service interfaces that use a common communication language over a network.
A service is a self-contained unit of software functionality, or set of functionalities, designed to complete a specific task such as retrieving specified information or executing an operation. It contains the code and data integrations necessary to carry out a complete, discrete business function and can be accessed remotely and interacted with or updated independently. [2]

II.3.2.2 How it works :

By exposing services using standard network protocols (like SOAP, "JSON", "ActiveMQ" or Apache Thrift) to send requests or access data, SOA prevents developers from having to perform integration from scratch. Instead, they can use patterns called enterprise service buses ("ESBs"), which perform the integration between a centralized component and backend systems and then make them available as service interfaces. This also allows developers to reuse existing functions instead of recreating them.
In a service-oriented architectural style, services communicate using a system of “loose coupling.” This is a way of interconnecting components (also called “elements”) in a system or network so that they can pass information or coordinate a business process while reducing the dependencies between them(See Figure II.7). This, in effect, creates a new app. [2]

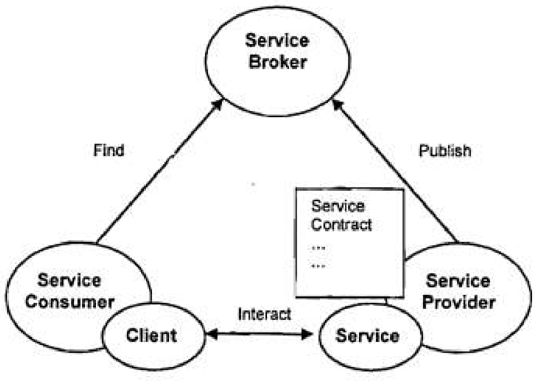


FIGURE II.7 – Basic SOA model.[56]

II.3.2.3 Advantages of SOA :

In this section, we will present some of SOA architecture's advantages compared to the old architecture (monolithic) according to [2] :

— Faster time to market and greater flexibility.

— Convenience of availability.

— Use legacy infrastructure in new markets.

— Greater reliability.

II.3.2.4 Challenges of SOA :

There is no doubt that Service Oriented Architecture has gained acceptance as a way to exchange data previously trapped in legacy systems and isolated databases. According to a 2006 study by "AMR" Research, 35% of executives said their companies had implemented one or more projects using SOA.[38]
According to [59] here are some of the challenges that faced the SOA architecture :

— Real-time communication.

— Testing services.

— Reliability, assurance of stable service.

— Organizational change (including SOA value).

— Making services work together (cooperation).

II.3.3 Microservices :

Microservices are the latest trend in software service design, development, and delivery.1 they constitute an approach to software and systems architecture that builds on the well-established concept of modularization but emphasizes technical boundaries.
Each module each microservice is implemented and operated as a small yet independent system, offering access to its internal logic and data through a well-defined network interface. this increases software agility because each microservice becomes an independent unit of development, deployment, operations, versioning, and scaling. [45]

II.3.3.1 Definition :

There is no official definition of microservices, but there are some definitions that are wholesome about it according to AWS. [1]
Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs. These services are owned by small, self-contained teams.

Microservices architectures make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features.

II.3.3.2 Microservice architecture :

Microservices architecture is an alternative way to overcome the challenges of monolithic architecture by decomposing the monolith into a set of small services and making them communicate with each other through a lightweight mechanism like REST API or message bus. This approach allows to build and maintain applications in a simpler way when a solution is broken down into smaller pieces that work seamlessly together. [47]
Splitting application into distinct independent microservices allows for managing them by individual teams within a software development organization and working them independently. Usually, teams developing microservices are organized around business capabilities, but not technical capabilities. Each new requirement should be addressed by only one microservice to retain independent development.[60]
Figure II.8 below shows the microservice architecture :

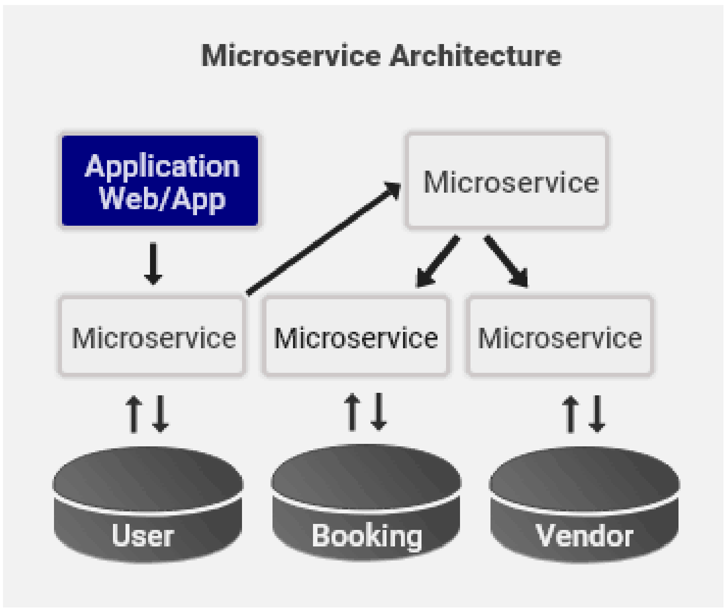


FIGURE II.8 – Microservice flow on the example of application.

II.3.3.3 Characteristics of Microservices :

Here we will mention some of the reasons why Microservices got widely used in the last years :
•**Scalability :** With the instances of small-sized microservices being easy to develop, implement and operate, the scalability of large complex systems can be increased manifolds.
•**Continuous delivery :** Implementation of new microservices and re-deployment of existing microservices with an upgraded functionality is a lot easy.[64]

•**Development :** Microservices implementing single tasks/logic involve less effort in the development and deployment of individual microservices.[64]
•**Independence :** Microservices are considered to be developed and implemented in a way that they should work independently hence removing various dependency issues like error cascading through other operations.[69]
•**Maintainability :** Due to the small size, it is prone to fewer bugs and is easy to maintain and upgrade. [64]

II.3.4 Software architecture comparison :

As was established in the previous sections, we have presented a few of the software architectures, so now we will show some studies that compare some of their functionalities. In the next Table II.2, we will show the comparison of SOA vs Monolithic vs Microservices according to a study was made by [55] in terms of componentization, component size, elasticity, deployment, storage mechanisms, technology and scalability.

	Monolithic Architecture	Service-Oriented Architecture	Microservices Architecture
Componentization	Module	Service	Microservice
Component size	Big	Coarse-grain	Fine-grained
Elasticity	A single point of failure	Coarse-grain	No single point of failure
Deployment	Holistic creation and deployment.	Each component deployed independently	Each service is built and deployed independently.
Storage mechanism	Shared database	Shared database	Private database
Technology	The same programming language and framework	Isomorphism	Heterogeneous
Scalability	Unable to scale on demand.	Scale on demand.	Scale on demand

TABLE II.2 – Comparison of different architectures.

Since that monolithic architecture is considered legacy architecture we will bypass discussing it there is no doubt that no one recommends this architecture we are going to focus on the microservices and SOA architectures as reported by (Vinay Raj and Ravichandra Sadam). [73] They considered SOA and microservices architectures for comparison to find the appropriate one to use in the design of enterprise applications.
They have used a standard web-based application, vehicle management system (VMS) (Bhallamudi et al., 2009) which is used to select, customize, and purchase vehicles and its parts using a web interface. The goal of this application is to help customers to select, customize, compare vehicles, locate dealers, and request a quote. All the details of the vehicles, their parts, and prices are configured in the database and help customers with details using the user interface. Customers

can select the vehicle of choice and the dealer for the selected vehicle from the inventory data. Customers can even select the part and the product type of the vehicle from the interface.
The selected information is generated as a lead and sent to the dealer who helps the customer in purchasing the vehicle and its parts.
The next Table II.3 shows the services and components of each architecture that were used :

Notation in SG SOA	SOA services	Microservices	Notation in SG MSA
81	Config service	Config service	ms1
82	Part service	Part service	ms2
83	Product servic	Product service	ms3
84	Compare service	Compare service	ms4
85	Incentives and pricing service	Incentives service	ms5
		Pricing service	ms6
86	Dealer and inventory service	Dealer service	ms7
		Dealer locator service	ms8
		Inventory service	ms9
87	Lead service	Get-a-quote service	ms10
		Lead processor service	ms11
88	User interface client	User interface client	ms12

TABLE II.3 – Details of services of both SOA and microservices-based applications.

This work helps in understanding the differences in terms of complexity and performance of both styles. Based on the analysis done for the complexity of the applications, it is clear that microservices application has more NoSs and application is more complex than SOA. The coupling between the services is also less in a microservices architecture. However, the performance testing shows better results for microservices with quick response times with 500 and 1,000 users.
Furthermore, the chosen case study application exhibits better results when chosen the BRs with the same NoSs in both styles. After this experimental study, we conclude that microservices architecture exhibits better performance results with the use of cloud-based environments and can be used in the design of large enterprise applications. In this work, they compared both the architectures in terms of coupling, complexity, and performance. As part of future work, other features such as scalability, maintenance, etc. will be considered for comparison.

II.4 API's (Application Programming Interface) :

The software has become indispensable in today's business environment. It has become challenging to envisage business success without software. Given this, there have been advances in the field of software development on how software is developed and tested. Software development over the years is perceived as a daunting task and demands a lot of activities. Hence, developers are continually exploring innovations that will aid the software development process. Application Programming Interface (hereafter referred to as APIs) is one such innovation in the software development domain. APIsform an integral component of the software ecosystem. These software ecosystems have become an ideal way of constructing large software solutions on top of a common technology platform.

II.4.1 Definition :

As stated by (Joshua Ofoeda et al) [66] Various authors have explained what APIs are, from diverse perspectives. While some authors have given a concrete definition for APIs, others prefer to give clues by providing the attributes or characteristics of.
We will use the definition provided by [66] This common ground enables different software to exchange information. Through the exchange of information, services between and within organizations, they can create value.

II.4.2 Types of API's :

Broadly classifying, APIs can be divided into two types : public APIs and private APIs. Going by name, public APIs are open to all for use. Private APIs, on the other hand, are accessible only by a restricted group. Private APIs may be for B2B partner integrations or for internal use. Those used for partner integration are also known as partner APIs. Those for internal use are referred to as internal APIs. An internal API can ease and streamline internal application integrations. It can also be used by internal developers for building mobile apps for an organization's own use. [30]
Figure II.9 below shows the different types of API's :

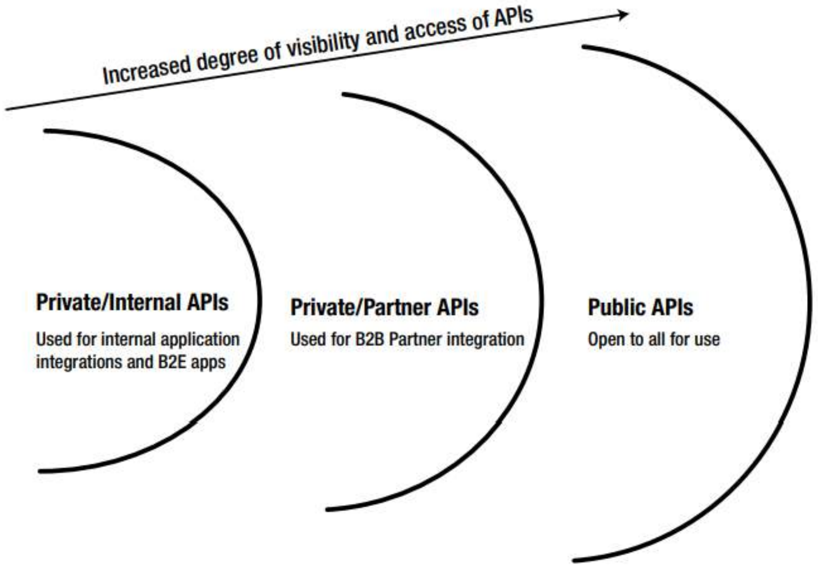

Increased degree of visibility and access of APIs
Private/Internal APIs
Used for internal application integrations and B2E apps
Private/Partner APIs
Used for B2B Partner integration
Public APIs
Open to all for use

FIGURE II.9 – Types of APIs [30]

Here is a quick description of each type provided by RedHat [1] :

1. https ://www.redhat.com/en/topics/api/ what-are-application-programming-interfaces

•**Private :** The API is only for use internally. This gives companies the most control over their API.
•**Partner :** The API is shared with specific business partners. This can provide additional revenue streams without compromising quality.
•**Public :** The API is available to everyone. This allows third parties to develop apps that interact with your API and can be a source for innovation.

II.4.3 Types of API architectures :

We can also understand APIs in terms of their architecture. An API's architecture consists of the rules that guide what information an API can share with clients and how it shares the data. REST, SOAP, and "GraphQL" are the most popular API architectures in use today — let's unpack each one in more detail.

II.4.3.1 SOAP(Simple Object Access Protocol) :

SOAP or Simple Object Access Protocol is a lightweight "XML"-based protocol for exchanging information in a decentralized distributed environment. It supports a Remote Procedure Call style and also message-oriented data exchange. SOAP has derived from the XML-"RPC" standard, and all SOAP messages are encoded with XML. A SOAP message is typically transported via HTTP. This makes it firewall-friendly and suitable for usage over the internet.
• SOAP architechture :

SOAP-based web services are web services that are located on remote machines to allow access from anywhere. SOAP typically follows some standards for sending and receiving data. SOAP-based web services are platform-independent, operating system-independent, and development platform-independent. depicts this architecture. This means one can access any type of web service from anywhere on any platform. For example, to access the web service that has been developed using Java on MAC from the .NET platform on Windows (See Figure II.10). SOAP commonly uses HTTP, but other protocols such as "SMTP" may also be used. SOAP (Simple Object Access Protocol) is a way for a program running in one kind of operating system (such as Windows 2000) to communicate with a program in the same or another kind of an operating system (such as Linux) by using the World Wide Web's Hypertext Transfer Protocol (HTTP) and its Extensible Markup Language (XML) as the mechanisms for information exchange. [49]
SOAP-based architecture typically revolves around XML-encoded messages transmitted over HTTP. First of all, the "WSDL" file has to be written which contains a Server-side data model in the form of XML schema types.
Once all this has been set, now WSDL to be made public so that every portal client can reach the particular portal server, this can be done by using a SOAP-enabled HTTP web server. Moreover,

server-side code to be written in a way that it will handle incoming service requests and formulate appropriate responses.[49]

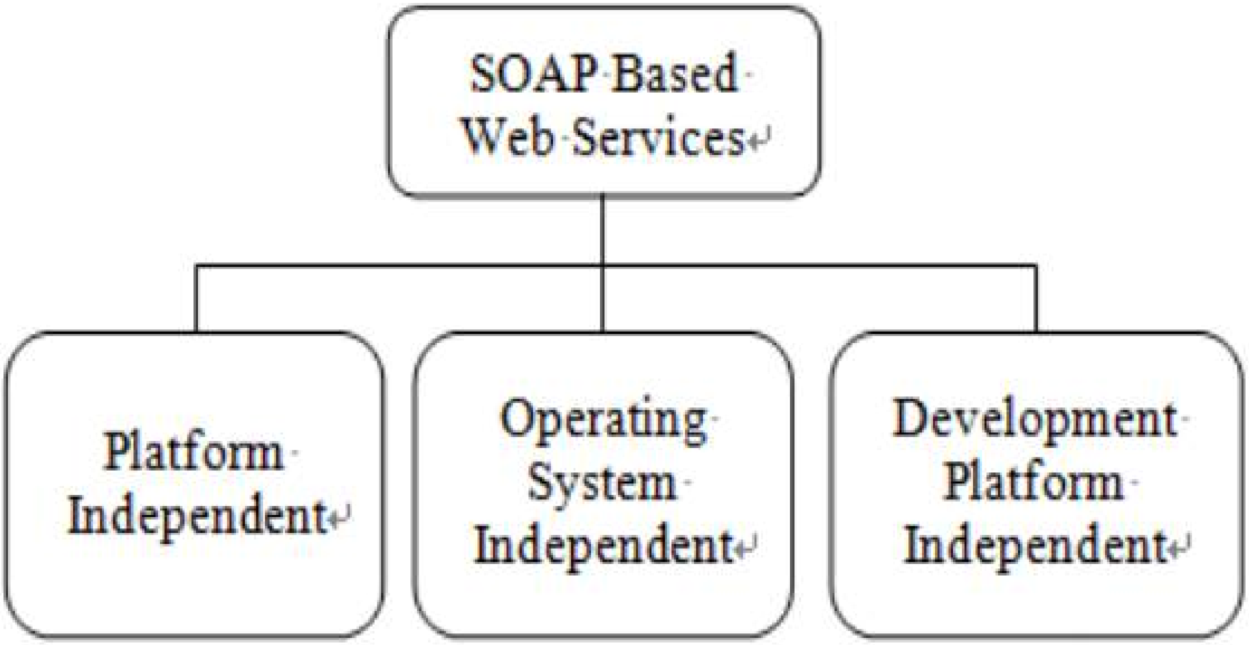

SOAP Based Web Services
Platform Independent
Operating System Independent
Development Platform Independent

FIGURE II.10 – SOAP architecture. [49]

• **WSDL :**

WSDL [1] is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME.

• **SOAP message structure :**

A SOAP message is the basic unit of communication between peer SOAP nodes.[40]
SOAP envelope The outermost syntactic construct or structure of a SOAP message is defined by SOAP within which all other syntactic elements of the message are enclosed. SOAP block A syntactic construct or structure is used to delimit data that logically constitutes a single computational unit as seen by a SOAP node.
A SOAP block is identified by the fully qualified name of the outer element for the block, which consists of the namespace URI and the local name. A block encapsulated within the SOAP header is called a header block and a block encapsulated within a SOAP body is called a body

1. https ://www.w3.org/TR/2001/NOTE-wsdl-20010315

block.
SOAP header A collection of zero or more SOAP blocks that may be targeted at any SOAP receiver within the SOAP message path.
SOAP body A collection of zero, or more SOAP blocks targeted at the ultimate SOAP receiver within the SOAP message path.
SOAP fault A special SOAP block that contains fault information generated by a SOAP node.
The following diagram illustrates how a SOAP message is composed (See Figure II.11).

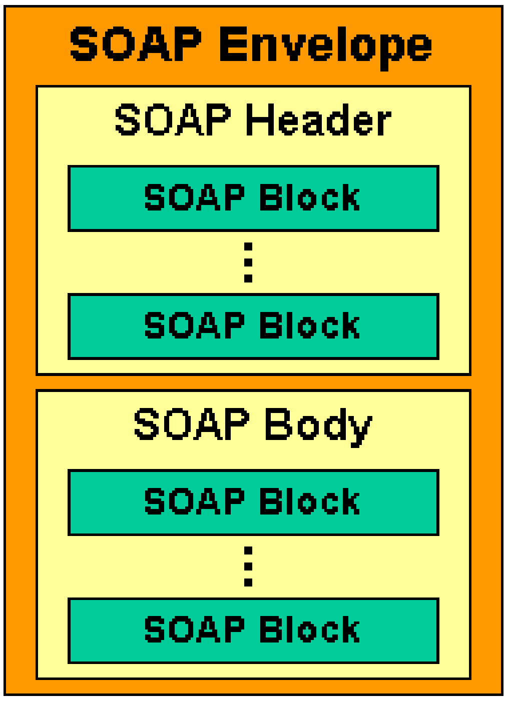


FIGURE II.11 – SOAP message structure. [40]

SOAP advantages :

Here we will mention some of the SOAP advantages that helped it get its popularity [11] :
•**WS security :** SOAP defines its security known as WS Security.
•**Language and platform independent :** SOAP web services can be written in any programming language and executed on any platform.
Disadvantages of SOAP Web Services :

And here are some disadvantages.[11] :
•**Slow :** SOAP uses XML format that must be parsed to be read. It defines many standards that must be followed while developing the SOAP applications. So it is slow and consumes more bandwidth and resource.
•**SWSDL dependent :** SOAP uses WSDL and doesn't have any other mechanism to discover the service.

II.4.3.2 REST API's :

In the last few years, REST has gained increasing attention from the Web, the mobile, and the service communities.

•Definition :

REST, which stands for the Representation State Transfer, is an architecture style that was firstly introduced by Fielding [36] in his PhD thesis. The REST architecture aims to make the Web into a scalable yet reliable network-based hypermedia system. The World Wide Web (WWW) has evolved with the REST architecture and has become the largest REST system ; this rapid and large-scale WWW development has shown the potential of what REST can bring into a system, such as evolvability, simplicity, and performance.

•Charachterstics of REST :

REST is a set of architectural constraints, not a protocol or a standard. API developers can implement REST in a variety of ways. For an API to be considered RESTful, it has to conform to these criteria[14] :

- A client-server architecture is made up of clients, servers, and resources, with requests managed through HTTP.
- Stateless client-server communication, meaning no client information is stored between getting requests and each request is separate and unconnected.
- Cacheable data that streamlines client-server interactions.
- A uniform interface between components so that information is transferred in a standard form. This requires that :

  •Resources requested are identifiable and separate from the representations sent to the client.

  •Resources can be manipulated by the client via the representation they receive because the representation contains enough information to do so.

  •Self-descriptive messages returned to the client have enough information to describe how the client should process it.

  •Hypertext/hypermedia is available, meaning that after accessing a resource the client should be able to use hyperlinks to find all other currently available actions they can take.
- A layered system that organizes each type of server (those responsible for security, load-balancing, etc.) involved the retrieval of requested information into hierarchies, invisible to the client.
- Code-on-demand (optional) : the ability to send executable code from the server to the client when requested, extending client functionality.

Advanteges of REST :

RESTful Web Services have the following features and advantages : [62]
•**Addressability :** Resources are marked with "URIs" (a global identifier). They are accessible once they are exposed on the Web rather than require a separate resource discovery and location mechanism such as "UDDI".
•**Links and connectedness :** Resources are linked with each other by using hyperlinks which point to valid future states in representations. What's most important is that we can implement state transfer by following the links.
•**Statelessness :** Statelessness : Each request includes all the necessary information for the servers to understand it, so each transaction is independent and unrelated to previous ones. Servers don't need to keep states between requests.
•**Uniform interface :** Resources are manipulated using a fix set of HTTP methods (GET, PUT, DELETE and POST) without special code to deal with each one. Additionally, these methods (except POST) are Idempotent.

II.4.3.3 GraphQL :

Over a few years back, REST APIs were considered standard web APIs, which now have a strong competitor. REST APIs provide some excellent features like stateless servers and structured access to resources. However, over time, it doesn't offer flexibility with the access of data and client changing requirements. In 2015 GraphQL was introduced by Facebook, which overcomes the problems with the REST and provides more flexibility and efficiency to the client requirements. For example, remove the over and under fetching.[43]
•**Definition :**
GraphQL [1] is a query language for APIs and a runtime for fulfilling those queries with your existing data. GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.
•**Architectural structure of GraphQL :**
GraphQL query language is increasingly adopted and easy to use, based on a conceptual framework and introducing a new type of data interface for the web. [18] There are three different architecture of GraphQL, i.e. simple, gateway and hybrid5 architecture. Each of these is elaborated on below.[43]

1. **Simple architectural structure :**
   Client-server architecture is when the Server is connected to a database and the client makes a request and receives a response quickly by the Server (See Figure II.12). Different protocols such as "TCP", WebSockets and other options can create a connection between Server and client. GraphQL servers can work with different types of databases, be it a "NoSQL" or an "SQL" or an relational database.

1. https ://graphql.org/

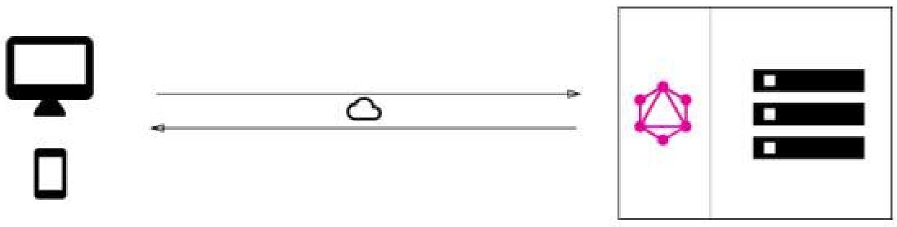


FIGURE II.12 – Simple architect of GraphQL.[43]

2. **Gateway architecture of GraphQL :**

   The gateway architecture is an approach to developing web applications that hides the complexity behind a simple GraphQL API. The basic idea behind this architecture was to develop a query language that can interact with multiple services at the same time in order to deliver a single endpoint to its query.
   This feature of gateway architecture (Figure II.13) makes this query language convenient and less complex ; this also makes GraphQL flexible to use. Another advantage of this architecture is its ability to integrate newer technology without changing its old infrastructure.

