POLITECNICO DI TORINO

Master’s degree course
in Data Science

College of Computer Engineering, Mechatronics, and of Cinema

Master's degree thesis

AWS Cloud: Infrastructure, DevOps techniques, State of Art.

Supervisors:
Prof. Ing. Luciano LAVAGNO

Co-supervisors:
Stefano AMEDEO (company tutor)

Candidate:
Simone BOSCAIN

Academic Year 2022/2023

Summary

Image Index

Acronym

AWS: Amazon Web Services

IaaS: Infrastructure as a Service

PaaS: Platform as a Service

SaaS: Software as a Service

FaaS: Function as a Service

AZ: Availability Zones

EC2: Elastic Compute Cloud

S3: Simple Storage Service

RDS: Relational Database Service

VPC: Virtual Private Cloud

VPN: Virtual Private Network

DNS: Domain Name System

IAM: Identity and Access Management

SNS: Simple notification service

API: Application Programming Interface

HTTP: Hypertext Transfer Protocol

REST: Representational State Transfer

DevOps: Development Operations

ECR: Elastic Container Registry