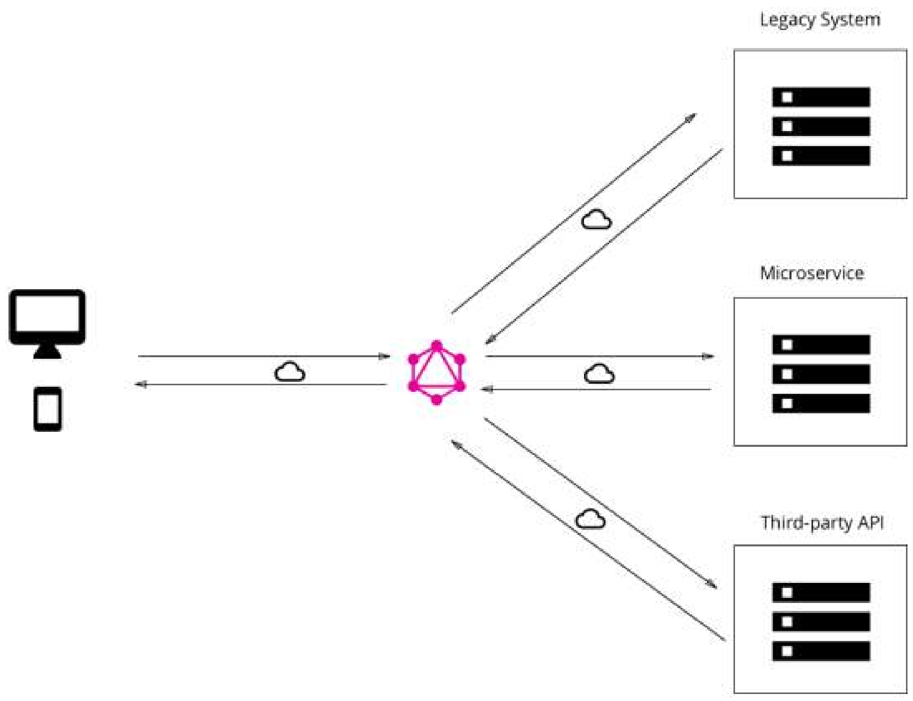


FIGURE II.13 – Gateway GraphQL architect.[43]

3. **Hybrid GraphQL architecture :**

   This is a combination of simple and gateway GraphQL architecture. In this model, we can combine the two approaches and build a server system that can be connected to the

database and communicate with the legacy system or third-party APIs. This model can receive the query and then resolve it by fetching data from the database or communicating with the integrated system.
This flow is shown in Figure II.14 :

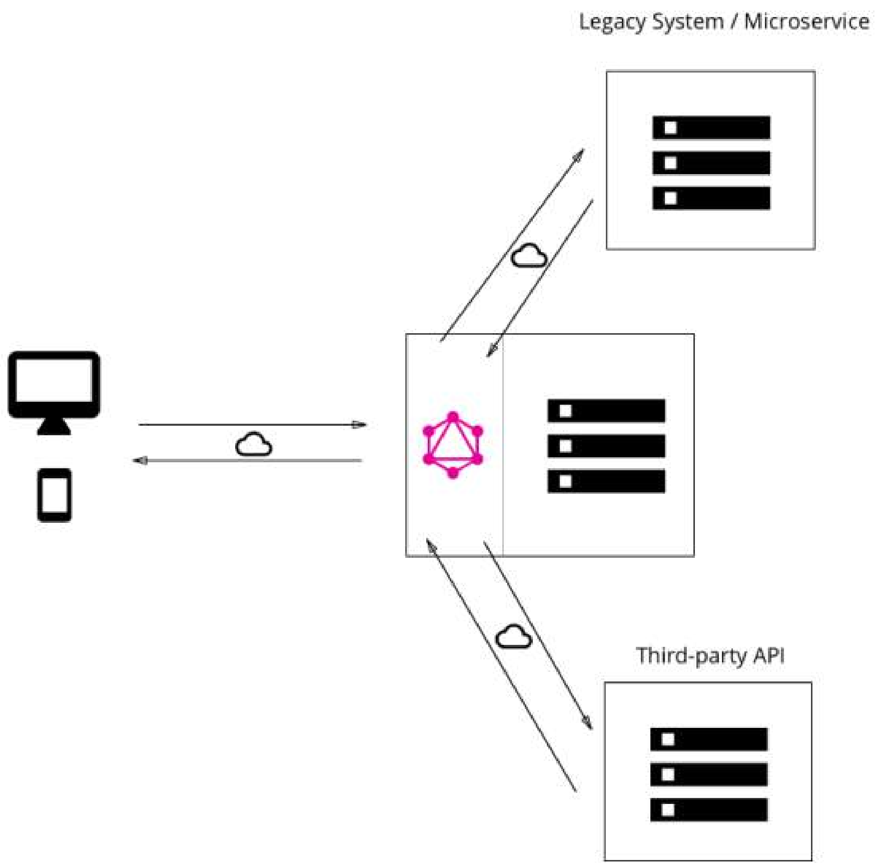


FIGURE II.14 – Hybrid GraphQL architect.[43]

Advantages of Graphql :

GraphQL provides a lot of advantages we will list a few of them according to RedHat : [16]
•A GraphQL schema sets a single source of truth in a GraphQL application. It offers an organization a way to federate its entire API.
•GraphQL calls are handled in a single round trip. Clients get what they request with no overfetching.
•Strongly defined data types reduce miscommunication between the client and the server.
•GraphQL is introspective. A client can request a list of data types available. This is ideal for auto-generating documentation.
•GraphQL does not dictate a specific application architecture. It can be introduced on top of an existing REST API and can work with existing API management tools.

Disadvantages of Graphql :

Despite all the advantages provided by GraphQL, it has some disadvantages like any other technology :[16]
•GraphQL presents a learning curve for developers familiar with REST APIs.
•GraphQL shifts much of the work of a data query to the server side, which adds complexity for server developers.
•GraphQL shifts much of the work of a data query to the server side, which adds complexity for server developers.
•Caching is more complex than with REST.
•API maintainers have the additional task of writing maintainable GraphQL schema.

II.4.3.4 Comparative studie :

In this section, we will showcomparative study between SOAP, REST, and GraphQL. As was stated by Sayago Heredia, J., Flores-García, E., Solano[76], who made a Comparative Analysis Between Standards Oriented to Web Services : SOAP, REST, and graphql.
The methods used were a systematic mapping to define the metrics to use for comparisons such as response time and performance. A test environment was implemented, starting with the development of a web application using each of the technologies to be evaluated in different programming languages. Then, the performance of web services was tested with the defined metrics and tools.
As a result, GraphQL has less response time and higher performance than SOAP and REST based web services.

II.5 Communication in cloud :

All this talking about software and API architectures is going to lead us to pose the question of how and where these applications communicate ? to answer this question we wrote this section which will explain where to deploy the application and how they communicate and the types of communications in the cloud.

II.5.1 Definition :

Communication is a two way process where both sender and receiver take turns to send and receive a message. This is depicted in Schramm's Model of Communication which is shown in the next Figure II.15 . The model shows that the communicator is the source of information. The communicator translates his idea into words, symbols, pictures, graphics voice, tone, facial expressions and body language through a common medium.
The receiver then picks up the message and decodes it. The decoding process is not that simple

because the receiver interprets the message using personal experience, expectation, comprehension level and other factors that influence his perception or understanding [5]. After decoding, the important thing now is the one that completes the process of communication which is known as the feedback. This is signifies that the message of the sender has been received. The nature of the response usually shows the quality of understanding. Basically, we can say that the elements of communication include the source, medium, receiver and feedback.[53]

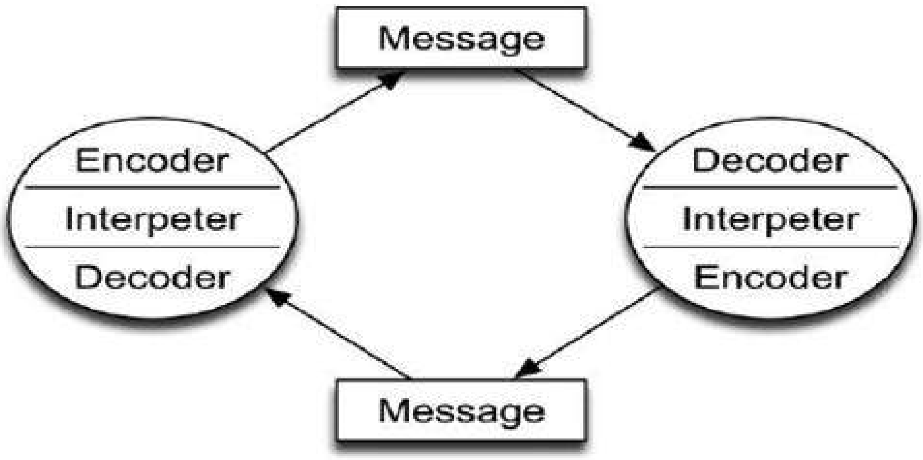


FIGURE II.15 – Schramm's Model of communication. [53]

II.5.2 Communication in cloud :

Cloud computing presents opportunities for scaling applications to virtual servers effectively by allowing enterprises to dynamically adjust their computing resources. Many companies usually deploy their applications on Infrastructure as a Service (IaaS), Platform as a Service (PaaS) environments, or use Software as a Service (SaaS) applications. One of the main motivations to use cloud solutions is to scale applications on-demand easily. However, this becomes cumbersome in the case of monolithic applications that consist of large modules.
That led to the creation of the previous software architectures which means that every application became a set of a lot of mini-applications that communicate between their endpoints to act like one wholsome application.[18]
In the next section will display the different types of communication in the cloud to clarify the differences between them.

II.5.3 Synchronous communication :

There is no official definition of this communication so we introduced the definition provided by Adrian Tarnowski. [12]
In the synchronous flow, the connection remains open from the moment the request is submitted to the server by the client (See Figure II.16). The client waits until the server sends back a

response.
A real-life equivalent to this situation is when you go to a website and must wait for it to load and return the content to you. A more casual example is calling your friend on a cell phone. You know immediately whether he is available for a conversation (picks up), or not (doesn't pick up). When you talk, you get an immediate response to your questions.

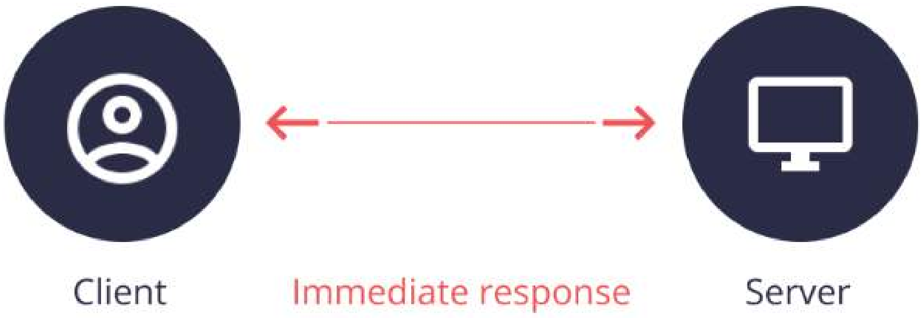


FIGURE II.16 – Synchronous communication. [12]

II.5.4 Asynchronous communication :

In case of the asynchronous flow, the client doesn't wait for the response to the request, only for the confirmation that the request was received by the server (See Figure II.17). For example, when you place an order in an online store and get an e-mail confirmation – it doesn't have to be sent immediately, because you don't wait "here and now" to receive it.
Another example of asynchronous communication are text messages. The only information you get immediately is whether the message was delivered. You wait for the response, which may take several minutes or a couple of hours.[12]

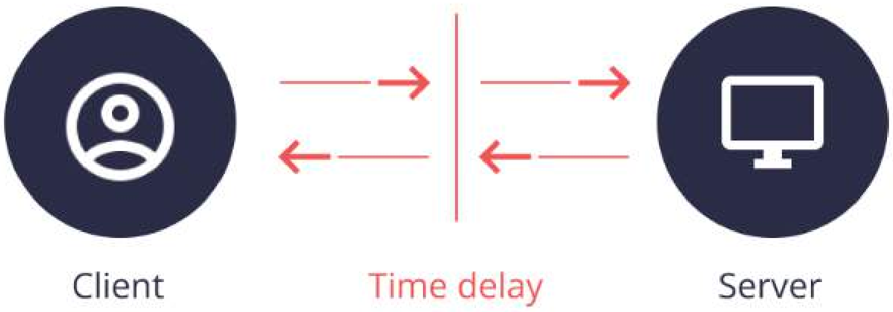


FIGURE II.17 – Asynchronous communication. [12]

II.6 Messaging softwares :

As was stated previously synchronous communication is direct and simple (relatively) however, Asynchronous communication requires messaging software for it to be accomplished it's called broker. so before we start talking about these message software we will explain the some of the concept that they cover.

II.6.1 Message brokers :

A Message broker is an architectural pattern that can receive messages from multiple destinations, determine the correct destination, and route the message along the correct route, as stated in the book Enterprise Integration Patterns by Hohpe and Woolf.Message brokers enable systems to deal with messaging and routing by mediating communication among components. Once applications implement a message broker pattern, it decreases the coupling between application components.[23]
Message Brokers are centralized, in the architectural sense, to control and manage all messages. Therefore, all of the incoming and outgoing messages are sent through Message Brokers, which analyze and deliver the messages to their correct destination. This procedural step can be understood in the following Figure II.18 :

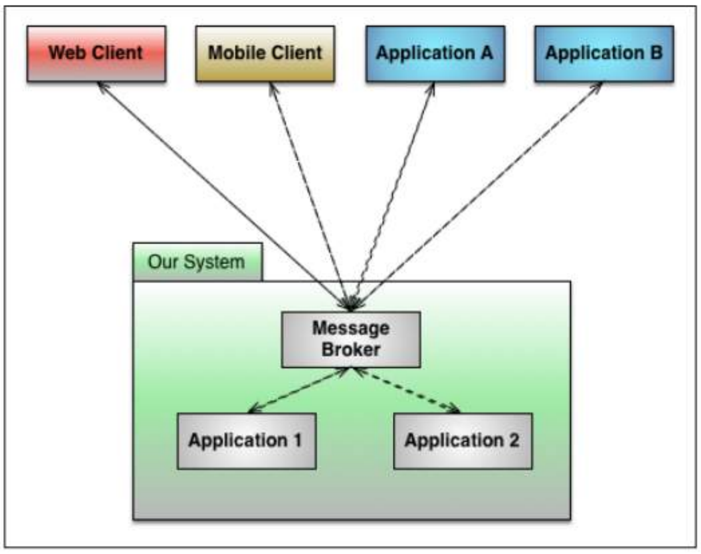


FIGURE II.18 – Message broker.[23]

II.6.2 Message queues :

A Message Queue is, briefly, a queue for messaging. Queue is the basic data structure behind the functioning of a Message Queue. Message Queue operations are similar to Queue data structure operations, such as the enqueue and dequeu operations. An enqueue operation leads

to adding an element to the back of the queue. A dequeue operation leads to the deletion of an element from the front of the queue.[23]
Message Queues provide concurrent and asynchronous operations to scale applications. In a message queue, messages wait up until a message is retrieved by an application.Let's take it see the chart as shown in the following Figure II.19 :

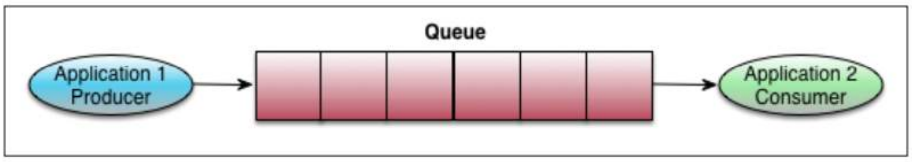


FIGURE II.19 – Message queue.[23]

All this development and technology led to the emergence of "AMQP" technology (Advanced Message Queue Protocol) which we will address below.

II.6.3 AMQP (advanced message queue protocol) :

John O'Hara from J. P. Morgan started AMQP in 2003. He put incredible amount of work into it. Then, J. P. Morgan approached other firms to establish an organization for creating open standards in messaging. According to AMQP's [1] official website AMQP is an open standard for passing messages between applications or organizations. So, AMQP just defines the messaging properties, queue properties, how messages are routed between applications and clients, how Message Brokers ensure that the message is received or sent, and other concerns such as reliability and security. According to the AMQP [2] website, AMQP has lots of capabilities to accomplish goals.

II.6.4 "RabbitMQ" :

RabbitMQ is an open source Message Broker software that tries to solve messaging problems by implementing the AMQP. RabbitMQ is licensed with Mozilla Public License. RabbitMQ became part of GoPivotal in May 2013 and the community has helped in the development of RabbitMQ. Since then, the community has been trying to improve RabbitMQ. [23]
In summary, RabbitMQ has the following functionalities to solve messaging problems :

— Ensures that messages are sent and received.
— Routes the messages to the correct destinations.
— Saves the state of the messages.
— Supports multiple transportation protocols (AMQP, "MQTT", "STOMP", HTTP).

1. https ://www.amqp.org/
2. https ://www.amqp.org/

— Supports clustering.

— Highly scalable, reliable and available.

II.6.4.1 RabbitMQ architecture :

In the above section, we got you covered what is RabbitMQ. Now we are discussing how RabbitMQ works ?.
Each mirrored queue consists of one master and one or more mirrors where each Broker contains all the data in a Queue. The master is hosted on one node commonly referred as the master node. Each Queue has its own master node. All operations for a given queue are first applied on the Queue's master node and then propagated to mirrors. Once failure occurs, consumers can be forwarded to digest data from the mirror queue in other Brokers by Push mode or Pull mode. Exchange can be used as a forwarding agent, which helps implement route conversion and forward messages to the corresponding queue. After Exchange is bound to a queue, messages are distributed to the message queue according to different binding rules according to the type of Exchange, either one message is distributed to multiple message queues or one message is distributed to a message queue.[37]
Figure II.20 below clearly shows the architecture of RabbitMQ :

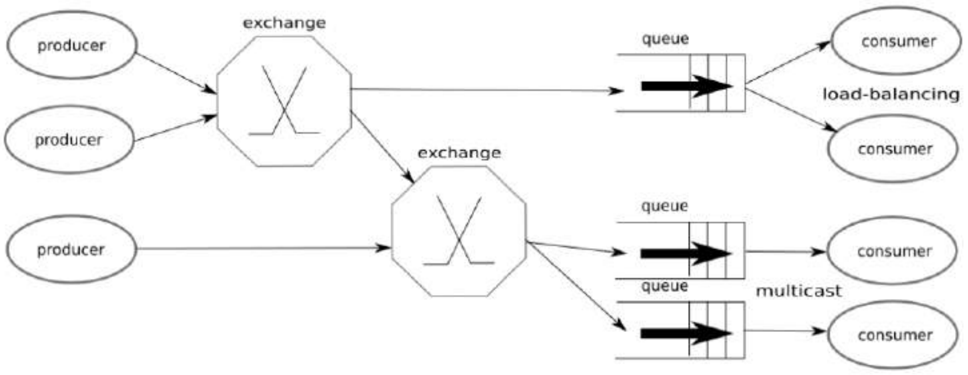


FIGURE II.20 – RabbitMQ (AMQP) architecture.[32]

This is the full RabbitMQ architechture some part can be excluded when using simple methods as stated in the RabbitMQ website. [8]

II.6.4.2 Advantages of RabbitMQ :

RabbitMQ has many advantages we are going to demonstrate some of them in this section :[13]
•**Reliability :** RabbitMQ offers a variety of features to let you trade off performance with reliability, including persistence, delivery acknowledgements, publisher confirms, and high availability.[13]

•**Flexible routing :**Messages are routed through exchanges before arriving at queues. RabbitMQ features several built-in exchange types for typical routing logic. For more complex routing you can bind exchanges together or even write your own exchange type as a plugin.[13]
•**Clustering :**Several RabbitMQ servers on a local network can be clustered together, forming a single logical broker.[13]
•**Highly available queues :**RabbitMQ supports replicated queues and streams. machines in a cluster, ensuring that even in the event of hardware failure, messages are safe as long as there is a majority of cluster nodes online.[13]
•**Large community :**There's a large community around RabbitMQ, producing all sorts of clients, plugins, guides, etc. Join our mailing list to get involved !.[13]

II.6.4.3 Disadvantages of RabbitMQ :

With all the advantages of Rabbitmq, it's still has some drawbacks that can't cover
•Issues with processing big amounts of data. [37]
•Needs Erlang.

II.6.5 "KafkaMQ" :

Kafka was developed at LinkedIn and primarily used for log processing. This worked well for Kafka's user engagement metrics collection use case. The fundamental features behind Kafka are performance over reliability and it offers high throughput, and low latency message queuing. Kafka can be used either for online or offline log processing. The fact that the reliability of message reception is traded off to some extent implies the loss of a single record among a multitude is not a huge deal-breaker. The rationale behind this is, that for log aggregated data, delivery guarantees are unnecessary. [46]

II.6.5.1 Kafka architecture :

Apache Kafka is a platform for real time environment using distributed public-subscribed messaging system and it can handle a large volume of data which enables you to send messages at end-point. Apache Kafka is developed at LinkedIn and available as an open source project with Apache Software Foundation.[41]
First we will explain how Distributed stream computing is now an unavoidable processing paradigm. It allows to get results continuously and in real-time from huge volumes of fresh data. Over the last years, the amount of data streams has not stopped increasing in many fields such as financial applications, network monitoring, sensor networks, and web log mining. Indeed, the Internet of Things accelerated the development of stream computing through the fast expansion of sensors deployed at geographically distributed locations. Large Internet companies also shifted their workloads towards the streaming model : for example, Facebook has to handle 106 events/s within a latency between 10 and 30s for advertisement purposes, which represents a data rate of

9GB/s at peak. The typical stream processing pipeline is illustrated in the next Figure II.21 and consists of 4 phases : [51]

— **Data collection :** The first step is to collect the data in real-time, from a set of sensors for example.

— **Ingestion :** The purpose of this step is to distribute the data stream on several machines in order to process it in parallel. Relevant state-of-the-art systems : Apache Kafka.

— **Processing :** This phase consists in processing the stream of data, typically through a topology of distributed operators. Relevant systems : Apache Flink, Apache Spark, Apache Apex.

— **Storage :** The last step stores the results of the intermediate data persistently for fault tolerance, archival as well as a posteriori processing purposes. Relevant systems : "REDIS", Apache "HDFS", Cassandra.

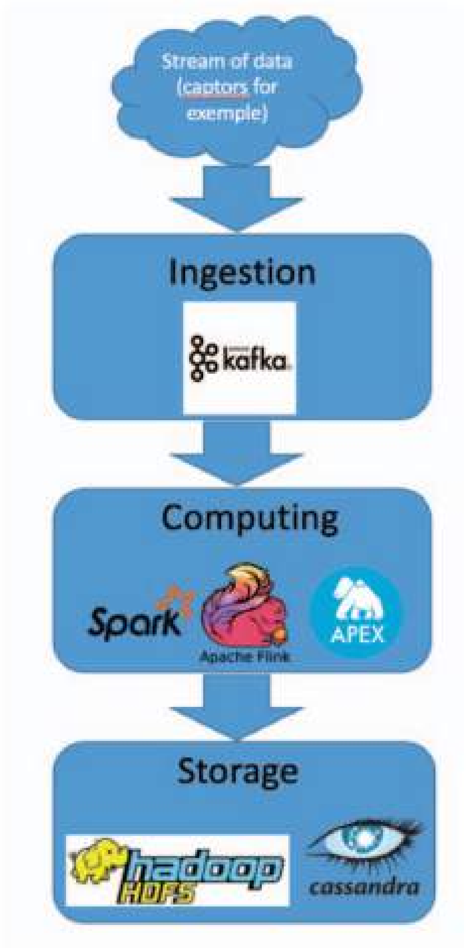


FIGURE II.21 – The typical stream processing pipeline. [51]

Kafka has a very simple storage layout. Each partition of a topic corresponds to a logical log. Physically, a log is implemented as a set of segment files of approximately the same size (e.g., 1GB). Every time a producer publishes a message to a partition, the broker simply appends the message to the last segment file. Compared to the traditional pub/sub systems, the notion of a consumer in Kafa is generalized to be a group of co-operating processes running as a cluster. Each message on the topic is delivered to one consumer in each of these consumer groups. As a result, the partition is the unit of parallelism of the topic and controls the maximum parallelism of the consumers.
Furthermore, because each partition has a sole consumer within its group, the consuming process can lazily record its position, rather than marking each message immediately, which is essential for performance. If the process crashes before the position is recorded it will just reprocess a small number of messages, giving at-least-once delivery semantics. [32]
Figure II.22 below clearly shows the architecture of kafka :

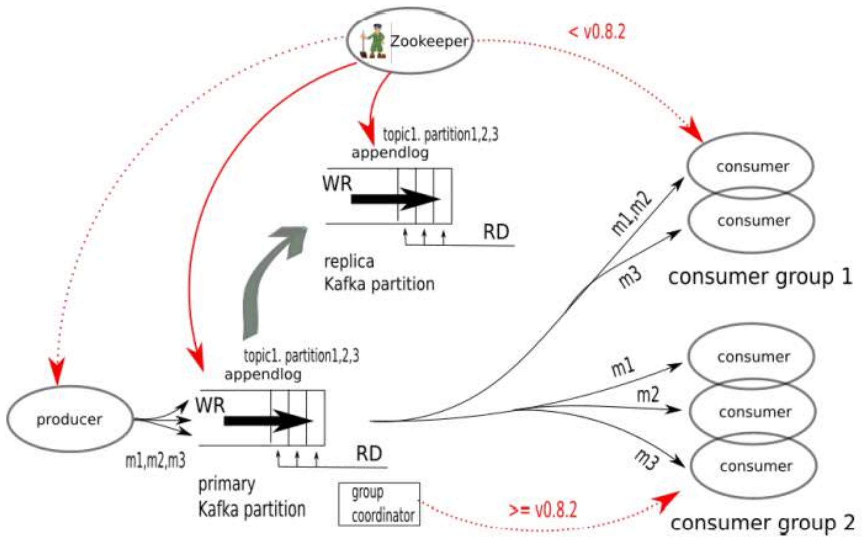


FIGURE II.22 – Kafka architecture. [32]

II.6.5.2 Kafka advantages :

Following advantages of Apache Kafka makes it worthy : [9]
•**Low latency :**Apache Kafka offers low latency value, i.e., upto 10 milliseconds. It is because it decouples the message which lets the consumer to consume that message anytime.
•**Real-time handling :** Apache Kafka is able to handle real-time data pipeline. Building a real-time data pipeline includes processors, analytics, storage, etc.
•**High throughput :**Due to low latency, Kafka is able to handle more number of messages of high volume and high velocity. Kafka can support thousands of messages in a second. Many companies such as Uber use Kafka to load a high volume of data.
•**Reduces the need for multiple integrations :** All the data that a producer writes go through

Kafka. Therefore, we just need to create one integration with Kafka, which automatically integrates us with each producing and consuming system.

II.6.5.3 Kafka disadvantages :

With The above advantages, there are following limitations/disadvantages of Apache Kafka : [9]
•**Do not have complete set of monitoring tools :** Apache Kafka does not contain a complete set of monitoring as well as managing tools. Thus, new startups or enterprises fear to work with Kafka.
•**Reduces performance :** Brokers and consumers reduce the performance of Kafka by compressing and decompressing the data flow. This not only affects its performance but also affects its throughput.

II.6.6 Comparaison between kafkaMQ and RabbitMQ :

In the present section, we will present a quick study applied to Kafka and RabbitMQ, in order to compare them.
This comparison was made by Amir Rabiee [72] to tests two different message brokers, Apache Kafka and RabbitMQ, to discuss and show the impact on throughput and latency when using a variety of parameters. The experiments conducted are focused on two primary metrics, latency, and throughput, with secondary metrics such as disk- and CPU usage. The parameters chosen for both RabbitMQ and Kafka are optimized for maximized throughput and decreased latency. The experiments conducted are tested on a cloud platform ; Amazon Web Services.
The results show that Kafka outshines RabbitMQ regarding throughput and latency. RabbitMQ is the most efficient in terms of the quantity of data being written, while on the other hand being more CPU-heavy than Kafka. Kafka performs better than RabbitMQ in terms of the number of messages being sent and having the shortest one-way latency.
On the other hand, the comparison made by (Sharvari T and al) [78] shows the usage of each one of them, in conclusion, said Kafka is more mature when it comes to distributed log systems and should be considered for real-time data analytics that requires high throughput and low latency, Kafka can be used in cases which are not real-time data critical, that is the streaming is unaffected even if few messages are missed/lost. Such applications are website user tracking, social media reactions, and internet advertisements . While RabbitMQ acts as a general message broker system and can be used for complex routing using exchanges and queues. This special routing is required in designing Internet of Things applications that connect many sensors. With the ACK transaction guarantee of RabbitMQ, all messages are guaranteed to be sent over to consumers, this feature is useful for financial transactions.
The following Table II.4 shows the Feature Comparison between Apache KafKa and RabbitMQ :

Features	**Apache Kafka**	**RabbitMQ**
Language developed in	Scala	Erlang
Started In	2011	2007
Messaging models supported	pub/sub Message queue	pub/sub Message queue
Brokered/Brokerless	Brokered	Brokered
Throughput	High	Medium to High
Latency	Low	Low to Medium
Protocols supported	Binary over TCP	AMQP, STOMP, MQTT
Message size	1 MB max	2 GiB
Message Delivery	At most once, Exactly once, At least once delivery	At most once, At least once delivery
Languages supported	About 17 languages	About 30 languages
Message Ordering	Yes	Yes
Message Storage	Disk	In-memory/disk
Distributed Units	Topics	Queues
Used By	LinkedIn, Netflix, Facebook, Twitter, Chase Bank	Mozilla, AT T, Reddit

TABLE II.4 – Feature comparison table

II.7 Conclusion :

In this chapter, we discussed cloud services and API deployment in general , Then we defined related concepts such as Cloud virtualization , Software Architecture , API's , Communication in cloud and Messaging softwares.
In the next chapter, we will examine some relevant work in this area.

Chapter III

Cloud computing portability

III.1 Introduction :

In this chapter, we will get to the core of our report, which is data portability in multi-cloud. But before that, we will get into cloud API interoperability and its standard and solution that was proposed to fill this gap, then do a small comparative study between them because interoperability and portability are Two interrelated concepts.
After that, we will get back to data portability, what it is and its types, and why we need it, then the effort made to find solutions for this issue in the multi-cloud.

III.2 Interoperability :

At the moment, each cloud service provides a unique interface for users, apps, and clients to connect with the cloud. Interoperability is crucial for cloud computing on several levels.

III.2.1 Definition :

There are no official interoperability definitions in computer science in general and in Multicloud specifically but we will stick with the description provided by Wegner[85], who described interoperability as The capacity of two or more systems to interact in cooperation between additional software components despite variations in execution platform, language, and interface. It concerns the reuse of server resources by clients who are contacting the server in a scalable manner.

III.2.2 Types of interoperability :

There are many different forms of interoperability in the cloud, and each one focuses on a different aspect to make it interoperable. We'll list a few of them below according to [39] [84] :

— Application Interoperability.

— Management Interoperability.

— Data interoperability.

— Computing service interoperability.

— Manufacturing process interoperability.

— Publication and Acquisition Interoperability.

The main interoperability type that we focus on is data interoperability wich was defined by [86] [17] as data interoperability is the ability to exchange data between two systems in structural manners that both of them understand portability more specifically API which are the way that our data is offered and consumed.

III.2.3 Benefits of interoperability :

Intentionally or not, many cloud service providers utilize distinct architectures (REST, SOAP, GraphQL, RPC), making it difficult to establish API compatibility, which prevents such services from being able to connect with one another. Here are a few of the consequences of the lack of interoperability :

— Vendor lock-in :refers to a situation in which, once an organization has selected a cloud provider, either it cannot move to another provider or it can change providers but only at great cost.[22]

— Enable services to be scaled across multiple providers by guaranteeing the smooth transfer of data and applications from one service provider platform to another.[7]

— Through the usage of cloud computing, a business is able to leverage its IT investments more creatively and more quickly accept new ideas from the cloud provider.[7]

III.2.4 Exciting standard :

The lack of interoperability was caused by the divergence of cloud protocols and architectures, but this didn't stop major standards organizations from suggesting ways to improve interoperability. We will list a few of them in this section(See Table III.2.4) :

Standard	**Orgnization**	**Type of Interoperability**	**Description**
OpeOpen Virtualization Format [31]	Distributed Management Task Force	Management interface	Establishes a transport mechanism for moving virtual machines from one hosted platform to another
Open Cloud Computing Interface [87]	The Open Grid Forum	Application Program Interface	REST-based interfaces for management of cloud resources including computing, storage, and bandwidth.
Web Services Description Language 2.0 [4]	The World Wide Web Consortium	Web description	Unlike WSDL1.0 which describe only SOAP based web services WSDL2.0 provides description for both REST and SOAP based web services
OpenAPI itnitiative [10]	OpenAPI Organization	Application Program Interface	Created by a consortium of forward-looking industry experts who recognize the immense value of standardizing on how APIs are described
Cloud Computing Interoperability Forum [3]	Cloud industry forum	Application Program Interface	Sponsors of the Unified Cloud Interface Project to create an open and standardized cloud interface for the unification of various cloud APIs

TABLE III.1 – Interoperability standards.

Even with all of this, some providers didn't use the standards that were offered by international organizations and academies, which prompted some academics to develop solutions to avoid this interoperability lock-in ; this will be covered later.

III.2.5 Interoperability related works :

In this section, we will discuss a few relevant works and offered solutions to the problem of data interoperability (API interaction in our case), research will be briefly addressed, and a little comparison study will follow.

III.2.5.1 Solution of Bipin Upadhyaya et al [83] :

This solution focus on proposing a REST Access to SOAP-based web services. we can divide the work of this approach into two major sections the first section is the identifying of the method of the HTTP from the operation name in WSDL using a semantic approach that extract the name and identifies its HTTP verb(GET, PUT, POST, UPDATE) from the cluster that is used. the second section is the SOAP to REST converter this method does not affect the operation of traditional services. Instead, all the RESTful requests are converted to corresponding SOAP-based service operations and vice versa (See Figure III.1).

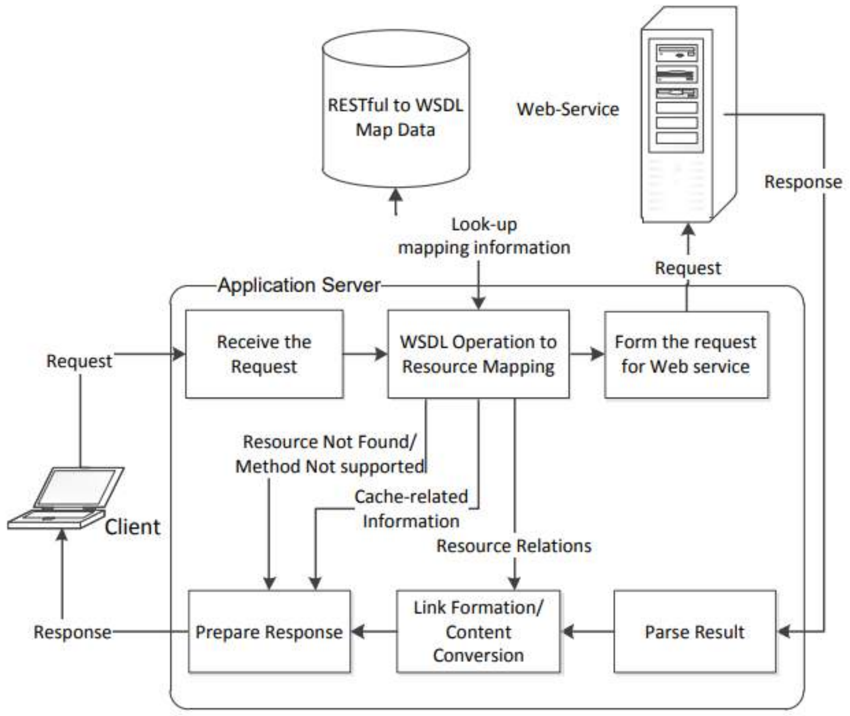

RESTful to WSDL Map Data
Web-Service
Response
Look-up mapping information
Request
Application Server
Request
Receive the Request
WSDL Operation to Resource Mapping
Form the request for Web service
Resource Not Found/ Method Not supported
Cache-related Information
Client
Resource Relations
Response
Prepare Response
Link Formation/ Content Conversion
Parse Result

FIGURE III.1 – REST/SOAP converter architecture . [83]

III.2.5.2 Solution of Bouzerzour and Slimani [25] :

Bouzerzour and Slimane presented a way to unify cloud web service descriptions or change them into a Generic Cloud Service Description (GCSD). They suggested a model as a service called "CLOUD SERVICE INTEROPERABILITY PIVOT MODEL" CIPiMo (See Figure III.2), which is based on MDE (Model-Driven Engineering). In this approach, they generated a WSDL and OWL-S meta-model and mapped it to GCSD using ATL model transformation language to change it into a GCSD meta-model and reinforce it with an example demonstrating the outcome of their effort.

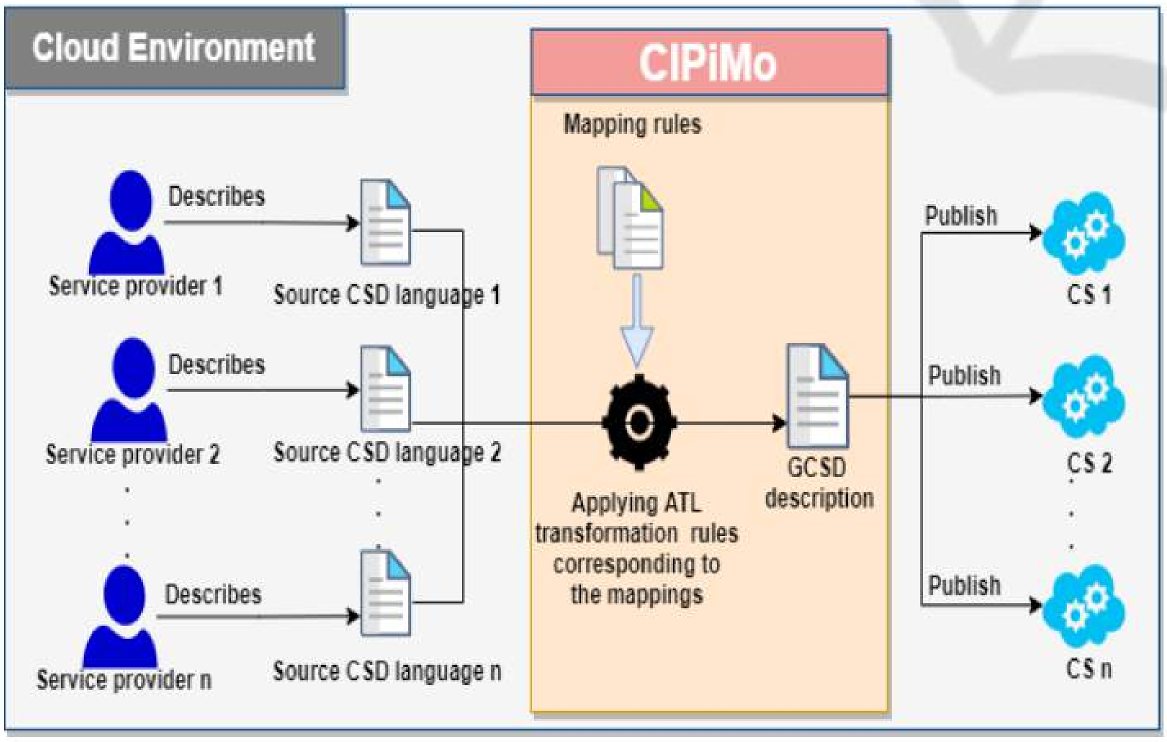


FIGURE III.2 – Interoperability model CIPiMo. [25]

III.2.5.3 Solution of Sean Kennedy et al [48] :

This solution proposed a protocol a new protocol called (SOAP to RESTful HTTP mapping) StoRHm (See Figure III.3).
The "STORHM" protocol allows an existing SOAP client to connect with a REST service. A configuration wizard is used to do the translation, which accepts as input a WSDL file, a WSDL schema, and, optionally, a "WADL" file. To choose parameters, the user must interact with a form interface. Furthermore, the transition from SOAP to REST is semi-automatic. The translation is from SOAP service to RESTful service in one sense ; in the converse scenario, this technique is ineffective.

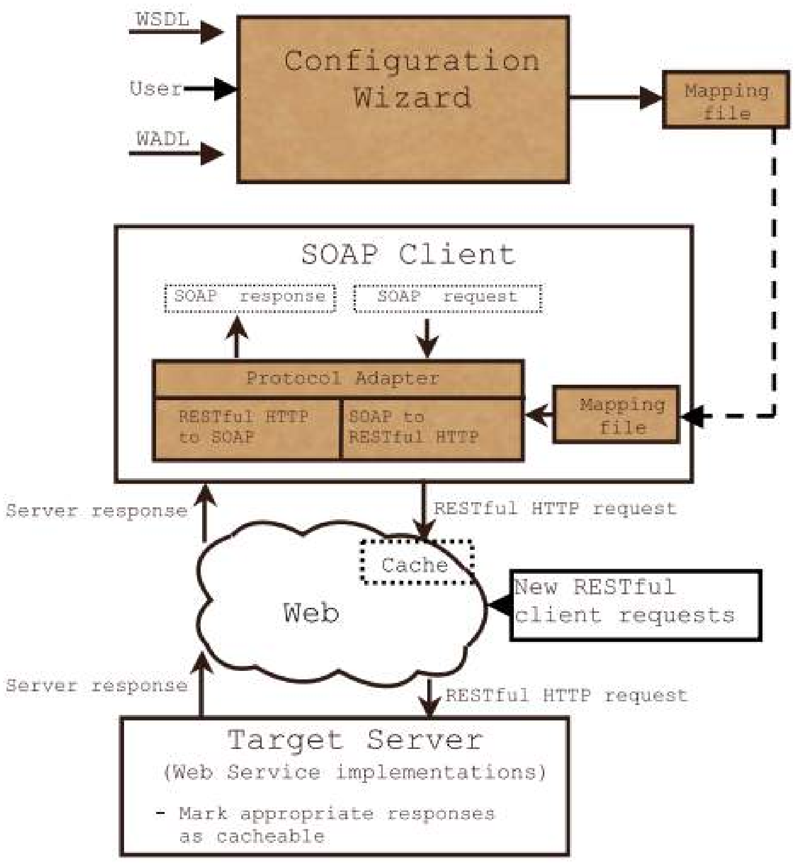


FIGURE III.3 – STORHM architect . [48]

III.2.5.4 Solution of Majda Elhozmari and Ahmed Ettalbi [57] :

This solution focuses on the use of the SaaS cloud to ensure interoperability and standardization in the heterogeneous cloud environment.
The solution to this proposal consists of adding an intermediary cloud service between Cloud provider and consumer in heterogeneous Cloud environments interfaces. In this case Cloud consumer and provider using REST interface can communicate easily with Cloud provider and consumer Using SOAP interface (See Figure IV.2). SOAP and REST Converter is a trainee component that ensures standardization and interoperability in cloud-based heterogeneous environments. This solution is essentially Cloud SaaS based on three principal components :
•**Interface with cloud consumer :** An interface is related directly to the consumer. The objective is to processing consumer request messages and response. This Interface allows the interaction with different services access standards REST and SOAP. It can deal and differentiate between SOAP and REST messages coming from Cloud consumers. This interface has an auto detect service access and can make a compatible environment of standards that help an easy communication between service access interfaces. In this case consumers can outsource their IT or applications without thinking about witch technology used by side of Cloud providers.
•**Interface with cloud provider :** There is an interface related directly to the Cloud provider, the

objective is to process provider response messages and request. This Interface allows the interaction with different services access standards REST and SOAP by the side of Cloud provider. It can deal and differentiate between SOAP and REST messages coming from Cloud consumers. The interface has an autodetect service access that can make a compatible environment of standards that help an easy communication between service access interfaces.
•**REST/SOAP converter :** SOAP and REST Converter is an intern component in our proposed solution that ensure standardization and interoperability in heterogeneous Cloud based environments. Using this component, Cloud consumer and provider using REST interface can communicate easily with Cloud provider and consumer Using SOAP interface, by converting SOAP to REST and vice versa.

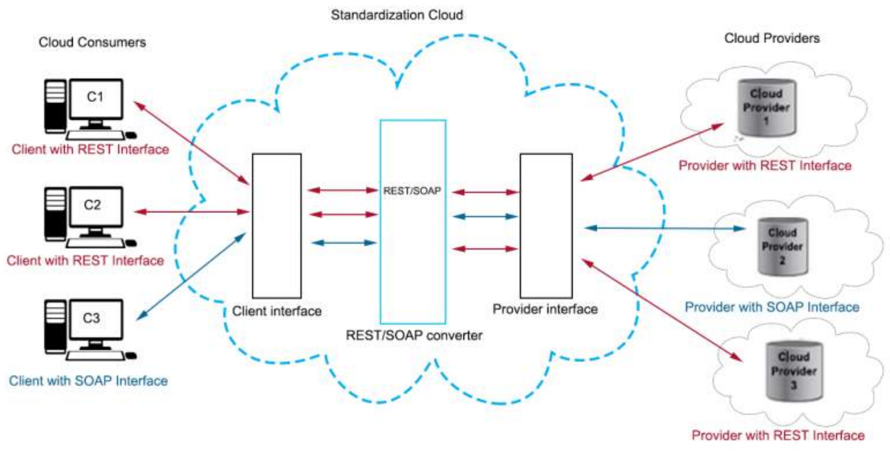


FIGURE III.4 – The proposed solution for cloud standardization. [57]

III.2.5.5 Solution of William Appleton [6] :

This solution present the DreamFactory software. DreamFactory is also a REST API middleware platform wraps a SOAP service into a REST API to make it simple to use and create a REST endpoints, instantly turn any SOAP into a live automatically. This conversion can make a request with JSON, calls the legacy SOAP service and then the SOAP response is converted back to JSON for the client application. The translation from SOAP to REST is automatic but it requires developer interaction to make this conversion available and understands what the mapping methods are talking about and how to use it in the application.

III.2.6 Interoperability solutions and standards table :

After we have discussed these solutions and standers, we will create a comparative table (See Table III.2) to demonstrate the differences and key areas of each of the solutions offered previously, based on the following criteria :

C1«Deployment mode» : This criterion will discuss how the solution is going or was deployed.
C2«Type of web description» : This criterion shows the description that describes the web service through the solution.
C3«Type of request» : Here we say how the solution receives the request.
C4«Level of the solution » : This criterion represents the level of the solution.
C5«Position of the solution» : This criterion represents the level of the solution.
C6«Type of the mapping» : How to mapping is done whether automatically or manually or semi-automatic.

Criteria / *Solution*	**C1**	C2	C3	C4	C5	C6
Open Virtualization Format [31]	Packaging	-	-	Management interface	-	-
Open Cloud Computing Interface [87]	Specification	-	-	API	Provider	-
Web Services Description Language 2.0 [4]	-	WSDL2.0	-	API	Provider / Client	-
OpenAPI itnitiative [10]	-	REST Description	-	API	Provider	-
Cloud Computing Interoperability Forum [3]	-	Unified description	-	API	Provider / Client	-
Solution of Bipin Upadhyaya et al [83]	Broker	WSDL	SOAP to REST	API	-	Semi automatic
Solution of Bouzerzour and Slimani [25]	MaaS	GCSD	-	API	Provider / client	-
Solution of Sean Kennedy et al [48]	Protocole	-	SOAP to REST∼	API	-	Semi automatic
Solution of William Appleton [6]	Software	-	REST To SOAP	API	Client	Automatic
Solution of Majda Elhozmari and Ahmed Ettalbi [57]	Broker	-	SOAP	API	Provider / Client	-

TABLE III.2 – Interoperability solutions comparison table.

III.2.7 Discussion :

As shown in the table above (see Table III.2) we can notice that there are some organizational and individual attempts to solve the problem of API heterogeneity, and how most of them ([48][25][83]) focus on turning REST services into SOAP services (give SOAP access to REST web services), and none of the solutions provide access for both web services SOAP access to REST services and REST access to SOAP-based services, where on the other hand most of the standards try on pushing cloud provider to use REST based web service [10] [87].
Only Dream Factory [6] attempted to provide REST Access to SOAP-based services, but without providing the description transformation that will describe the services.

III.3 Portability :

While interoperability refers to the ability to properly communicate across or among systems, portability refers to something quite different, which we shall discuss in this section.

III.3.1 Definition :

According to OMG (The Object Management Group) [24], portability refers to a customer's ability to move and adapt their applications and data between their own systems and cloud services, as well as between cloud services from different cloud service providers and potentially different cloud deployment models. The fundamental issue created by lack of portability is that it may take a significant amount of effort to transfer the program or data from its form on the source system to the form required by the target system.

III.3.2 Types of portability :

Portability is done by removing reliance on the underlying environment. A portable component can be simply transported and reused regardless of the provider, platform, operating system, location, storage, or other factors of the surrounding environment.
As noted in [24] [21] [5], portability is classified into two categories :
•Portability of applications :
Cloud application portability refers to the simplicity with which an application or component of an application may be moved from one cloud service to another that is similar or from a cloud service customer's computer to a cloud service. The simplicity of relocating the program or application components is crucial. Although the program might need to be recompiled or relinked for the target cloud service, substantial modifications to the application code shouldn't be required.

•Portability of data :
Cloud data portability is the capacity to transfer data from one cloud service to another cloud service or between a cloud service customer's system and a cloud service without the need to make a lot of effort , in a commonly used electronic format. It is the ease of moving the data that is the essence here. This might be accomplished by the source service providing the data in precisely the format that the target service accepts.
However, even if the formats are incompatible, it may still be easy and quick to convert between them using readily accessible tools.

III.3.3 benefits of portability :

Many different service providers use cloud computing in very diverse ways. The existing software stacks of cloud services are heterogeneous, and the functionality offered by various service providers is commonly incompatible. When it comes to requirements like promoting portability and avoiding vendor lock-in, this variety is a barrier.
The portability is requested for reasons varying from optimal selection regarding utilization, costs, or profits, to technology changes, as well as legal issues. As stated by [70] here is a few reasons that caused the need for cloud portability :
•Economical reasons : Here was mentioned two main reasons why it's needed from economical perspective :
1. Customer : Protection of the end user investments in developments.
2. Market : Development of a Cloud ecosystem and market.
•Technical reasons : When it comes to technical reasons there are a huge amount of reasons but we will go only with these two :
1. Concept : To exploit the advantage of elasticity and the pay-as-you concept.
2. Continuity : To ensure continuity in application and service functionality.
•Legal reasons : Finally, the legal reasons we mention :
1. Constraints : data, application, and service protections according to the locations or national laws.
2. Free will : Avoid dependence on only one external provider.

III.3.4 Portability related works :

In this section, we will mention a few of the related works that discussed API unification and data portability in cloud environments.

III.3.4.1 Solution of Diego Serrano and Eleni Stroulia [77] :

In this approach, Serrano and Stroulia proposed and implemented their solution which is Semantics-based API discovery, matching, and composition using Linked REST API (LRA). They proposed a formal methodology for describing the semantics of REST APIs based on core SPARQL graph patterns. The network pattern nodes relate to the API operations' input and output data types, and their edges reflect the semantic links between them. Based on this formal description model, they provide a completely automated technique for service creation and enactment based on iterative subgraph isomorphism. The LRA service composition method entails discovering API activities relevant to the input SPARQL query and traversing these APIs through a depth-limited search in their data production-consumption linkages, narrowing the search space with a set of heuristic criteria.
At runtime, the semantic API specifications mediate the lifting of the data produced by the invoked APIs to the shared semantic model and its translation to the syntactic formats required by the consumer APIs.

III.3.4.2 Solution of Cassia Trojahn et al [35] :

This study developed an API for Multi-lingual Ontology Matching, which, as the title suggests, is an ontology that matches two ontologies from various languages, in this instance English, French, and Portuguese.
In this technique, there are three forms of matching : direct, translation-based, and indirect.
Each of these categories has its own technique ; direct employs an external translator such as Google Translate API, whilst indirect and baseline use the method of Jung et al, which uses ontology alignment methods to match across ontologies in various languages. They conducted tests on these methods after applying them, and each strategy produced excellent results in each case independently.duced excellent results in each case independently.

III.3.4.3 Solution of Doina Caragea and Tanveer Syeda-Mahmood [26] :

Doina Caragea and Tanveer Syeda-Mahmood suggested a matching algorithm in this work to match the input/output of API when used in automated web service construction.
Their strategy combines lexical and semantic matching techniques. By finding a maximum matching in a bipartite network generated from the qualities, the technique ensures an ideal match of corresponding entities.
And this strategy was demonstrated by the text they gave, both semantically and lexically.

III.3.4.4 Solution of Giuseppina Cretella and Beniamino Di Martino [27] :

This paper discusses the problem of dynamic discovery and mapping of cloud services and their APIs. The architecture of the software that can make this problem go away is shown in Figure III.5 :

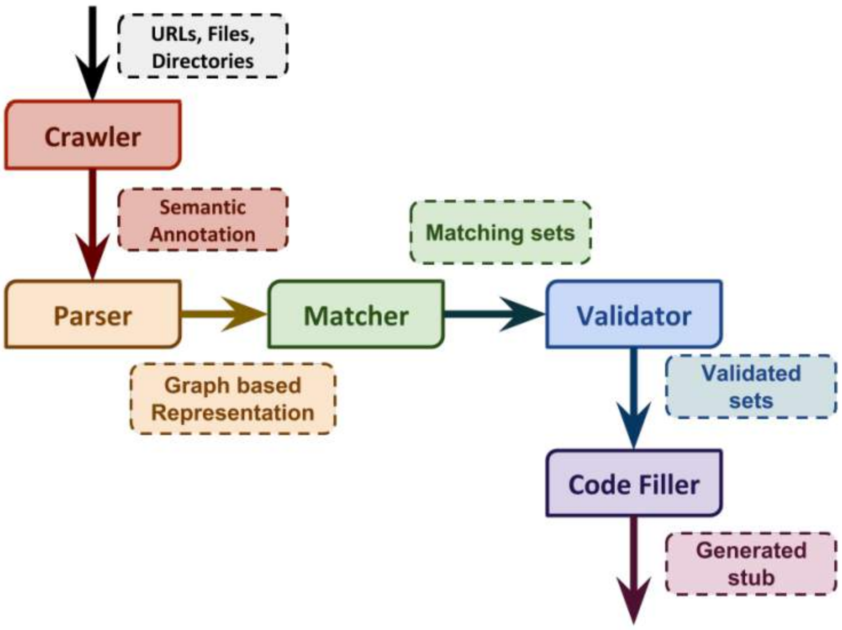


FIGURE III.5 – Architecture of Giuseppina Cretella and Beniamino Di Martin. [27]

As we can see in architecture there are five components which will be explained briefly After this :

1-Crawler : The crawler works by analyzing the files that represent the semantic annotations in the web resources it is trying to access, rather than analyzing just the data on the web pages it visits.

2-Parser : This component is responsible for reading the file of the semantic representation, The formats compliant with the parser are OWL, OWL-S, WSDL, SA-WSDL, and SA-REST. it takes this file and represents it in graph-based representation that contains syntactical and semantic information. this representation slightly differs from one description to another.

3-Matcher : The matcher can be described as the component performing the matching between the representations of a generic source with respect to a generic target one.

4-Validator : As the name implies, the user is needed to confirm the mapping performed by the previous phase following the matchmaking phase. The component that allows the user to validate the matched sets of couples is the Validator.

5-Code filler : Code Filler supports the mOSAIC Maintainer in the production of a new or changed Cloud API. It identifies the mapping between a single API call to the right connector, the driver, and the relative method of the driver. The generation of stubs is performed through the wsdl2java library.

If we can put in a small paragraph what has been done in this paper we can say that The Discovery and Mapping System is an API maintenance tool, supporting the mapping of Cloud vendor APIs to the mOSAIC API. It takes as input the semantic descriptors of the services expressed in a number of description languages. The matcher makes use of node-level matching and also of structural matching.

III.3.4.5 Solution of Yong Jo Lee and Jae Soo Kim [52] :

Yong Jo Lee and Jae Soo Kim offered unique ways of dealing with the Web API discovery and composition challenge by utilizing semantic technologies in this study. Without any human intervention, the suggested algorithms choose the different APIs involved in the composition. As well as a technique Choosing and integrating data mashup-appropriate APIs is critical for any mashup toolkit.
In this paper, we show how API features may be syntactically defined and semantically characterized, as well as how to use the syntactic and semantic descriptions to simplify Web API discovery and development. Finally, they demonstrate how API features may be syntactically and semantically specified, as well as how to utilize syntactic and semantic descriptions to facilitate Web API discovery and development.
The algorithm is based on the chart-based approach where They are gradually generated by backward sequencing from APIs.

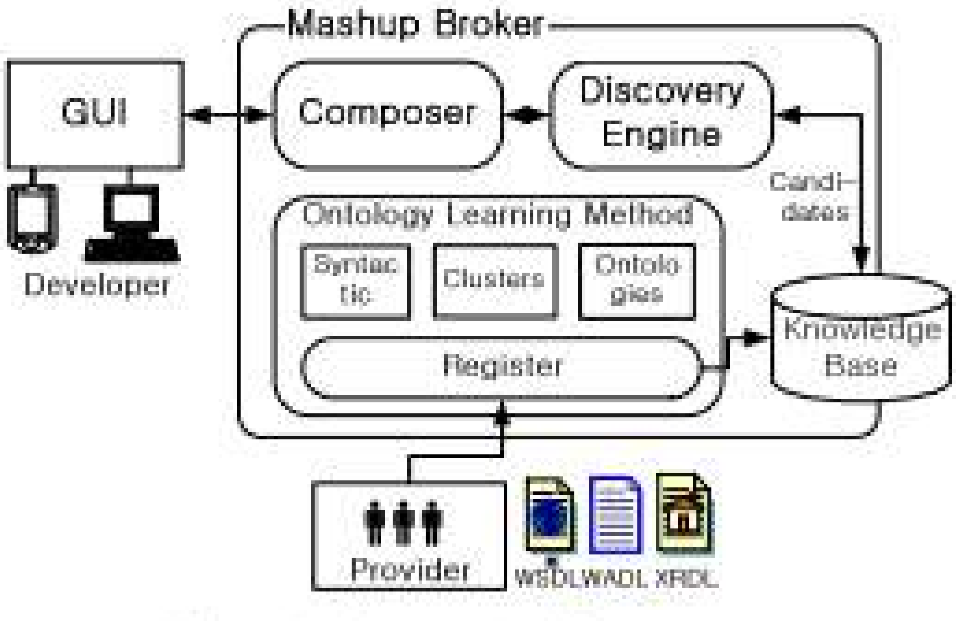


FIGURE III.6 – Architecture of Yong Jo Lee and Jae Soo Kim. [52]

•**The composer :** Is responsible for planning to achieve a composition relevant to the desired goal. It captures the current composition states and dynamically composes relevant APIs that can be added to the mashup.
•**The discovery engine :** Is used to select individual APIs that make up the composition. This engine is based on a semantic matching method that allows users to automate the discovery of Web APIs.
•**Ontology learning method :** Is an unsupervised technique to derive several semantically meaningful concepts from input/output parameters. The parameters are often combined as

a sequence of several terms. When clustering terms residing in the parameters into several meaningful semantic concepts.

III.3.4.6 Solution of Tanveer Syeda-Mahmood et al [81] :

This work describes a method for selecting web services that use both domain-independent and domain-specific ontologies. Based on the description of the web service, they derive domain-specific language from the description of the web using domain-independent language. After that, they utilize both name and type information to discover a match for it, and then they employ an efficient indexing method to allow scalability of search across huge service repositories, as well as an offline search, which virtually eliminates search time at the expense of storage. They also discussed why storage is so affordable when compared to other criteria. Finally, they conducted tests to demonstrate the effectiveness of their solution.

III.3.5 Portability solutions and standards table :

These were some of the ways presented to find a solution to the data portability problem that we addressed previously ; now we will conduct a comparison analysis of the solutions described (See Table III.3) based on the following criteria :
C1«Cloud-based solution» : If the solution is oriented toward cloud web services.
C2«Mapping aspect» : If the solution treats the parameter mapping.
C3«API model» : The type of API the mapping is applied on.
C4«Description model» : The type of description model used in the semantic solution.
C5«Language used» : The language in which the semantic is applied.
C6«Composition aspect» : Describe if the solution proposes a composition method.
C7«Language domain» : Describe whether the ontology use domain-specific or independent language.
C8«Discovery aspect» : It describes if the solution covers the service discovery aspect.
C9«Type of mapping» : Tells whether the mapping happens online or offline it means happens once and is stored to be used locally.

Criteria / *Solution*	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**
Solution of Diego Serrano and Eleni Stroulia [77]	Yes	Treated	REST	Proposed model	English	Treated	Domain-specific	Treated	Online
Solution of Cassia Trojahn et al [35]	Yes	Not treated	-	-	En-Fr Po	Not Treated	-	Not Treated	-
Solution of Doina Caragea and Tanveer Syeda-Mahmood [26]	Yes	Treated	-	Use API I/O	English	Treated	Domain-specific	Not Treated	-
Solution of Giuseppina Cretella and Beniamino Di Martino [27]	Yes	Treated	REST - SOAP	OWL, OWL-S WSDL SA-WSDL SA-REST	English	Not Treated	-	Treated	-
Solution of Yong Jo Lee and Jae Soo Kim [52]	Yes	-	-	WSDL WADL XRDL	English	Treated	-	Treated	-
Solution of Tanveer Syeda-Mahmood et al [81]	Yes	Treated	SOAP -REST	WSDL	English	Not Treated	Domain-specific and domain-independent	Not Treated	Offline

TABLE III.3 – Portability solutions comparison table.

III.3.6 Discussion :

This section will go through the previously described solution. so as we can see all the solutions are addressed to solve the problem of data heterogeneity on the web and are meant to be for cloud services but not directly, they are meant for web services and we concluded that we can use them in the cloud environment, and it can be noticed that the solutions [77] [35] [52] are aiming for the automatic compositions of web services and that will create the problem of API heterogeneity that's why they also aim to parameter mapping using domain domain-specific language ([77] [26]) while on the other hand, the solutions [35] [52] [81] did not address the mapping aspect but used multi-language API mapping [35] when solution [27] can read multiple description models that already exist.
All these solutions handle this problem from different aspects and even though non of the solutions cover the entire aspect still these solutions provided huge benefits to data portability in the application program interface.

III.3.7 Conclusion :

In this chapter, we discussed both interoperability and portability in the cloud environment, as well as their definitions and types, and why they are needed in the multi-cloud environment. We then discussed the solutions that address our problem mentioned at the beginning of this work, which are applied to both portability and interoperability, and we did a comparison with a brief analytic discussion about these solutions.

Chapter IV

Design of our solution

IV.1 Introduction :

This chapter will explain our solution and how it was created. This phase is critical since the reasons for everything we did while implementing this solution will be detailed in the following chapter. Our approach is separated into two key portions. The first is intended to tackle the interoperability problem, while the second is intended to solve the data portability problem.

IV.2 Needs analysis :

In this part, we will establish our system's requirements by introducing the actors who interact with it :

IV.2.1 The actors :

Identifying actors is one of the first steps in needs analysis. Each type of external entity with which the system must interact is represented by an actor. For example, the operating environment of a software system consists of the users, devices, and programs that the system interacts with. An actor in use case modeling specifies a role played by a user or any other system (mostly systems in our case)that interacts with the subject.
In the following table IV.1 we present the actors that will interact with our system in this approach :

Actor	**Description**
Service provider	This is the SOAP service owner that want to provide REST access to it
Client	Client that want to access SOAP web services through a REST endpoint

TABLE IV.1 – Actors and thier roles.

The following use case diagram Figure IV.2 shows the possible actions of each actor :

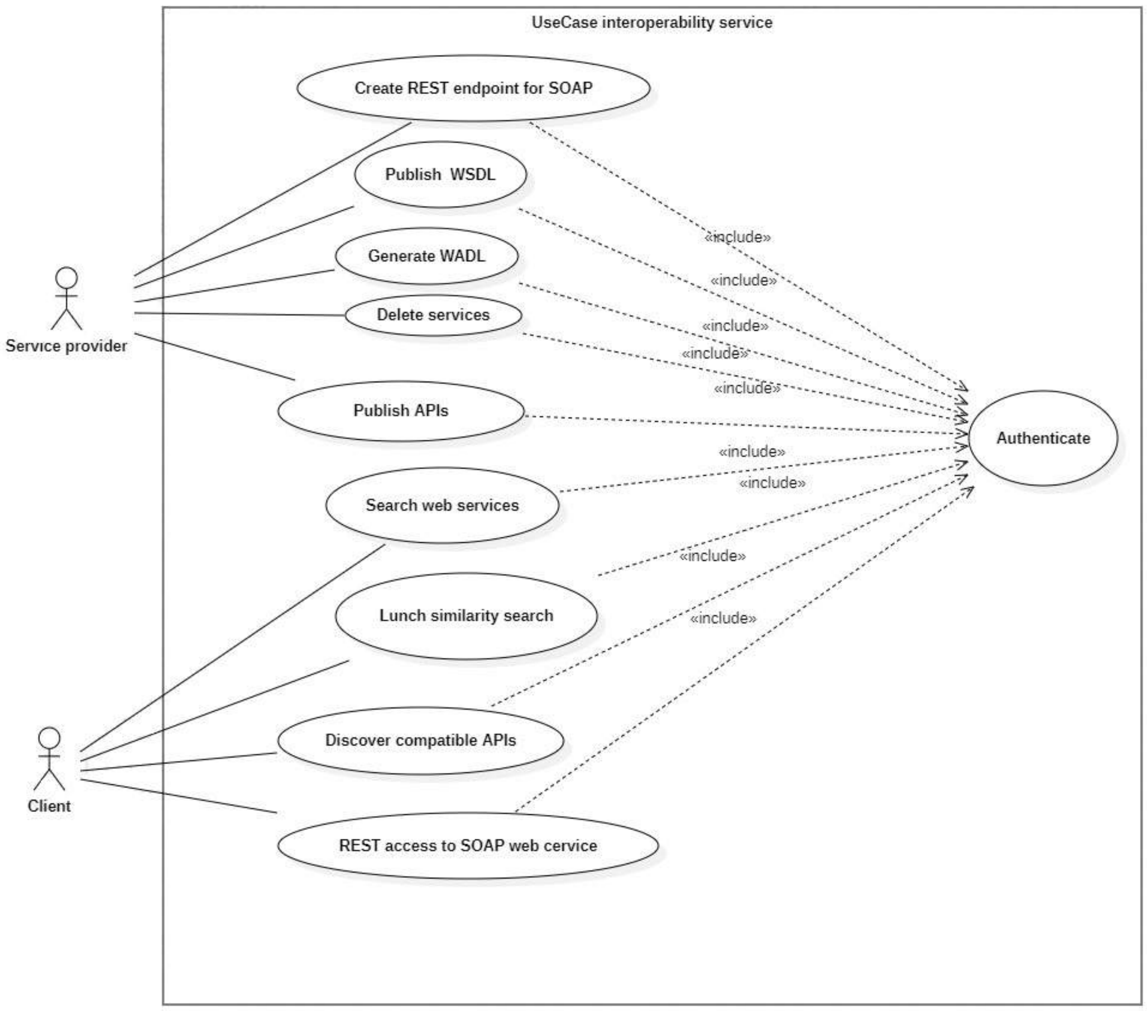


FIGURE IV.1 – Use case diagram.

While Table IV.2 provides a detailed description of each use case.

Use Case	**Description**
Authenticate	Service provider or client can't access any service without authentication and have the right only to access authorized services only.
Publish WSDL	SOAP service provider will publish his WSDl file for each of his services.
Publish APIs	Publish REST API inputs and outputs for discovery
Generate WADL	SOAP service provider can create WADL file from WSDL file.
Create REST endpoint	SOAP service provider can create REST endpoint to every SOAP endpoint.
Delete service	Provider can delete his own services.
Search services	Client can search from all available services.
REST access to SOAP services	Client can send REST message to one of the available services.
Lunch similarity search	Client can search semantic and syntactic match to their APIs
Discover compatible APIs	The client can choose any compatible API with his APi

TABLE IV.2 – Textual description of use case.

IV.3 Our proposed solution :

The solution to be implemented needs to be divided into two separate phases we will introduce each one alone we will show the combination between them .

IV.3.1 Interoperability solution :

After performing needs analyses, we started arriving at the shape of our solution that will solve the problems of API communication. In this section, we will show the architecture of our solution concerning the unification of API interoperability (See Figure IV.2) and explain how it works and its main purposes, we can divide them into 3 main purposes :

— **Web description unification.**

— **REST to SOAP conversion.**

— **SOAP to REST conversion.**

IV.3.1.1 General explanation of the solution :

And we will explain each one of these sections in detail, but before we shall give an overview of our solution. As was stated in [89] that 85 % of developers use the REST interface, whereas only 15 % choose the SOAP interface, that is what motivated us to propose REST access to SOAP microservices since it's the most used one and SOAP is legacy interface if we can say so, now let's explain our three main purposes of our solutions :

-Unification of web service description :
Following the previous decision, we have chosen REST access to SOAP microservices we think that we should provide the description that gets along with it which is WADL (Web Application Description Language), so we decided to convert the WSDL (Web Service Description Language) to WADL so that we have unified format of web descriptions.
This will help the discovery of APIs (this is not covered in our work), and the usage of APIs.

-REST to SOAP transformation :
Since our solution is to provide REST access to SOAP microservices, actually it's the first step of the access (we will see that in the next sections) going from JSON REST to XML SOAP is time expensive transformation regarding the work that has to be done and the information that must be added and the format changing ie : GET method information to the SOAP envelope.

-SOAP to REST transformation :
If we can say this is the final step for this system here we will transform information from the SOAP APIs response to JSON format that can be read by REST APIs, even though this step is not as expensive as the REST to SOAP transformation but still crucial for our system.
All these steps will be explained in the following section.

IV.3.1.2 Architechture of the solution :

The following Figur IV.2 shows the architecture of our solution along with the flow of requests, responses, and data flow inside it we will take a deep dive inside this system with an example and pseudo algorithms each one showing how the previous transformation works, we start by defining the names of each action :

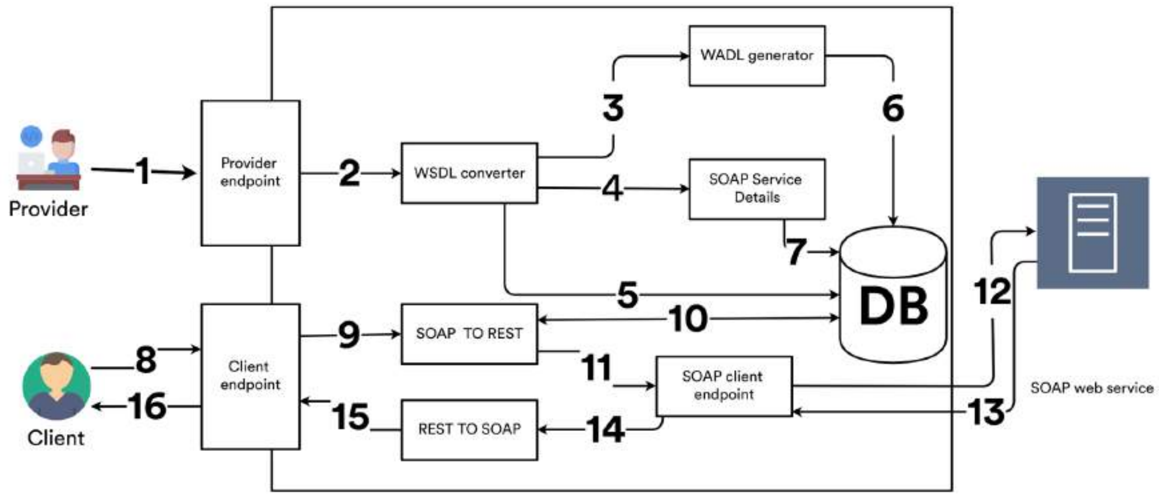


FIGURE IV.2 – Solution architecture.

As the Figure IV.2 we see the shows the flow inside our proposed solution, each number describe an action done by the system or the user (machine or human), sometimes a group of action is required to achieve a major step like the transformations that are going to be done. here is the description of each action :

1. The SOAP service provider sends the WSDL file as a text.
2. The WSDL string mapped into the WSDL-object.
3. WADL generator takes WSDL-object into WADL following mapping method will be provided in the TableIV.3 NOTE WSDL file can be mapped to multiple WADL file.
4. SOAP service details will be generated also from the WSDL for each endpoint.
5. Save the WSDL file in the database.
6. Save the WADL file to the database.
7. Save the SOAP service details object in the database.
8. Client selects the wanted SOAP service and sends a REST message.
9. SOAP to REST message will generate an XML version of the message.
10. Get SOAP service details from the database.
11. Form SOAP message and encapsulate it in soam envelope schema using the information provided by the SOAP service details.
12. The SOAP client will send SOAP requests to the specific SOAP web server.
13. The web server will respond with a SOAP response.
14. The SOAP client will receive the response and send it to REST TO SOAP.
15. REST to SOAP will convert the SOAP message to JSON and remove all the additional information.

16. REST response will be sent to the client.

After we mentioned these actions now let us explain them using an example .Note : some of these actions are going to act in a group to achieve one step of our solution.

IV.3.1.3 Unification of web service description :

Like the title says we will start by the unification of the service description but before we start we will explain it a little bit. service description languages are used to describe the APIs, and in our case, we want to transform from WSDL to WADL as previously mentioned so we have to propose a model for both description and the mapping rules to go from WSDL to WADL (See Figures (IV.3,IV.4)).

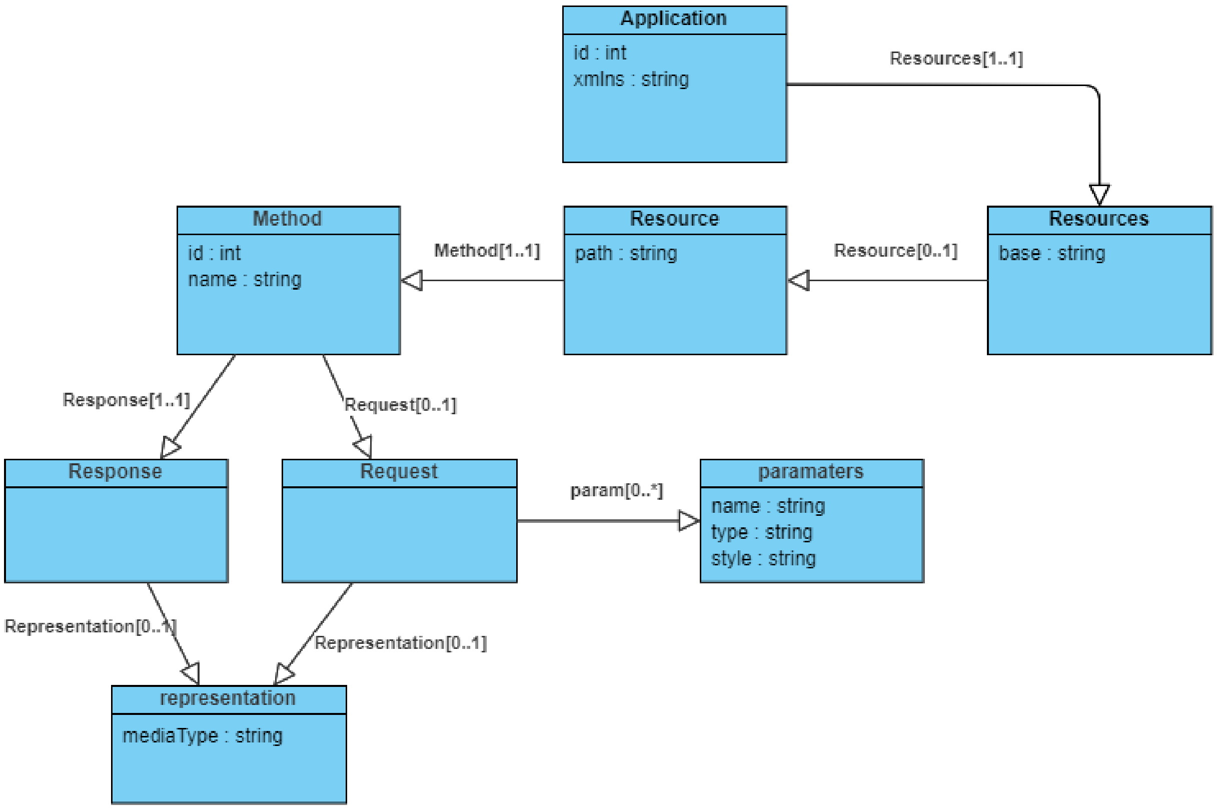


FIGURE IV.3 – WADL class diagram.

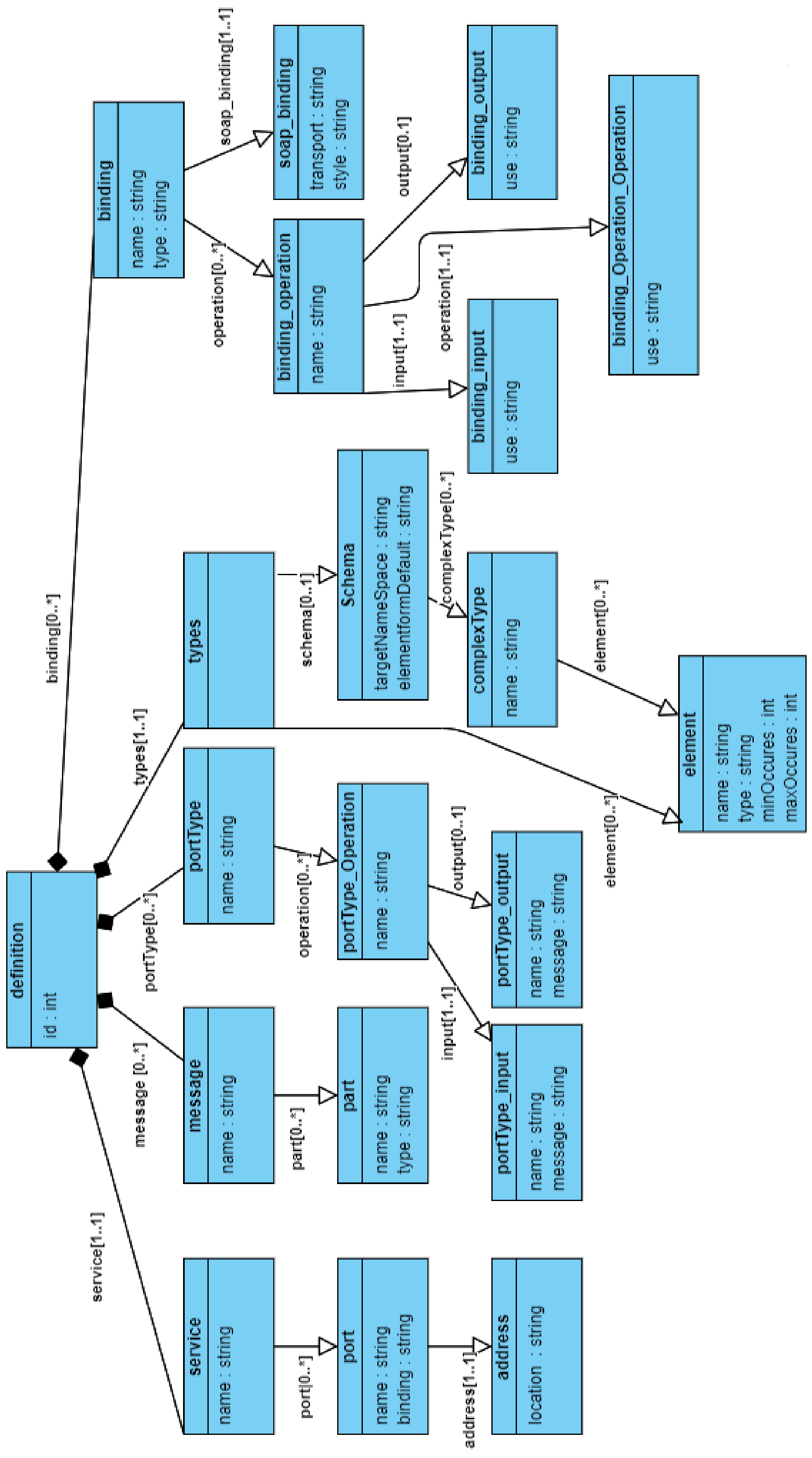


FIGURE IV.4 – WSDL class diagram.

The goal here is to transform a WSDL file that has the XML format into a WSDL object The WSDL object architecture corresponds to a WSDL file Each WSDL file contains one main element "definition" which by itself contains the "Service" element that contains the how many "Port" we have each port has its specific "location" by which the service element is done. "definition" element contains also "Binding" element that specifies the format of each operation XML wise, and "PortType" tell which "Message" element is associated to Operation, while finally "Types" element contains the definition of each type and element mentioned in the "Message" element.
Meanwhile, the WADL object architecture corresponds to a WADL file. Each WADL file contains 'application' and 'resources'. The service is described using a set of 'resource' elements. Each 'resource' contains elements of "parameters" to describe the "inputs" and "methods" that describe the "response" and "request" of a resource. Each answer has its "presentation", which describes the representation of the service response The next step is to convert the WSDL object to a WADL object respecting mapping rules coming from mapping rule Storage described in table IV.3, the first column of this table represents WSDL elements, the equivalent WADL elements are represented in the second column, and the third column represents mapping rules from WSDL to WADL, the equivalent of "Resources" is the "Port" element of WSDL so the resources-base is Port_address_Location and so on we are going to use these rules in an algorithm to implement this solution in our system.NOTE(semantic engine should be used to generate method names as GET, POST, PUT, DELETE).

WADL	**WSDL**	**Mapping**
Resources	Port	Resouces_base=Port_Address_Location
Resource	PortType_Operation	Resource_Path=PortType_Operation_Name
Method	-	Method_Id=PortTypeOperatio_Name
Method	-	Method_Name=generatedFromeOperation_Name
Request	Operation_Input	-
Response	Operation_Output	-
Param	Element	Pram_name=element_name
Param	Element	Param_type=element_type

TABLE IV.3 – Mapping rules from WSDL to WADL.

After all, this being said the rules will be used according to the next algorithm 1 that helps transform the WSDL object into the WADL object.

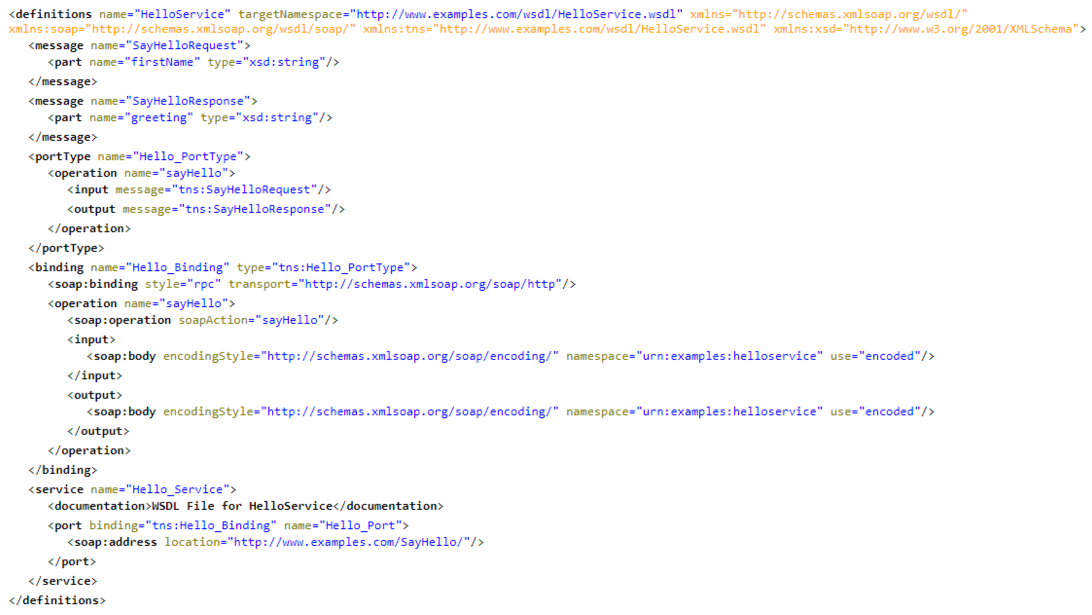
Algorithm 1 Transforamtion from WSDL to WADL pseudo algorithm.

```
Require: $wsdl \leftarrow$ mapped from user WSDL file ;
Ensure: $wsdl \neq null$ ;
 1: $wadl$ ←create new emplty WADL object ;
 2: $wadl.Resources.base \leftarrow wsdl.service.port.location$;
 3: while $wsdl$ has PortType_Operation do
 4:     $wadl.Resources.Resource.path \leftarrow wsdl.portType.operation.name$ ;
 5:     $wadl.Resources.Resource.method.id \leftarrow wsdl.portType.operation.name$ ;
 6:     $wadl.Resources.Resource.method.name$ ← this will be generated from operation-
    Name ;
 7:     if $wadl.Resources.Resource.method.name = $ GET then
 8:         $wadl.Resources.Resource.method.request.param$ ←wsdl.portType .opera-
    tion.input.message.element ;
 9:     end if
10:     if $wsdl.porType.operatoin.output \neq true$ then
    ▷ test if there is an output to create Response
    $wadl.Resources.Resource.method.response$ ← create response object ;
11:     end if
12: end while
```

To show how our solution theoretically and display the wanted result we will use a simple example to show the main points of our transformation this example we are used to demonstrate all the remaining steps. This phase covers the step mentioned in the architecture from 1 to 7

1. In the Figure IV.5 you can see the WSDL file that the service provider will enter.

   This is a hello world service that takes a first name as input and returns "hello first-name" as stated in the WSDL file.



```xml
<definitions name="HelloService" targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl" xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <message name="SayHelloRequest">
      <part name="firstName" type="xsd:string"/>
   </message>
   <message name="SayHelloResponse">
      <part name="greeting" type="xsd:string"/>
   </message>
   <portType name="Hello_PortType">
      <operation name="sayHello">
         <input message="tns:SayHelloRequest"/>
         <output message="tns:SayHelloResponse"/>
      </operation>
   </portType>
   <binding name="Hello_Binding" type="tns:Hello_PortType">
      <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="sayHello">
         <soap:operation soapAction="sayHello"/>
         <input>
            <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:examples:helloservice" use="encoded"/>
         </input>
         <output>
            <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:examples:helloservice" use="encoded"/>
         </output>
      </operation>
   </binding>
   <service name="Hello_Service">
      <documentation>WSDL File for HelloService</documentation>
      <port binding="tns:Hello_Binding" name="Hello_Port">
         <soap:address location="http://www.examples.com/SayHello/"/>
      </port>
   </service>
</definitions>
```

FIGURE IV.5 – WSDL entered by the service provider.

2. In the Figure IV.6 is the mapped WSDL file inside the WSDL object as we can see this object has two message element and no type element because it’s simple so it doesn’t have any complexe types.

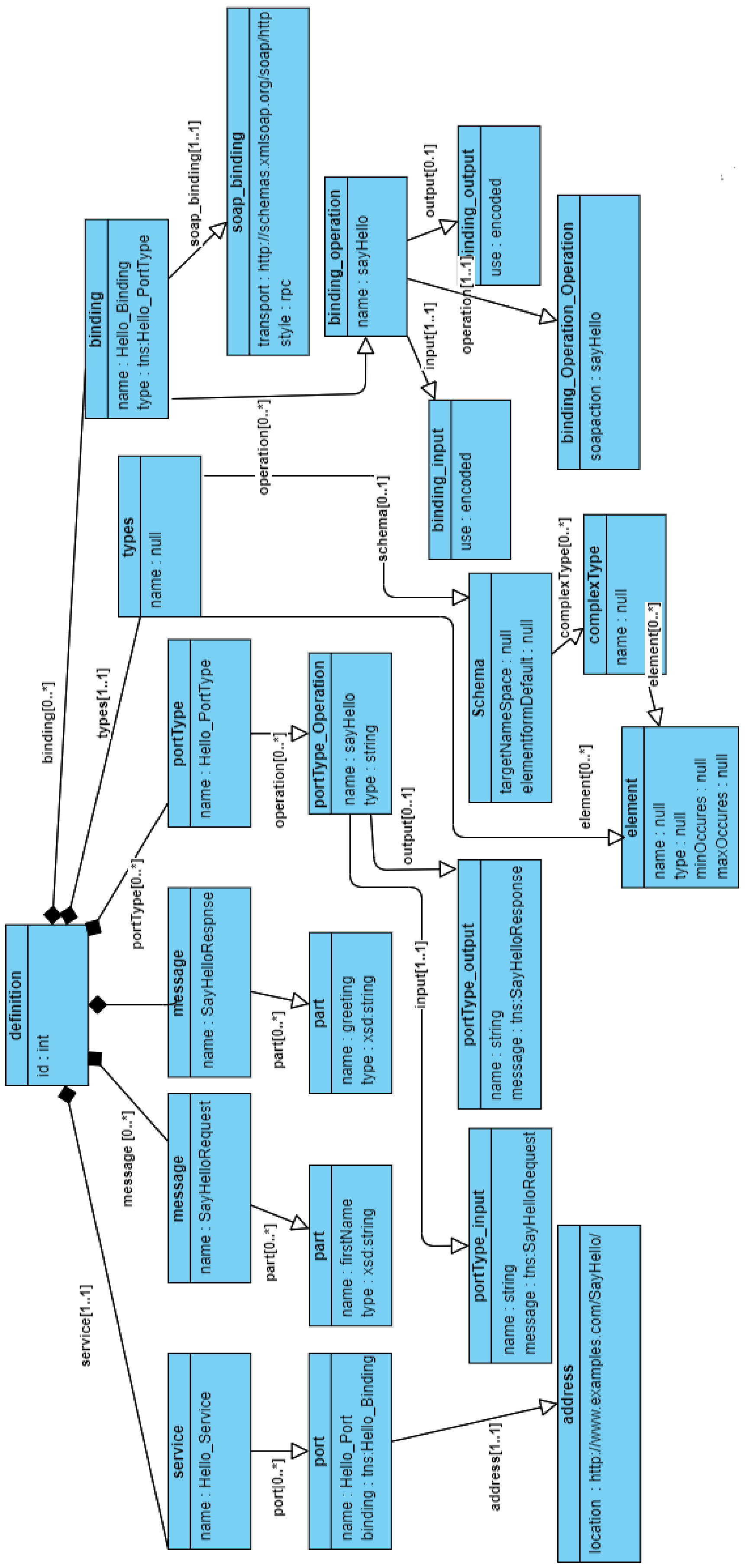


FIGURE IV.6 – Mapping WSDL file information inside WSDL object.

3. After that is done now using the WSDL to WADL algorithm mentioned before(See Algorithm 1) by using it we will create a new WADL object and map the information from the WSDL object To WADL using the earlier mentioned rules we will have the following object(See FigureIV.7).

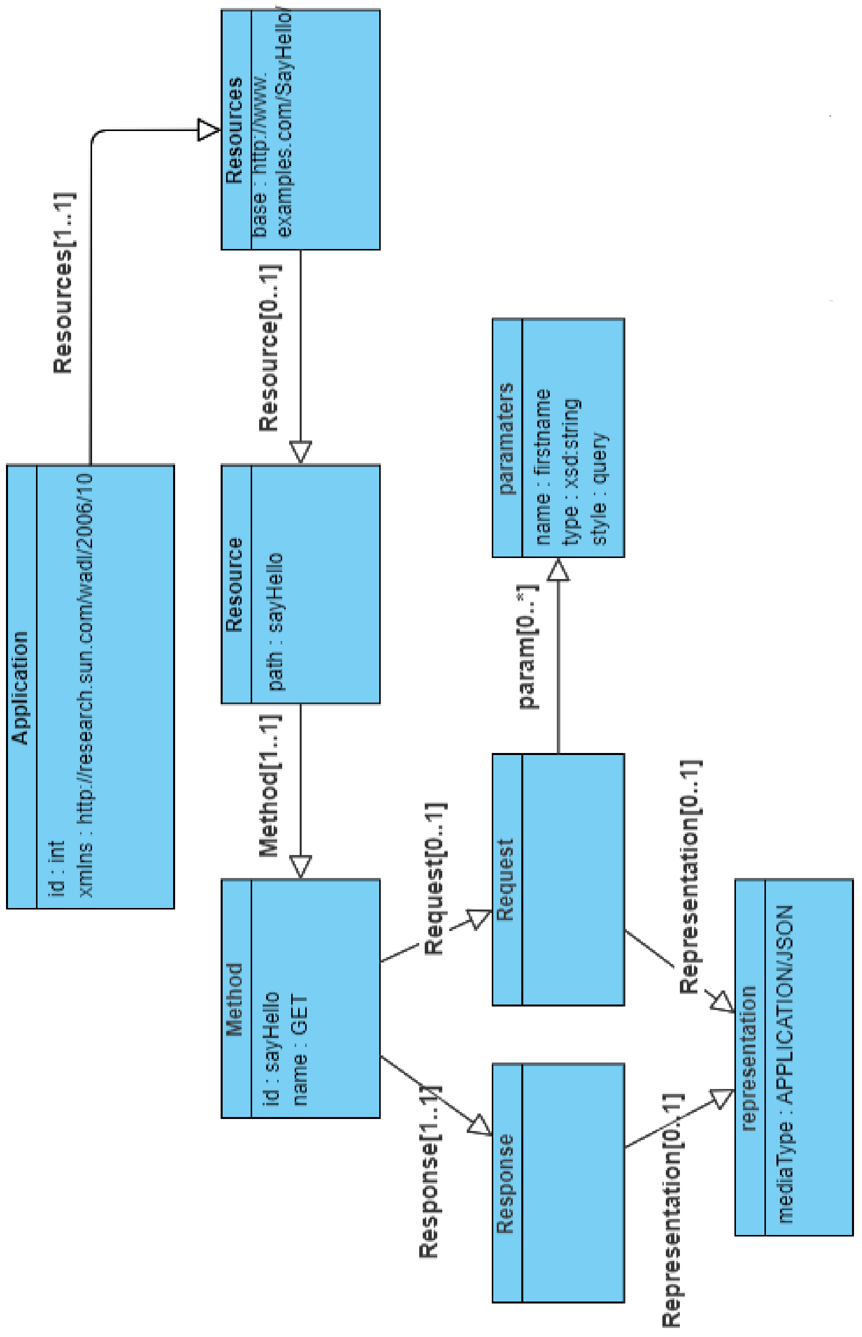
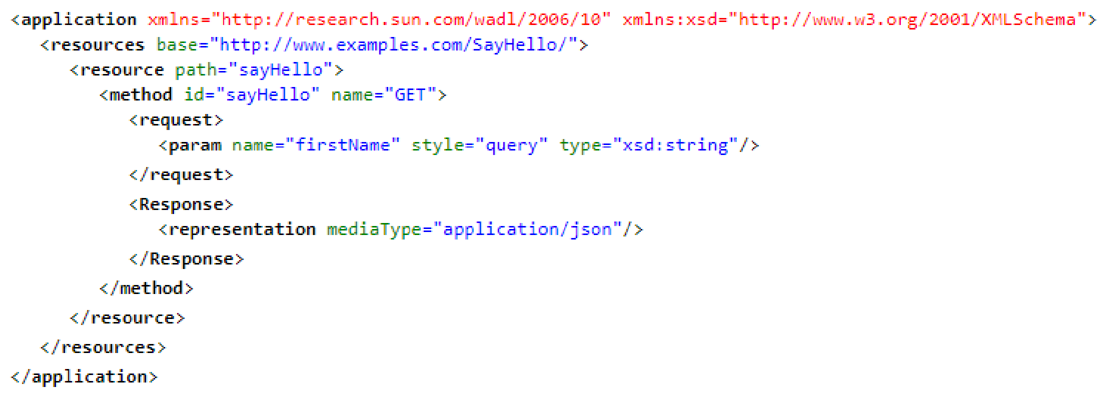


FIGURE IV.7 – Generated WADL object from WSDL.

We can notice that the information will be matched this way according to the matching table, the WSDL_location IV.6 is the same as WADL_base IV.7 , and the method names and parameters we have part name=firstName type=xsd : string since the type of port is simple we don't need to create an element to nest it in that's why it's directly turning into pram name= firstname, and it’s type=xsd : string too, but the style is a query that because the method is GET according to the context so we can’t create body any information attached to the request need to be mentioned in the URL and we think that the query method is the most simple one to handle later in the conversion.

4. Now will generate the information that will hold the information about SOAP message format in our example we have the target name space link along with the some additional information concerning the format.
5. Saving the WSDL file in the same format tha was recieved in it for any future changes.
6. Save the WADL file in XML format for any future requests (See FigureIV.8).



```
<application xmlns="http://research.sun.com/wadl/2006/10" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <resources base="http://www.examples.com/SayHello/">
      <resource path="sayHello">
         <method id="sayHello" name="GET">
            <request>
               <param name="firstName" style="query" type="xsd:string"/>
            </request>
            <Response>
               <representation mediaType="application/json"/>
            </Response>
         </method>
      </resource>
   </resources>
</application>
```

FIGURE IV.8 – Generated WADL object from WSDL.

7. Save the SOAP service detailes in the database.

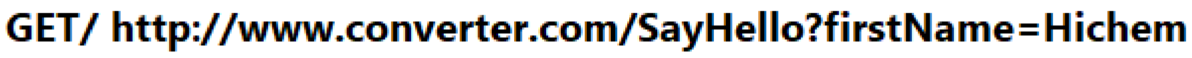
IV.3.1.4 REST to SOAP transformation :

This transformation is the costing operation regarding the information that must be added because REST is lightweight compared to SOAP and the transformation is different from one method to another so we created the next algorithm (See Algorithme2)to achieve this transformation.
This is the algorithm we are going to use to transform the following GET request, we will use the same service example from before. according to the generated WADL (See Figure IV.8), we will send GET request to described service with variable firstName in query style.

Algorithm 2 REST to SOAP transformation algorithme

```
Require: client send REST request to one of the registered SOAP microservices.
 1: if $Method.verb = GET$ then
    ▷ GET method doesn't have any body so it needs special treatement
 2:     $variables[]$ ← exctract data from url query ;
 3:     $XML_request$ ← convertVariablesToXMlFormat($variables[]$) ;
 4: else if $Method.Verb = POST$ then
 5:     $XML_Request$ ← convertVariablesToXMlFormat($REST_Request$) ;
 6: end if
 7: $soap\_Service\_Detailes$ ← retrieve the service detailes from database ;
 8: $soap\_envelope$ ←createSoapenvlope($soap\_Service\_Detailes$) ;
    ▷ parse the information inside the SOAP envelope
 9: $soap\_envelope ← soap\_envelope + XML_Request$ ;
10: $sendTheSOAPMessageToClient()$;
    ⋮
```

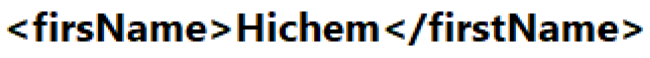
8. Client selects the wanted SOAP service and sends a REST message in our case there is only one which is the sayHello service and send REST request that looks like the Figure IV.9 , we will spose that our service link is //http :www.converter.com.



GET/ http://www.converter.com/SayHello?firstName=Hichem

FIGURE IV.9 – REST request for sayHello servive through our converter.

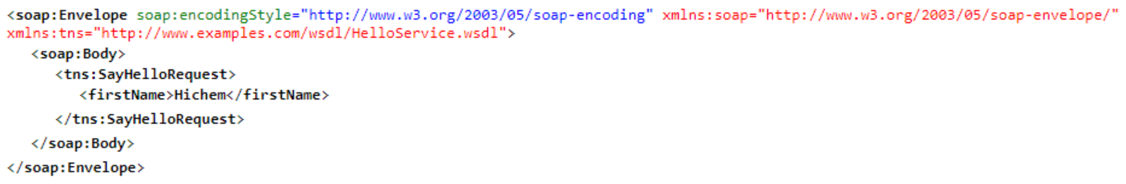
9. Generate Xml version of message , here the message is firstName=Hichem so by converting it we would get this result (See Figure IV.10).



```
<firsName>Hichem</firstName>
```

FIGURE IV.10 – XML version of REST request for sayHello servive through our converter.

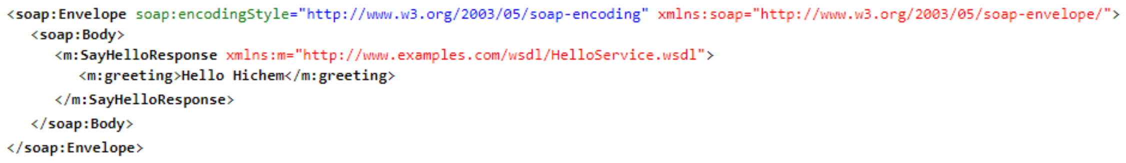
10. Get SOAP service details from the database, which means we will extract them by service name then we would need like target namespace to recognize the elements service link to send the generated SOAP request.
11. Form SOAP message and encapsulate it in SOAP envelope schema using the information provided by the SOAP service details, following these steps we would have well structured SOAP envelope as we mentioned in the second chapter.



```
<soap:Envelope soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding" xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl">
   <soap:Body>
      <tns:SayHelloRequest>
         <firstName>Hichem</firstName>
      </tns:SayHelloRequest>
   </soap:Body>
</soap:Envelope>
```

FIGURE IV.11 – SOAP version of REST request.

12. The SOAP client will send SOAP requests to the specific SOAP web server. in this case the URL is http ://www.examples.com/sayHello.
13. The web server will respond with a SOAP response(See Figure IV.12).



```
<soap:Envelope soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding" xmlns:soap="http://www.w3.org/2003/05/soap-envelope/">
   <soap:Body>
      <m:SayHelloResponse xmlns:m="http://www.examples.com/wsdl/HelloService.wsdl">
         <m:greeting>Hello Hichem</m:greeting>
      </m:SayHelloResponse>
   </soap:Body>
</soap:Envelope>
```

FIGURE IV.12 – SOAP response from SOAP service.

IV.3.1.5 SOAP to REST transformation :

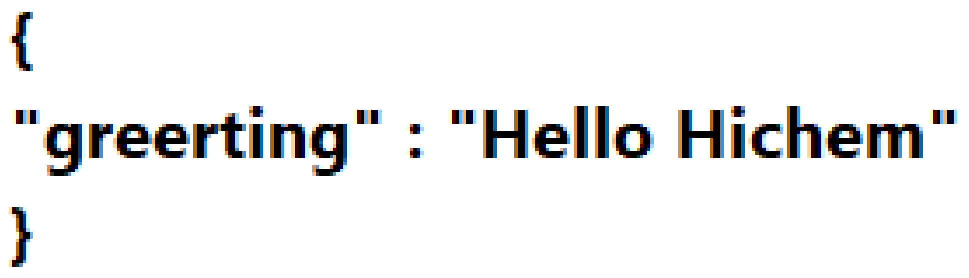
This transformation is the last piece of our interoperability solution after the SOAP service returns its response to the converter there is still some work to be done to get to the client.
We needed to propose an algorithm to help transform from SOAP to REST just put in mind that this transformation is not as expensive data-wise compared to REST to SOAP transformation (See Algorithm3).

Algorithm 3 REST to SOAP transformation algorithm

```
:
Require: SOAP service send SOAP response SOAP client in our system.
1: $XML_document$ ←parse the response in XML document ;
2: remove the SOAP-envelope element and and SOAP-body ;
3: remove the prefix from the variables names ;
4: remove the root element if exists ;
5: convert the resulting xml document into json ;
6: convert JSON document into string ;
7: send the JSON string back to sending client ;
```

14. The SOAP client will receive the response(IV.12) and send it to Rest TO SOAP.

15. REST to SOAP will convert the SOAP message to JSON and remove all the additional information using the proposed algorithm the SOAP response IV.12 we be turned into the following JSON (See Figuire IV.13).



```
{
"greerting" : "Hello Hichem"
}
```

FIGURE IV.13 – REST version of SOAP response from SOAP service.

16. REST response IV.13 will be sent to the client.

By arriving, at this step, we would be done with our solution of interoperability now we will jump into the next section where we will discuss the problem of portability and discuss our proposed solution.

IV.3.2 Portability solution :

In the previous section, we gave our vision on how to solve the problem of interoperability between SOAP and REST APIs but that doesn't enough when it comes to the portability of data because of the lack of global naming convention when it comes to APIs.

IV.3.2.1 General explanation of the solution :

Our proposed solution concerning portability is aimed at REST Microservice only, the solution has a flow of steps that need to be accomplished to start working but for now, we will give an overview of how it works :
The API owner will register his service in our system and the client will search for a service that satisfies his need if not found, here we will use the semantic and syntactic matching between them, and recommend the best suitable service for him.

IV.3.2.2 The measure of syntactic similarity Methods :

This is the method used in calculating the similarity between two words based on their syntax only no semantics is included. We will use two methods only we will explain each one of them :

— **Dice similarity coefficient :**

The dice similarity coefficient is defined as the total number of items multiplied by the number of common objects. As a result, the Dice measurement is defined by the following formula.

$$DiceCoefficient(A, B) = \frac{2 * |A \cap B|}{|A| + |B|}$$

The result of this formula is confined between 0 and 1 the nearest value is 1 this means that the words A and B are more similar, in our case it means the input is similar to the output.

Example of how dice coefficient work :

Let's suppose that the client has "userName" and "userPassword" as input while the registered service in our system is "name" and "password" using the dice coefficient method the result would be like this : we have userName=8 and userPassword =12 while name =4 and password = 8 we will see the workflow of one pair and the rest will be shown in the table we have name and username have 4 letters in commun so the formula would be like this :

$$DiceCoefficient(userName, name) = \frac{2 * |4|}{|8| + |4|} = 0.66$$

Now we do this wich each pair we would have the result in the following table IV.4 :

	name	password
userName	0.66	0
userPassword	0	0.8

TABLE IV.4 – Dice method example.

Based on the previous result the matching result would be (name,userName) and (password,userPassword) this is how mainly our method will work.

— **Jaccard Index :**

The Jaccard similarity index (sometimes called the Jaccard similarity coefficient) compares members for two sets to see which members are shared and which are distinct. It's a measure of similarity for the two sets of data, with a range from 0% to 100%.
The formula to find the Index is :

$$JaccardIndex = \frac{(the - number - in - both - sets)}{(the - number - in - either - set)} * 100$$

The same formula in notation is :

$$JaccardIndex(A, B) = \frac{|A \cap B|}{|A| \cup |B|} * 100$$

Example of how Jaccard Index work :

We will use the same example used in the dice method we will have the following result with the pair (name,userName) : userName=8 and name = 4 and name ∩ userName= 4 while name ∪ userName = 9 so the formula would be like :

$$JaccardIndex(userName, name) = \frac{|userName \cap name|}{|userName| \cup |name|} * 100$$

$$JaccardIndex(userName, name) = \frac{4}{9} * 100 = 44.44$$

)

Now we do this wich each pair we would have the result in the following table (See Table IV.5) :

	name	password
userName	44.44	0
userPassword	0	66.66

TABLE IV.5 – Jaccard Index method example.

IV.3.2.3 The measure of semantic similarity methods :

The semantic measure method requires external tools that use taxonomy to calculate the similarity between words based on their meaning, not syntax, as we said we need an external resource that helps do that we decided to wordnet.

WordNet is the product of a research project at Princeton University . It is a large lexical database of English. In WordNet Nouns, verbs, adverbs, and adjectives are organized by a variety of semantic relations into synonym sets (synsets), which represent one concept. Examples of semantic relations used by WordNet are synonymy, autonomy, hyponymy, member, similar, domain and cause, and so on.[63]

Semantic similarity metrics might be used for tasks like word disambiguation, text segmentation, and evaluating ontologies for consistency or coherency. Many proposals have been made. Overall, all of the measurements fall into four categories : path length-based measures, information content-based measures, feature-based measures, and hybrid measures.

— **Path-based measures :**

The main idea of path-based measures is that the similarity between two concepts is a function of the length of the path linking the concepts and the position of the concepts in the taxonomy and one of these Measures is Wu & Palmer's Measure.

- ■ **Wu & Palmer's measure[63] :**

  Wu and Palmer introduced a scaled measure . This similarity measure takes the position of concepts c1 and c2 in the taxonomy relatively to the position of the most specific common concept lso(c1,c2) into account. It assumes that the similarity between two concepts is the function of path length and depth in path-based measures.

  $$SimWup(c1, c2) = \frac{2 * depth(lso(c1, c2))}{len(c1, c2) + 2 * depth(lso(c1, c2))}$$

  From formula above it is noted that,
  The similarity of two concepts (c1, c2) depends on their distance and the lowest common subsumer (lso(c1,c2)).
  Simple example :
  SimWup(Name,Surname)=0.93

— **Information Content-based Measure :**

It was anticipated that each notion in WordNet has a large amount of information. Measures of similarity are based on each concept's information content The more typical information two The more concepts that share, the more related the concepts are.

- ■ **Lin's measure[54] :**

  Lin proposed another method for similarity measure :

  $$SimLin(c1, c2) = \frac{2 * IC(lso(c1, c2))}{IC(c1) + IC(c2)}$$

  It utilizes both the quantity of information required to indicate the similarities between the two ideas and the amount of information required to properly define these words.
  From formula above it is noted that,
  The measure has taken the information content of compared concepts into account respectively. As IC(lso(c1,c2)) <=IC(c1) and IC(lso(c1,c2)) <=IC(c2), therefore the values of this measure vary between 1 and 0.
  Simple example :
  SimWup(Name,Surname)=0.0

We just mentioned only two methods even though there are still 2 more types of measures and each type has a different method, the already mentioned methods are going to be used in our solution and we will see how in our next section.

IV.3.2.4 Result aggregation :

After the calculation of each input is done using the previous method both semantic and syntactic now we will prpose a method to choose the final result based on all of them , the final semilarity value is going to be calculated using the follwing formula :

$$SimFinal = \frac{\alpha 1 * simSemantic + \alpha 2 * simSyntactic}{\alpha 1 + \alpha 2}$$

Where :

$$simSemantic = \frac{simLin + simWuPalmer}{2}$$

$$simSyntactic = \frac{simDice + simJaccard}{2}$$

AND $\alpha 1$,$\alpha 2$ both a weight to add.

Threshold

we need to define a threshold to distinct which APIs are similar and which ones are not , after that we will use all the above calculation and method as it is shown in the Algorithm4 below.

Algorithm 4 Portability algorithm

```
⋮
 1: Retrieve all service inputs and outputs;
 2: $input\_list$ ← all inputs;
 3: $output\_list$ ← all outputs;
 4: while input_list≠ null do
 5:     while output_list≠ null do
 6:         $similarity\_value$ ←Calculate the similarity between each input and all outputs;
 7:     end while
 8: end while
 9: if $similarity\_value \geq threshold$ then
10:     Store the results in list;
11: end if
12: Send the client the list of the service compatible with his requirements;
    ⋮
```

After all this is done now the client will choose the API he wants and the mapping to that API will be stored for offline mapping, now we will show how these prementioned solutions will work together in the next section.)

IV.3.3 Combining the solutions :

We would recommend that these solutions work in cloud environments in microservice architecture to enhance the elasticity and efficiency of these services along with the most use of them, along with registering the SOAP APIs in the Convert system first and then registering the outcoming REST endpoint inside the Portability System.

IV.4 Conclusion :

In this chapter, we have introduced our solutions regarding the interoperability and portability of the API in REST and SOAP environments, we ended up proposing two different solutions but regarding that, they are meant for the cloud it would be easy to combine. And in the next section, we will show our implementation.

Chapter V

Implementation of the proposed solution

V.1 Introduction :

In the previous chapter, we introduce the vision of our solution now we will show the hardware as the software used to bring this solution to life without forgetting screenshots of the examples used in this example and their programming interface.

V.2 Hardware resources :

To implement our solution we used resources that have the following characteristics :

- Laptop hp 250 g6 notebook pc.
- Processor Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.70 GHz.
- RAM 8.00 GB.
- Operating System Microsoft Windows version 21H1.

V.3 Development tools and environment :

In this section, we will present the tools and IDEs that have been used to achieve this step :

Eclipse[2]

Eclipse is a free, Java-based development platform known for its plugins that allow developers to develop and test code written in other programming languages. Eclipse is released under the terms of the Eclipse Public License.

Eclipse IDE was initially overseen by a consortium of software vendors seeking to create and

2. https ://www.techtarget.com/searchapparchitecture/definition/Eclipse-Eclipse-Foundation

foster a new community complementing Apache's open-source community.

JDK [1]

The JDK includes tools for developing and testing programs written in the Java programming language and running on the Java platform. Linux ; macOS ; Windows.

JAVA [2]

The Java™ Programming Language is a general-purpose, concurrent, strongly typed, class-based object-oriented language. It is normally compiled to the bytecode instruction set and binary format defined in the Java Virtual Machine Specification.

Spring framework [3]

The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications - on any kind of deployment platform.
A key element of Spring is infrastructural support at the application level : Spring focuses on the "plumbing" of enterprise applications so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments.

Mysql [4]

MySQL is the world's most popular open-source database. According to DB-Engines, MySQL ranks as the second-most-popular database, behind Oracle Database. MySQL powers many of the most accessed applications, including Facebook, Twitter, Netflix, Uber, Airbnb, Shopify, and Booking.com.
Since MySQL is open source, it includes numerous features developed in close cooperation with users over more than 25 years. So it's very likely that your favorite application or programming language is supported by MySQL Database.

SQL [5]

SQL is an abbreviation for Structured Query Language, which allows you to access and alter databases. SQL was adopted as an American National Standards Institute ("ANSI") standard in 1986 and as an International Organization for Standardization (ISO) standard in 1987.

Insomnia [6]

Insomnia is a free cross-platform desktop application that takes the pain out of interacting with and designing HTTP-based APIs. Insomnia combines an easy-to-use interface with advanced functionality like authentication helpers, code generation, and environment variables.

Docker [7]

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your

1. https ://www.oracle.com/java/technologies/downloads/
2. https ://docs.oracle.com/
3. https ://spring.io/projects/spring-framework
4. https ://www.oracle.com/mysql/what-is-mysql/
5. https ://www.w3schools.com/sql/sql_intro.asp
6. https ://docs.insomnia.rest/insomnia/get-started
7. https ://docs.docker.com/get-started/overview/

applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

V.4 Our implementation :

In this section, we will show the architecture of our implementation of the solution, since our code is meant to work in a cloud environment we used microservice architecture as shown in the Figure below V.1. For the simplicity of the examples, we have neglected a few characteristics as the authentication as well as the GUI because the software is intended for other systems, not for human interaction. the architecture to simulate API calls we used Insomnia software as we mentioned in the previous section.

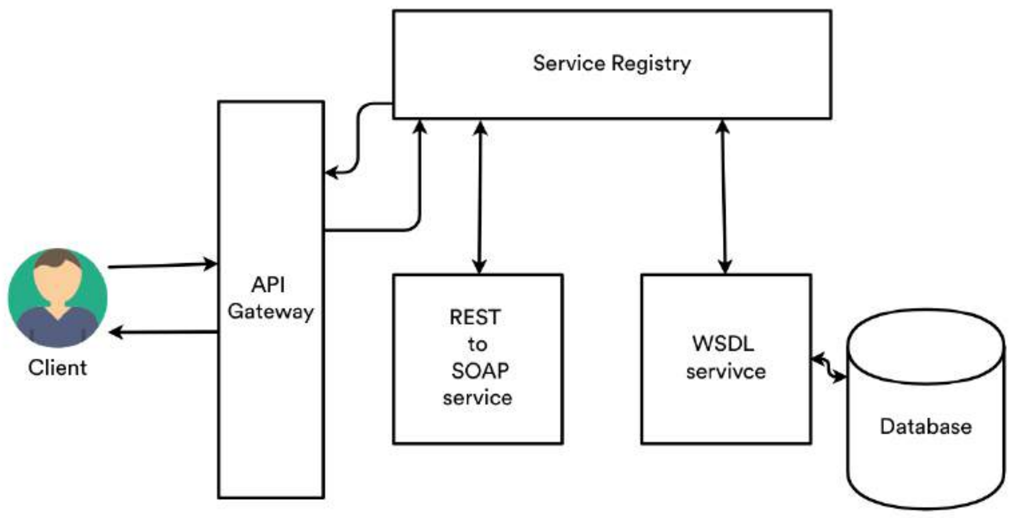


FIGURE V.1 – System microservice architecture.

As the architecture shows we have :

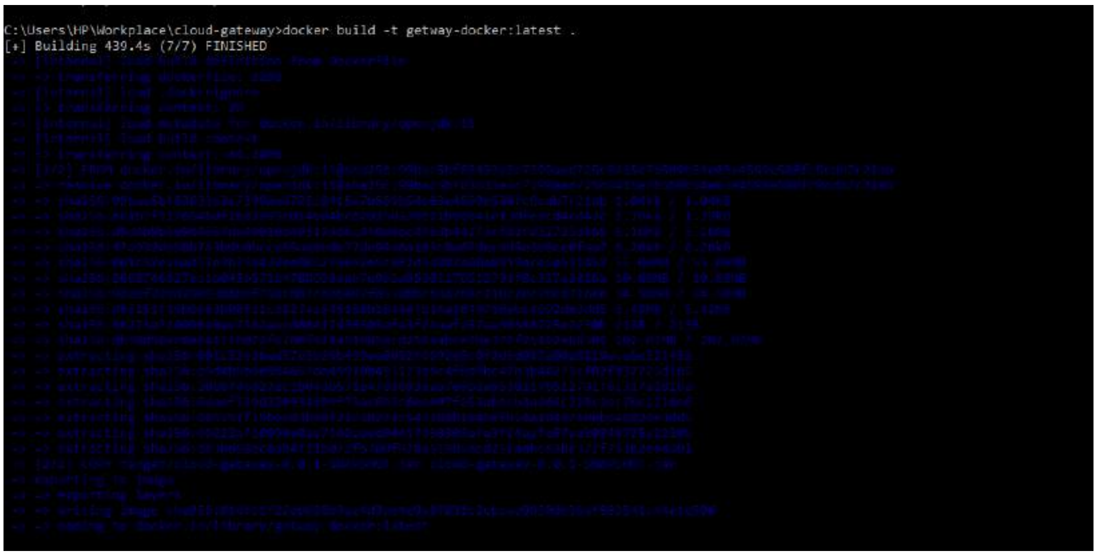
•Client :
Here client can mean both service provider or normal client in general it means any interaction with our system.

•API Gateway :
As the name explains it's the main gate for our system the user has to know its location only and it will handle the mapping request to the responsible service.

•Service registry :
This service is responsible for the registry of services which means any service have to register in this registry so it can work, almost the whole concept of microservices is handled by him it helps escalate application you can have multiple instances from the same service to handle

workloads as needed without additional configuration or programming.
•WSDL service :
This is the service responsible for receiving WSDL file and generating wall equivalent, also generate each service endpoint information that helps generate SOAP message.
•REST to SOAP service :
This service is responsible for converting REST Requests to SOAP requests as well as responses.

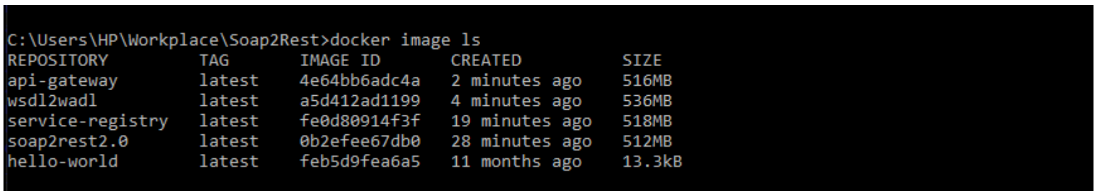
Now we will see a real example of how our system works as we mentioned several times our solution is aimed at a cloud environment so we used Docker for the virtualization we have to create a docker image for all the microservice in our architecture .
Figure V.2 show how to create a docker image of our API-gateway microservice.



```
C:\Users\HP\Workplace\cloud-gateway>docker build -t getway-docker:latest .
[+] Building 439.4s (7/7) FINISHED
[illegible]
```

FIGURE V.2 – API-gateway docker image creation.

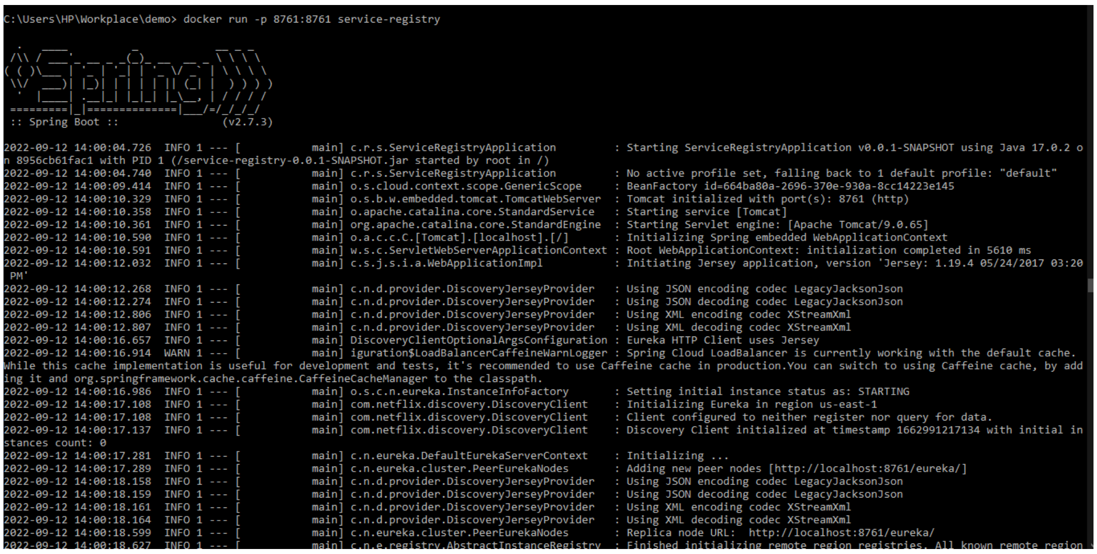
After the creation of every microservice image the same way we did now we will have the following list of docker images (See FigureV.3).



```
C:\Users\HP\Workplace\Soap2Rest>docker image ls
REPOSITORY         TAG       IMAGE ID       CREATED          SIZE
api-gateway        latest    4e64bb6adc4a   2 minutes ago    516MB
wsdl2wadl          latest    a5d412ad1199   4 minutes ago    536MB
service-registry   latest    fe0d80914f3f   19 minutes ago   518MB
soap2rest2.0       latest    0b2efee67db0   28 minutes ago   512MB
hello-world        latest    feb5d9fea6a5   11 months ago    13.3kB
```

FIGURE V.3 – Docker image list .

Now that we created these images we have to run them so that we can use them in Figure V.4 is how to launch a registry service microservice.



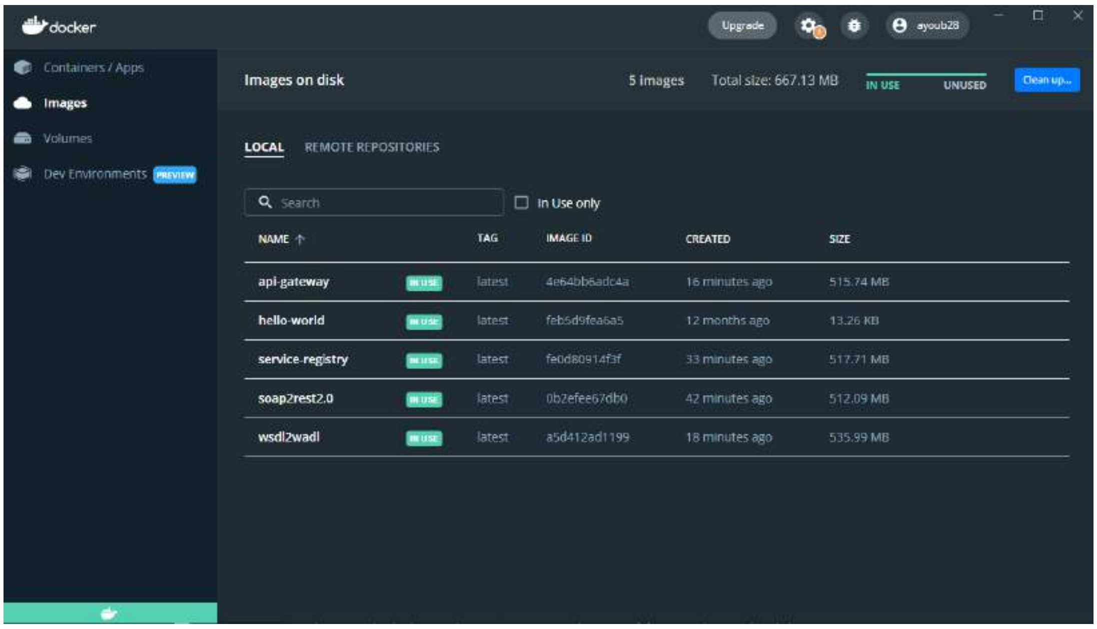
```
C:\Users\HP\Workplace\demo> docker run -p 8761:8761 service-registry

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.3)

2022-09-12 14:00:04.726  INFO 1 --- [           main] c.r.s.ServiceRegistryApplication         : Starting ServiceRegistryApplication v0.0.1-SNAPSHOT using Java 17.0.2 o
n 8956cb61fac1 with PID 1 (/service-registry-0.0.1-SNAPSHOT.jar started by root in /)
2022-09-12 14:00:04.740  INFO 1 --- [           main] c.r.s.ServiceRegistryApplication         : No active profile set, falling back to 1 default profile: "default"
2022-09-12 14:00:09.414  INFO 1 --- [           main] o.s.cloud.context.scope.GenericScope     : BeanFactory id=664ba80a-2696-370e-930a-8cc14223e145
2022-09-12 14:00:10.329  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8761 (http)
2022-09-12 14:00:10.358  INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2022-09-12 14:00:10.361  INFO 1 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-09-12 14:00:10.590  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2022-09-12 14:00:10.591  INFO 1 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 5610 ms
2022-09-12 14:00:12.032  INFO 1 --- [           main] c.s.j.s.i.a.WebApplicationImpl           : Initiating Jersey application, version 'Jersey: 1.19.4 05/24/2017 03:20
 PM'
2022-09-12 14:00:12.268  INFO 1 --- [           main] c.n.d.provider.DiscoveryJerseyProvider   : Using JSON encoding codec LegacyJacksonJson
2022-09-12 14:00:12.274  INFO 1 --- [           main] c.n.d.provider.DiscoveryJerseyProvider   : Using JSON decoding codec LegacyJacksonJson
2022-09-12 14:00:12.806  INFO 1 --- [           main] c.n.d.provider.DiscoveryJerseyProvider   : Using XML encoding codec XStreamXml
2022-09-12 14:00:12.807  INFO 1 --- [           main] c.n.d.provider.DiscoveryJerseyProvider   : Using XML decoding codec XStreamXml
2022-09-12 14:00:16.657  INFO 1 --- [           main] DiscoveryClientOptionalArgsConfiguration : Eureka HTTP Client uses Jersey
2022-09-12 14:00:16.914  WARN 1 --- [           main] iguration$LoadBalancerCaffeineWarnLogger : Spring Cloud LoadBalancer is currently working with the default cache.
While this cache implementation is useful for development and tests, it's recommended to use Caffeine cache in production.You can switch to using Caffeine cache, by add
ing it and org.springframework.cache.caffeine.CaffeineCacheManager to the classpath.
2022-09-12 14:00:16.986  INFO 1 --- [           main] o.s.c.n.eureka.InstanceInfoFactory       : Setting initial instance status as: STARTING
2022-09-12 14:00:17.108  INFO 1 --- [           main] com.netflix.discovery.DiscoveryClient    : Initializing Eureka in region us-east-1
2022-09-12 14:00:17.108  INFO 1 --- [           main] com.netflix.discovery.DiscoveryClient    : Client configured to neither register nor query for data.
2022-09-12 14:00:17.137  INFO 1 --- [           main] com.netflix.discovery.DiscoveryClient    : Discovery Client initialized at timestamp 1662991217134 with initial in
stances count: 0
2022-09-12 14:00:17.281  INFO 1 --- [           main] c.n.eureka.DefaultEurekaServerContext    : Initializing ...
2022-09-12 14:00:17.289  INFO 1 --- [           main] c.n.eureka.cluster.PeerEurekaNodes       : Adding new peer nodes [http://localhost:8761/eureka/]
2022-09-12 14:00:18.158  INFO 1 --- [           main] c.n.d.provider.DiscoveryJerseyProvider   : Using JSON encoding codec LegacyJacksonJson
2022-09-12 14:00:18.159  INFO 1 --- [           main] c.n.d.provider.DiscoveryJerseyProvider   : Using JSON decoding codec LegacyJacksonJson
2022-09-12 14:00:18.161  INFO 1 --- [           main] c.n.d.provider.DiscoveryJerseyProvider   : Using XML encoding codec XStreamXml
2022-09-12 14:00:18.164  INFO 1 --- [           main] c.n.d.provider.DiscoveryJerseyProvider   : Using XML decoding codec XStreamXml
2022-09-12 14:00:18.599  INFO 1 --- [           main] c.n.eureka.cluster.PeerEurekaNodes       : Replica node URL:  http://localhost:8761/eureka/
2022-09-12 14:00:18.627  INFO 1 --- [           main] c.n.e.registry.AbstractInstanceRegistry  : Finished initializing remote region registries. All known remote region
```

FIGURE V.4 – Lunching service Registry microservice.

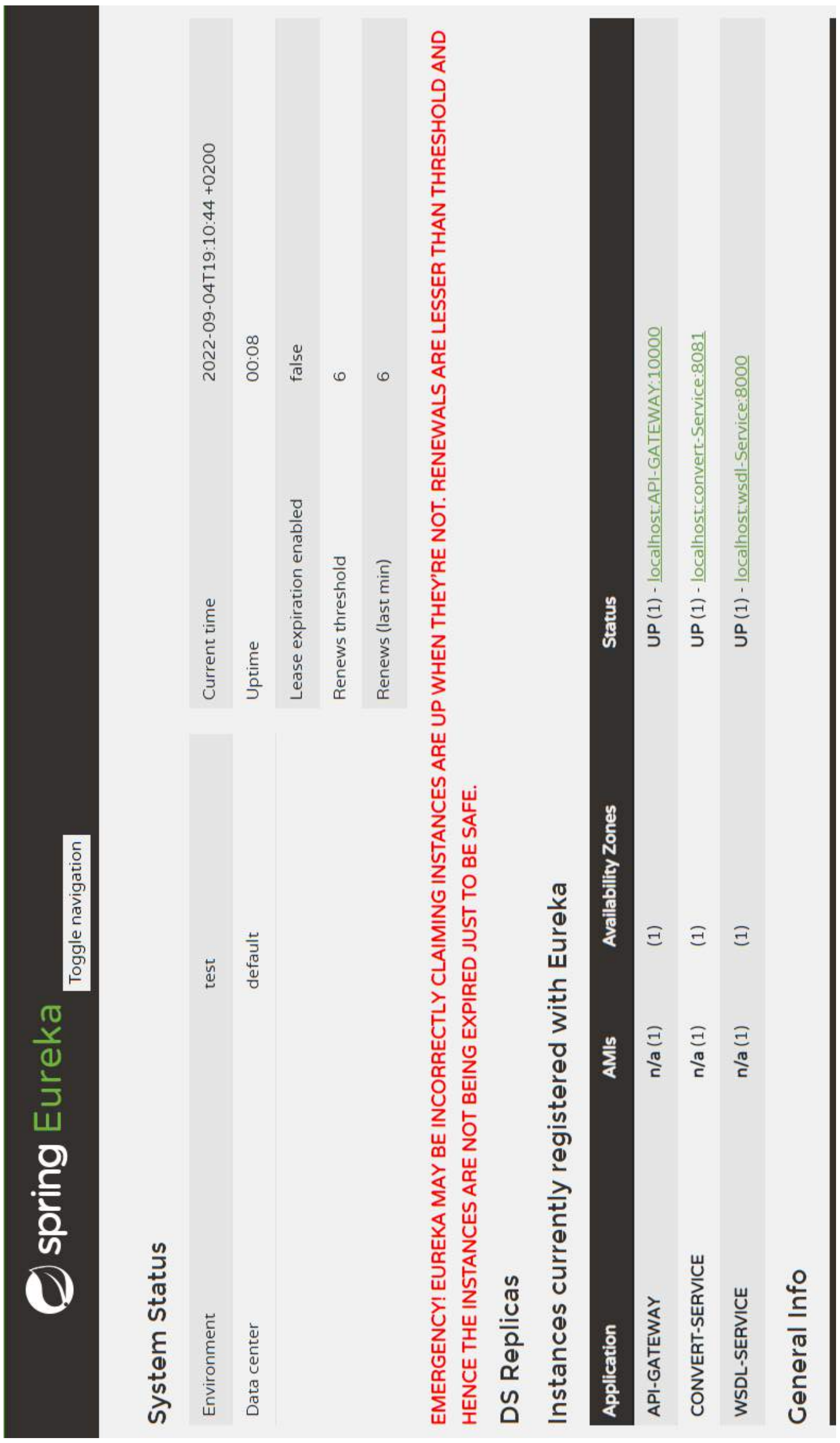
We repeat this step with every microservice image we have the finale result will look like the Figure V.5.



NAME		TAG	IMAGE ID	CREATED	SIZE
api-gateway	IN USE	latest	4e64bb6adc4a	16 minutes ago	515.74 MB
hello-world	IN USE	latest	feb5d9fea6a5	12 months ago	13.26 KB
service-registry	IN USE	latest	fe0d80914f3f	33 minutes ago	517.71 MB
soap2rest2.0	IN USE	latest	0b2efee67db0	42 minutes ago	512.09 MB
wsdl2wadl	IN USE	latest	a5d412ad1199	18 minutes ago	535.99 MB

FIGURE V.5 – Running containers list.

After we launch our microservices we can monitor them from the eureka service registry (See Figure V.6) homepage that allow service to register to avoid scalability of microservices so we can have multiple instances of our microservices it helps keep the URL of each service transparent to the calling microservice. V.6.



System Status

Environment	test	Current time	2022-09-04T19:10:44 +0200
Data center	default	Uptime	00:08
		Lease expiration enabled	false
		Renews threshold	6
		Renews (last min)	6

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

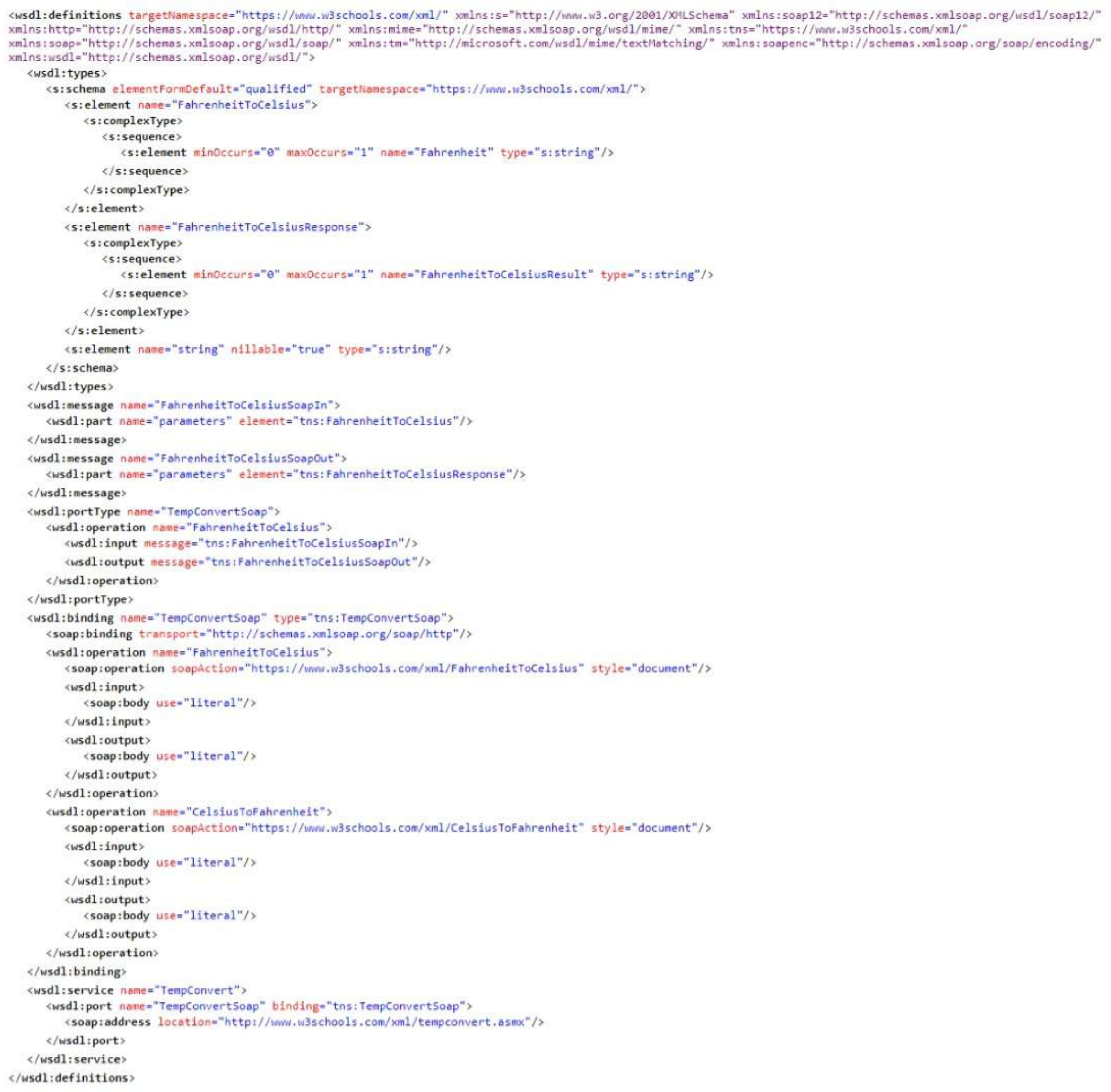
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - localhost:API-GATEWAY:10000
CONVERT-SERVICE	n/a (1)	(1)	UP (1) - localhost:convert-Service:8081
WSDL-SERVICE	n/a (1)	(1)	UP (1) - localhost:wsdl-Service:8000

General Info

FIGURE V.6 – Eureka registry server dashboard.

Since all our services are registered and working we will send requests to our services through our API gateway.
For more explanation we will use the temperature converter provided by W3School can be found in this URL "https ://www.w3schools.com/xml/tempconvert.asmx" we can access the WSDL file of this service by adding " ?WSDL" to the link above the WSDL file is long but for our example, we will summarize it a little bit (See Figure V.7).



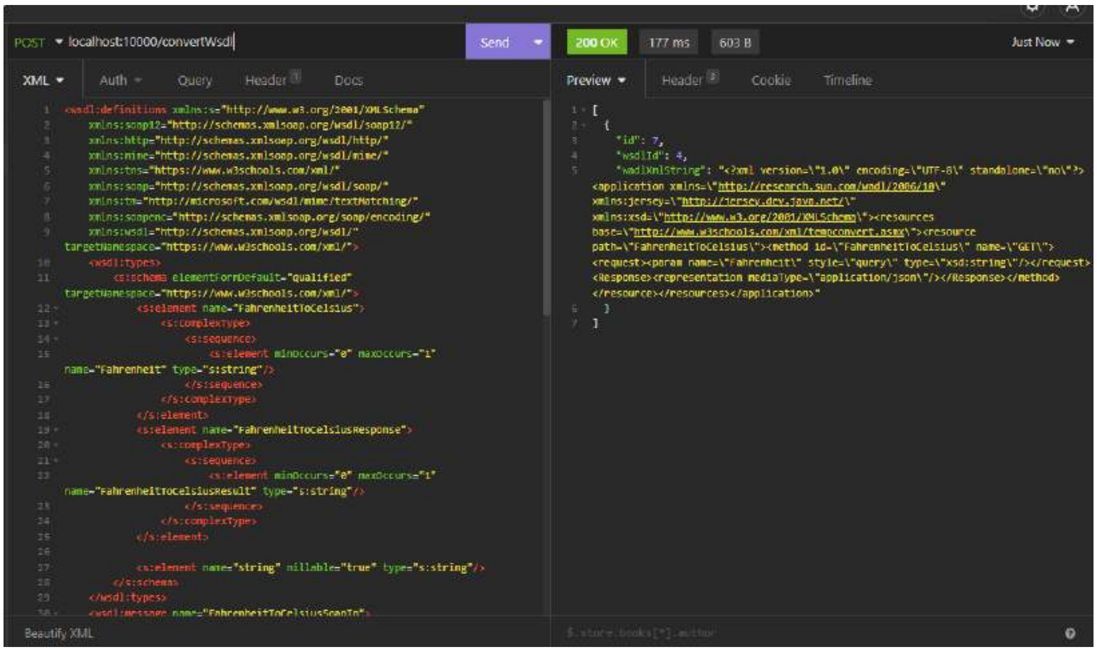
```xml
<wsdl:definitions targetNamespace="https://www.w3schools.com/xml/" xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="https://www.w3schools.com/xml/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="https://www.w3schools.com/xml/">
      <s:element name="FahrenheitToCelsius">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="Fahrenheit" type="s:string"/>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="FahrenheitToCelsiusResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="FahrenheitToCelsiusResult" type="s:string"/>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="string" nillable="true" type="s:string"/>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="FahrenheitToCelsiusSoapIn">
    <wsdl:part name="parameters" element="tns:FahrenheitToCelsius"/>
  </wsdl:message>
  <wsdl:message name="FahrenheitToCelsiusSoapOut">
    <wsdl:part name="parameters" element="tns:FahrenheitToCelsiusResponse"/>
  </wsdl:message>
  <wsdl:portType name="TempConvertSoap">
    <wsdl:operation name="FahrenheitToCelsius">
      <wsdl:input message="tns:FahrenheitToCelsiusSoapIn"/>
      <wsdl:output message="tns:FahrenheitToCelsiusSoapOut"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="TempConvertSoap" type="tns:TempConvertSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="FahrenheitToCelsius">
      <soap:operation soapAction="https://www.w3schools.com/xml/FahrenheitToCelsius" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="CelsiusToFahrenheit">
      <soap:operation soapAction="https://www.w3schools.com/xml/CelsiusToFahrenheit" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="TempConvert">
    <wsdl:port name="TempConvertSoap" binding="tns:TempConvertSoap">
      <soap:address location="http://www.w3schools.com/xml/tempconvert.asmx"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

FIGURE V.7 – Temperature converter WSDL.

After we see that is an example of WSDL now let's suppose the owner of this service is going to subscribe to our system to generate a WADL file as well add provide REST access to SOAP web services, he can do that by following the next steps.

NOTE : To show the workflow of the data between APIs we need to use Insomnia software that helps send and receive costume-made Requests.

Step one send WSDL file as a string to our service note that the address **http ://localhost :10000/** is the address of the API gateway while the location of **/convertWsdl** is, the response will be WADL string with an id of it and the id of its WSDL file as shown in the next screenshot (See Figure V.8).


POST localhost:10000/convertWsdl

```
<wsdl:definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:tns="https://www.w3schools.com/xml/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="https://www.w3schools.com/xml/">
    <wsdl:types>
        <s:schema elementFormDefault="qualified"
targetNamespace="https://www.w3schools.com/xml/">
            <s:element name="FahrenheitToCelsius">
                <s:complexType>
                    <s:sequence>
                        <s:element minOccurs="0" maxOccurs="1"
name="Fahrenheit" type="s:string"/>
                    </s:sequence>
                </s:complexType>
            </s:element>
            <s:element name="FahrenheitToCelsiusResponse">
                <s:complexType>
                    <s:sequence>
                        <s:element minOccurs="0" maxOccurs="1"
name="FahrenheitToCelsiusResult" type="s:string"/>
                    </s:sequence>
                </s:complexType>
            </s:element>

            <s:element name="string" nillable="true" type="s:string"/>
        </s:schema>
    </wsdl:types>
    <wsdl:message name="FahrenheitToCelsiusSoapIn">
```

200 OK 177 ms 603 B

```
[
  {
    "id": 7,
    "wsdlId": 4,
    "wadlXmlString": "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"?>
<application xmlns=\"http://research.sun.com/wadl/2006/10\"
xmlns:jersey=\"http://jersey.dev.java.net/\"
xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"><resources
base=\"http://www.w3schools.com/xml/tempconvert.asmx\"><resource
path=\"FahrenheitToCelsius\"><method id=\"FahrenheitToCelsius\" name=\"GET\">
<request><param name=\"Fahrenheit\" style=\"query\" type=\"xsd:string\"/></request>
<Response><representation mediaType=\"application/json\"/></Response></method>
</resource></resources></application>"
  }
]
```

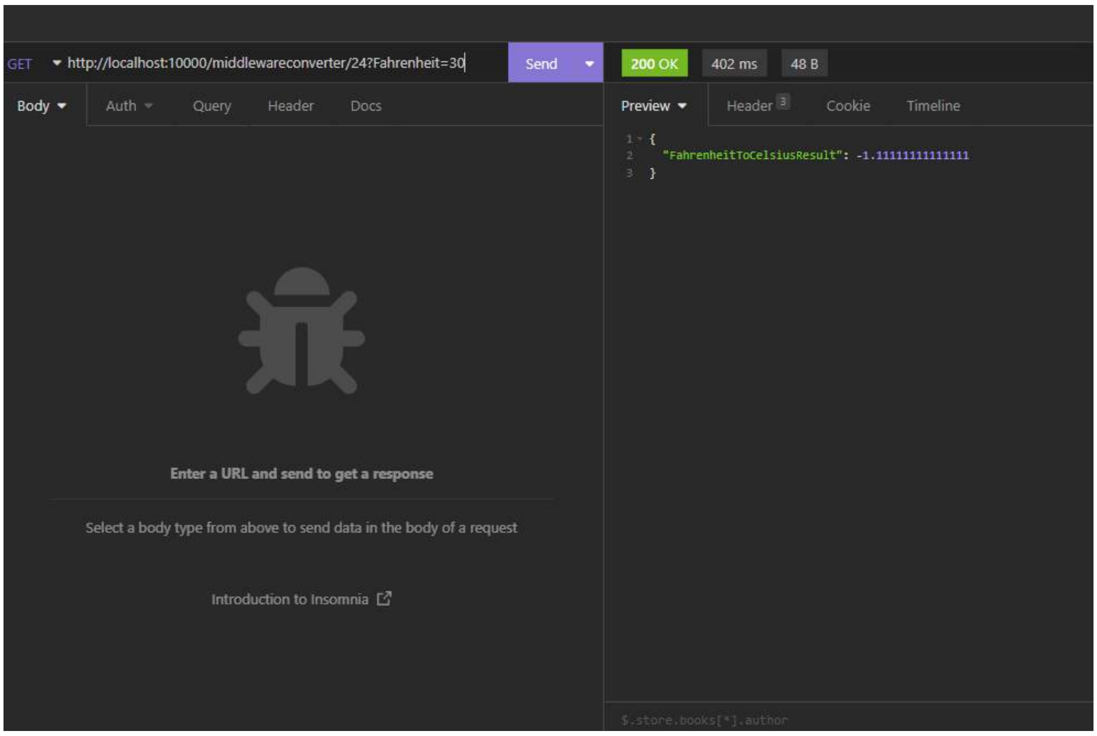
FIGURE V.8 – Publishing WSDL file.

For showing purposes, we will put the WADL file into XML format to see the transformation (See Figure V.9).

```
<application xmlns="http://research.sun.com/wadl/2006/10" xmlns:jersey="http://jersey.dev.java.net/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <resources base="http://www.w3schools.com/xml/tempconvert.asmx">
      <resource path="FahrenheitToCelsius">
         <method id="FahrenheitToCelsius" name="GET">
            <request>
               <param name="Fahrenheit" style="query" type="xsd:string"/>
            </request>
            <Response>
               <representation mediaType="application/json"/>
            </Response>
         </method>
      </resource>
   </resources>
</application>
```

FIGURE V.9 – Generated WADL.

After this, the service provider's job is finished, let's come to the client suppose that the client already bought access to these services, he will have access to the service also through the same gateway used before **http// :localhost :10000/middlewareconverter** as you noticed we changed the location **"/convertWSDL "** to **"/middlewareconverter"**, now we will send REST message to the previous location with adding the location of service will be provided by id **"http// :localhost :10000/middlewareconverter/id"** where **id** is variable.
After that, we will send the request (See Figure V.10) based on the generated WADL the method is GET while the parameter name is "Fahrenheit" and its style is a query which means we will add **" ?Fahrenheit=value"** where value is a string we will choose a random value like 30 and the service id will be 24 so the link will be **http ://localhost :10000/middlewareconverter/24 ?Fahrenheit=30.**

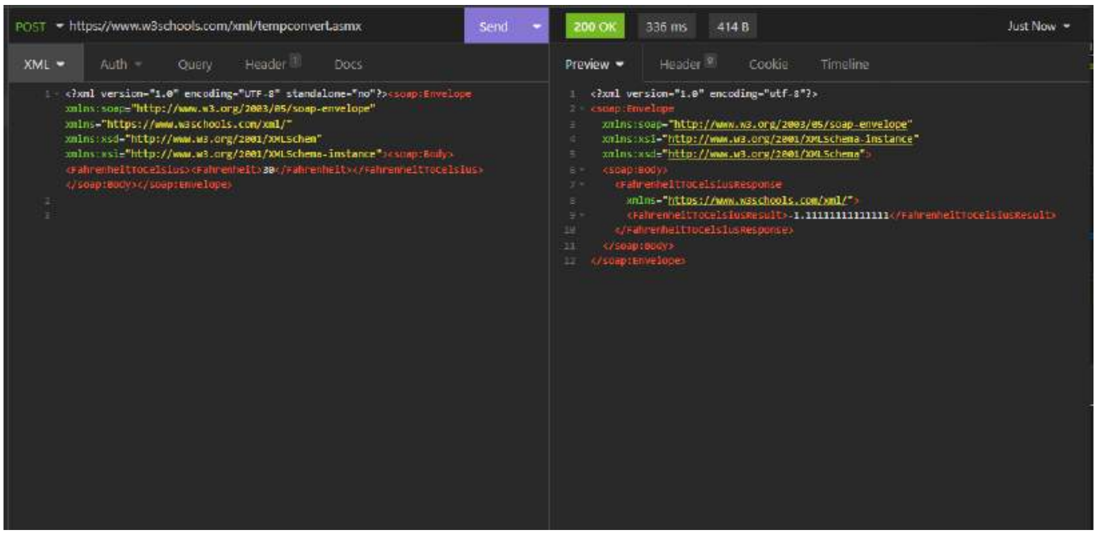


http://localhost:10000/middlewareconverter/24?Fahrenheit=30

```
{
  "FahrenheitToCelsiusResult": -1.11111111111111
}
```

FIGURE V.10 – Sending REST GET method.

As you can see the request is used as REST and the response is shown as REST too we will try to see the original service and see how requests are formed (See Figure V.11) as you can notice that it shows the same result but in SOAP format this is managed by REST to SOAP service transforms it from REST to SOAP and vice versa. This is proof that our method is efficient as was tested with several publicly available SOAP services mostly provided by W3C.

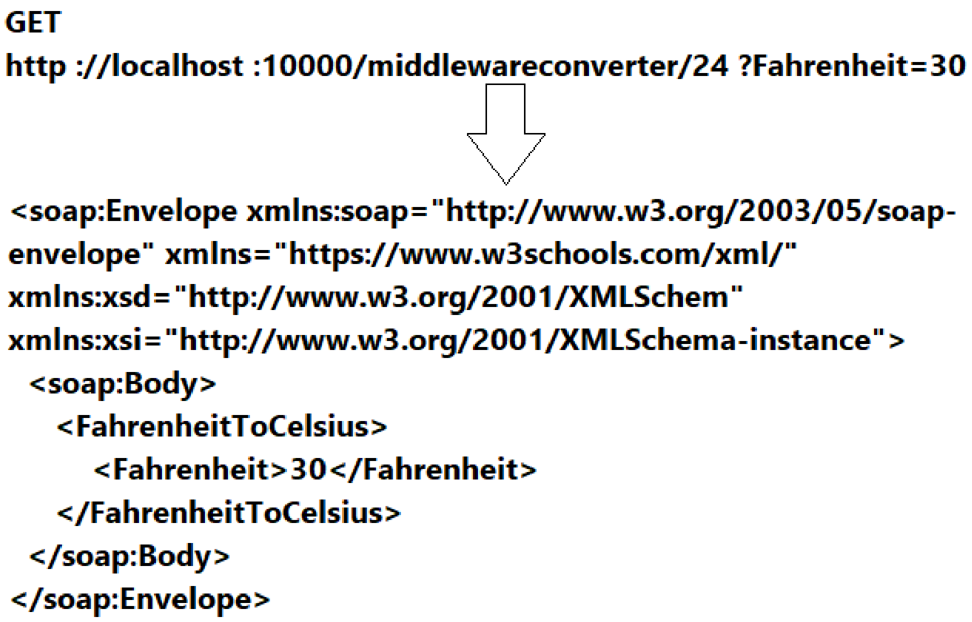


```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns="https://www.w3schools.com/xml/"
xmlns:xsd="http://www.w3.org/2001/XMLSchem"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><soap:Body>
<FahrenheitToCelsius><Fahrenheit>30</Fahrenheit></FahrenheitToCelsius>
</soap:Body></soap:Envelope>
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <FahrenheitToCelsiusResponse
      xmlns="https://www.w3schools.com/xml/">
      <FahrenheitToCelsiusResult>-1.11111111111111</FahrenheitToCelsiusResult>
    </FahrenheitToCelsiusResponse>
  </soap:Body>
</soap:Envelope>
```

FIGURE V.11 – Sending SOP request.

The transformation was in two directions from REST to SOAP as shown in the Figure V.12.

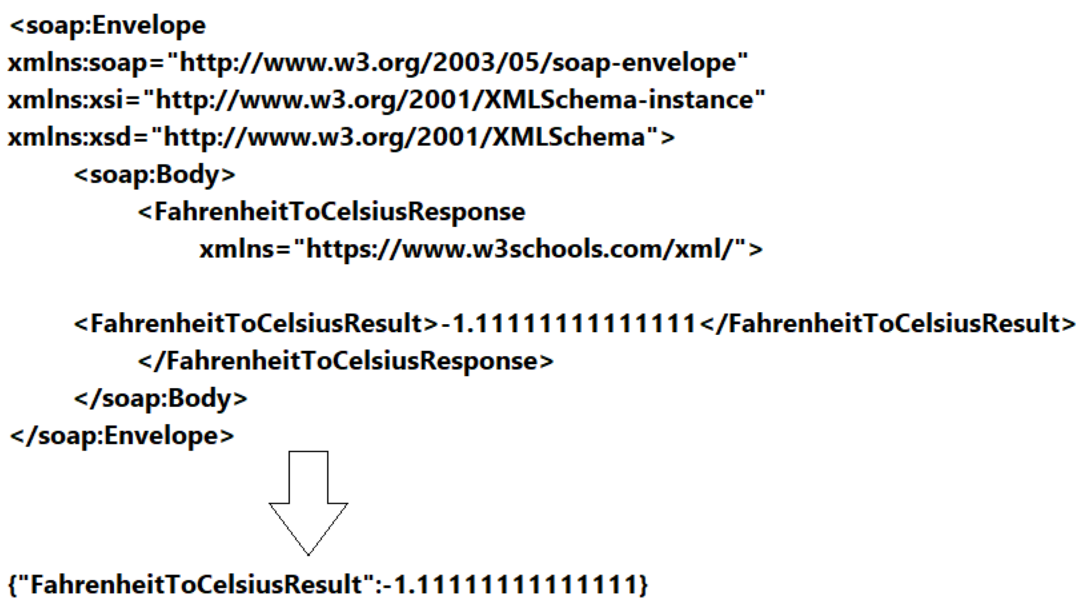


```
GET
http ://localhost :10000/middlewareconverter/24 ?Fahrenheit=30
```

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-
envelope" xmlns="https://www.w3schools.com/xml/"
xmlns:xsd="http://www.w3.org/2001/XMLSchem"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Body>
    <FahrenheitToCelsius>
      <Fahrenheit>30</Fahrenheit>
    </FahrenheitToCelsius>
  </soap:Body>
</soap:Envelope>
```

FIGURE V.12 – Going from REST to SOAP.

And in the Figure V.13 we can see the back transformation from SOAP to REST format.



```
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <soap:Body>
            <FahrenheitToCelsiusResponse
                  xmlns="https://www.w3schools.com/xml/">

      <FahrenheitToCelsiusResult>-1.11111111111111</FahrenheitToCelsiusResult>
            </FahrenheitToCelsiusResponse>
      </soap:Body>
</soap:Envelope>
```

```
{"FahrenheitToCelsiusResult":-1.11111111111111}
```

FIGURE V.13 – Going from SOAP to REST.

V.5 Comparing our work to the related works :

In this section, we will compare our solution to the solutions of the third chapter according to the criteria mentioned in it.

Criteria *Solution*	C1	C2	C3	C4	C5	C6
Our solution	Microservice	WADL	REST	SaaS	Provider	Automatic

TABLE V.1 – Evaluation of our solution.

V.6 Conclusion :

In this chapter, we show how we implemented our interoperability solution and brought it to life.
We started by showing our hardware and software setup then we described our architecture after that we showed our virtual environment used in this project finally we displayed screenshots from our system and conclude with a small conclusion.

Conclusion and perspectives

The main purpose of this work is to propose a solution to API unification in the cloud environments, between different APIs SOAP and REST. We started by providing a general introduction to the cloud and its architecture along with the characteristics and deployment model of the cloud and we concluded it by briefly introducing the multi-cloud and its characteristics, then in the second chapter we presented some basic concepts of the cloud and it's the environment like virtualization and API format, the application architecture used in the cloud services the described type of communication is used in it, where in the third chapter we mentioned related works to our topic, next we proposed our solution and presented its implementation in chapter five.

Our proposed solution consists of two topics portability and interoperability which allows for SOAP and REST API to communicate without any hardcoding or modification in their code and the second part allows for semantic matching between REST APIs inputs and outputs and by integrating these two solutions even SOAP API can use the portability with REST APIs.

This experience has pushed us to go deep into cloud computing and it's a complex concept, we learned a lot of information and applied them from microservices architecture to data format in the cloud .and opening a lot of prospects to discover more.

This work has many perspective and development work starting from adding more APIs format like GraphQL and XML-RPC protocol when it comes to portability we suggest adding complex matching methods to raise the matching precision, finally, we hope that our solution help solves these challenges and overcome provider lock.

Bibliography

[1] what is microservices ? https://aws.amazon.com/microservices/. Accessed : 2022-06-25.

[2] what is service-oriented architecture ? https://www.redhat.com/en/topics/cloud-native-apps/what-is-service-oriented-architecture. Accessed : 2022-06-20.

[3] (2007). The official ccif management team members. https://groups.google.com/g/cloudforum/c/uZFkue9qj4Q. Accessed : 2022-08-01.

[4] (2007). Web Services Description Language (WSDL) Version 2.0 Part 1. https://www.w3.org/TR/2007/REC-wsdl20-20070626/. Accessed : 2022-09-01.

[5] (2018a). Cloud Computing Portability and Interoperability. https://www.dreamfactory.com/. Accessed : 2022-08-03.

[6] (2018). DreamFactory. https://www.dreamfactory.com/. Accessed : 2022-07-31.

[7] (2018b). Knowledge Byte : The Benefits Of Cloud Interoperability And Portability. https://www.cloudcredential.org/blog/knowledge-byte-the-benefits-of-interoperability-and-portability. Accessed : 2022-07-30.

[8] (2022). AMQP 0-9-1 Model Explained. https://www.rabbitmq.com/tutorials/amqp-concepts.html. Accessed : 2022-06-28.

[9] (2022). Kafka : Advantages and Disadvantages. https://www.https://www.javatpoint.com/apache-kafka-advantages-and-disadvantages?fbclid=IwAR0dGcU2M95OzbD0Nr9zvBoPGtrQf--bhZirV5vyTfk0uksJJ-XCo7hV7B8. Accessed : 2022-06-24.

[10] (2022). OpenAPI. https://www.openapis.org/. Accessed : 2022-08-09.

[11] (2022). Soap web services. https://www.javatpoint.com/soap-web-services. Accessed : 2022-06-12.

[12] (2022). Synchronous vs. asynchronous communication (in the cloud and beyond. https://www.https://www.predicagroup.com/blog/synchronous-vs-asynchronous-communication/. Accessed : 2022-06-29.

[13] (2022). What can RabbitMQ do for you ? https://www.rabbitmq.com/features.html. Accessed : 2022-06-28.

[14] (2022). What is a rest api ? https://www.redhat.com/en/topics/api/what-is-a-rest-api. Accessed : 2022-06-26.

[15] (2022). What Is Cloud Computing Architecture : Front-End Back-End Explained. https://www.https://www.spiceworks.com/tech/cloud/articles/what-is-cloud-computing-architecture-front-end-back-end-explained/?fbclid=IwAR2DPgR6rfVzrWXGyFzOJWPQXtDcUy6bo11UI3g3e_KpFl_31Tl0ZAkBIZI/. Accessed : 2022-06-02.

[16] (2022). what-is-graphql ? https://www.redhat.com/en/topics/api/what-is-graphql. Accessed : 2022-06-26.

[17] Afsari, K., Eastman, C., and Shelden, D. (2017). Building information modeling data interoperability for cloud-based collaboration : Limitations and opportunities. *International Journal of Architectural Computing*, 15(3) :187–202.

[18] Aksakalli, I. K., Çelik, T., Can, A. B., and Tekinerdoğan, B. (2021). Deployment and communication patterns in microservice architectures : A systematic literature review. *Journal of Systems and Software*, 180 :111014.

[19] Ameen, R. Y. and Hamo, A. Y. (2013). Survey of server virtualization. *arXiv preprint arXiv :1304.3557*.

[20] Apostu, A., Puican, F., Ularu, G., Suciu, G., Todoran, G., et al. (2013). Study on advantages and disadvantages of cloud computing–the advantages of telemetry applications in the cloud. *Recent advances in applied computer science and digital services*, 2103.

[21] Aradhana, C., Balaji, R., Jim, P., Joe, W., Michele, D., and Tushar, B. (2011). Interoperability and portability. *Cloud Security Alliance Group*, 4.

[22] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., et al. (2009). Above the clouds : A berkeley view of cloud computing. Technical report, Technical Report UCB/EECS-2009-28, EECS Department, University of California ….

[23] Ayanoglu, E., Aytas, Y., and Nahum, D. (2016). *Mastering rabbitmq*. Packt Publishing Ltd.

[24] Baudoin, C., Dekel, E., and Edwards, M. (2014). Interoperability and portability for cloud computing : a guide. *Cloud Standards Customer Council*, 1(1) :1–31.

[25] Bouzerzour, N. E. H. and Slimani, Y. (2022). Towards a maas service for cloud service interoperability. In *MODELSWARD*, pages 72–83.

[26] Caragea, D. and Syeda-Mahmood, T. (2004). Semantic api matching for automatic service composition. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 436–437.

[27] Cretella, G. and Di Martino, B. (2013). Semantic and matchmaking technologies for discovering, mapping and aligning cloud providers's services. In *Proceedings of International Conference on Information Integration and Web-based Applications & Services*, pages 380–384.

[28] Daniels, J. (2009). Server virtualization architecture and implementation. *XRDS : Crossroads, The ACM Magazine for Students*, 16(1) :8–12.

[29] Davies, A. (2004). Computational intermediation and the evolution of computation as a commodity. *Applied Economics*, 36(11) :1131–1142.

[30] De, B. (2017). Introduction to apis. In *API Management*, pages 1–14. Springer.

[31] DMTF, O. (2009). Virtualization format specification version 1.0. 0.

[32] Dobbelaere, P. and Esmaili, K. S. (2017). Kafka versus rabbitmq : A comparative study of two industry reference publish/subscribe implementations : Industry paper. In *Proceedings of the 11th ACM international conference on distributed and event-based systems*, pages 227–238.

[33] Donepudi, P. K. (2016). Influence of cloud computing in business : are they robust. *Asian journal of applied science and engineering*, 5(3) :193–196.

[34] Dong, H., Hao, Q., Zhang, T., and Zhang, B. (2010). Formal discussion on relationship between virtualization and cloud computing. In *2010 International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 448–453. IEEE.

[35] Dos Santos, C. T., Quaresma, P., and Vieira, R. (2010). An api for multilingual ontology matching. In *Proc. 7th conference on Language Resources and Evaluation Conference (LREC)*, pages 3830–3835. No commercial editor.

[36] Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. University of California, Irvine.

[37] Fu, G., Zhang, Y., and Yu, G. (2020). A fair comparison of message queuing systems. *IEEE Access*, 9 :421–432.

[38] Gaughan, D. and Finley, I. (2006). Soa pace accelerating into 2007. *AMR Research, November*, 30.

[39] Group, T. O. (2017). Cloud Computing Portability and Interoperability. `http://www.opengroup.org/cloud/cloud_iop/p4.htm`. Accessed : 2022-07-30.

[40] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H. F., Karmarkar, A., and Lafon, Y. (2003). Soap version 1.2.

[41] Hiraman, B. R. et al. (2018). A study of apache kafka in big data stream processing. In *2018 International Conference on Information, Communication, Engineering and Technology (ICICET)*, pages 1–3. IEEE.

[42] Hong, J., Dreibholz, T., Schenkel, J. A., and Hu, J. A. (2019). An overview of multi-cloud computing. In *Workshops of the international conference on advanced information networking and applications*, pages 1055–1068. Springer.

[43] Ismail, M. (2022). Evaluation of generic graphql servers for accessing legacy databases.

[44] Jain, N. and Choudhary, S. (2016). Overview of virtualization in cloud computing. In *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, pages 1–4. IEEE.

[45] Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., and Tilkov, S. (2018). Microservices : The journey so far and challenges ahead. *IEEE Software*, 35(3) :24–35.

[46] John, V. and Liu, X. (2017). A survey of distributed message broker queues. *arXiv preprint arXiv :1704.00411*.

[47] Kazanavičius, J. and Mažeika, D. (2019). Migrating legacy software to microservices architecture. In *2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*, pages 1–5. IEEE.

[48] Kennedy, S., Stewart, R., Jacob, P., and Molloy, O. (2011). Storhm : a protocol adapter for mapping soap based web services to restful http format. *Electronic commerce research*, 11(3) :245–269.

[49] Khan, M. W. and Abbasi, E. (2015). Differentiating parameters for selecting simple object access protocol (soap) vs. representational state transfer (rest) based architecture. *Journal of Advances in Computer Networks*, 3(1) :63–6.

[50] Lamouchi, N. (2021). Introduction to the monolithic architecture. In *Pro Java Microservices with Quarkus and Kubernetes*, pages 35–38. Springer.

[51] Le Noac'H, P., Costan, A., and Bougé, L. (2017). A performance evaluation of apache kafka in support of big data streaming applications. In *2017 IEEE international conference on big data (Big Data)*, pages 4803–4806. IEEE.

[52] Lee, Y.-J. and Kim, J.-S. (2012). Automatic web api composition for semantic data mashups. In *2012 Fourth International Conference on Computational Intelligence and Communication Networks*, pages 953–957. IEEE.

[53] Lim, F. P. (2017). An analysis of synchronous and asynchronous communication tools in e-learning. *Advanced Science and Technology Letters*, 143(46) :230–234.

[54] Lin, D. et al. (1998). An information-theoretic definition of similarity. In *Icml*, volume 98, pages 296–304.

[55] Liu, G., Huang, B., Liang, Z., Qin, M., Zhou, H., and Li, Z. (2020). Microservices : architecture, container, and challenges. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 629–635. IEEE.

[56] MacKenzie, C. M., Laskey, K., McCabe, F., Brown, P. F., Metz, R., and Hamilton, B. A. (2006). Reference model for service oriented architecture 1.0. *OASIS standard*, 12(S 18).

[57] Majda, E. and Ahmed, E. (2015). Using cloud saas to ensure interoperability and standardization in heterogeneous cloud based environment. In *2015 5th World Congress on Information and Communication Technologies (WICT)*, pages 29–34. IEEE.

[58] Matthews, J. N., Dow, E. M., Deshane, T., Hu, W., Bongio, J., Wilbur, P. F., and Johnson, B. (2008). *Running Xen : a hands-on guide to the art of virtualization*. Prentice Hall PTR.

[59] Max Knutsson, T. G. (2015). Challenges of service-oriented architecture (soa) - from the public sector perspective.

[60] Mayer, B. and Weinreich, R. (2018). An approach to extract the architecture of microservice-based software systems. In *2018 IEEE symposium on service-oriented system engineering (SOSE)*, pages 21–30. IEEE.

[61] Mell, P., Grance, T., et al. (2011). The nist definition of cloud computing.

[62] Meng, J., Mei, S., and Yan, Z. (2009). Restful web services : A solution for distributed data integration. In *2009 International Conference on Computational Intelligence and Software Engineering*, pages 1–4. IEEE.

[63] Meng, L., Huang, R., and Gu, J. (2013). A review of semantic similarity measures in wordnet. *International Journal of Hybrid Information Technology*, 6(1) :1–12.

[64] Munaf, R. M., Ahmed, J., Khakwani, F., and Rana, T. (2019). Microservices architecture : Challenges and proposed conceptual design. In *2019 International Conference on Communication Technologies (ComTech)*, pages 82–87. IEEE.

[65] Newman, S. (2019). *Monolith to microservices : evolutionary patterns to transform your monolith*. O'Reilly Media.

[66] Ofoeda, J., Boateng, R., and Effah, J. (2019). Application programming interface (api) research : A review of the past to inform the future. *International Journal of Enterprise Information Systems (IJEIS)*, 15(3) :76–95.

[67] Ogu, E. C., Ayokunle, O., Yaw, M., and Achimba, O. (2014). Virtualization and cloud computing : The pathway to business performance enhancement, sustainability and productivity. *International Journal of Business and Economics Research*, 3(5) :170–177.

[68] Oludele, A., Ogu, E. C., Shade, K., and Chinecherem, U. (2014). On the evolution of virtualization and cloud computing : A review. *Journal of Computer Sciences and Applications*, 2(3) :40–43.

[69] Pahl, C. and Jamshidi, P. (2016). Microservices : A systematic mapping study. *CLOSER (1)*, pages 137–146.

[70] Petcu, D. and Vasilakos, A. V. (2014). Portability in clouds : approaches and research opportunities. *Scalable Computing : Practice and Experience*, 15(3) :251–270.

[71] Potdar, A. M., Narayan, D., Kengond, S., and Mulla, M. M. (2020). Performance evaluation of docker container and virtual machine. *Procedia Computer Science*, 171 :1419–1428.

[72] Rabiee, A. (2018). Analyzing parameter sets for apache kafka and rabbitmq on a cloud platform.

[73] Raj, V. and Sadam, R. (2021). Performance and complexity comparison of service oriented architecture and microservices architecture. *International Journal of Communication Networks and Distributed Systems*, 27(1) :100–117.

[74] Ranabahu, A. and Sheth, A. (2010). Semantics centric solutions for application and data portability in cloud computing. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 234–241. IEEE.

[75] redhat (2018). What is multicloud? `https://www.redhat.com/en/topics/cloud-computing/what-is-multicloud`. Accessed : 2022-06-21.

[76] Sayago Heredia, J., Flores-García, E., and Solano, A. R. (2019). Comparative analysis between standards oriented to web services : Soap, rest and graphql. In *International conference on applied technologies*, pages 286–300. Springer.

[77] Serrano, D. and Stroulia, E. (2020). Semantics-based api discovery, matching and composition with linked metadata. *Service Oriented Computing and Applications*, 14(4) :283–296.

[78] Sharvari, T. and Sowmya Nag, K. (2019). A study on modern messaging systems-kafka, rabbitmq and nats streaming. *CoRR abs/1912.03715.*

[79] Shetty, S., Manu, A., Kumar, V., and Antony, C. (2020). Need of multi-cloud environment and related issues : A survey. *Journal of Xi an University of Architecture & Technology*, 12 :78–87.

[80] Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A., and Peterson, L. (2007). Container-based operating system virtualization : a scalable, high-performance alternative to hypervisors. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys european conference on computer systems 2007*, pages 275–287.

[81] Syeda-Mahmood, T., Shah, G., Akkiraju, R., Ivan, A.-A., and Goodwin, R. (2005). Searching service repositories by combining semantic and ontological matching. In *IEEE International Conference on Web Services (ICWS'05)*, pages 13–20. IEEE.

[82] Szyperski, C., Gruntz, D., and Murer, S. (2002). *Component software : beyond object-oriented programming*. Pearson Education.

[83] Upadhyaya, B., Zou, Y., Xiao, H., Ng, J., and Lau, A. (2011). Migration of soap-based services to restful services. In *2011 13th IEEE International Symposium on Web Systems Evolution (WSE)*, pages 105–114. IEEE.

[84] Wang, X. V., Wang, L., and Gördes, R. (2018). Interoperability in cloud manufacturing : a case study on private cloud structure for smes. *International Journal of Computer Integrated Manufacturing*, 31(7) :653–663.

[85] Wegner, P. (1996). Interoperability. *ACM Computing Surveys (CSUR)*, 28(1) :285–287.

[86] Winters, L. S., Gorman, M. M., and Tolk, A. (2006). Next generation data interoperability : It's all about the metadata. In *IEEE Fall Simulation Interoperability Workshop*. Citeseer.

[87] Yangui, S., Mohamed, M., Sellami, M., and Tata, S. (2013). Open cloud computing interface-platform. In *Open Grid Forum, Tech. Rep*. Citeseer.

[88] Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing : state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1) :7–18.

[89] Zur Muehlen, M., Nickerson, J. V., and Swenson, K. D. (2005). Developing web services choreography standards—the case of rest vs. soap. *Decision support systems*, 40(1) :9–29.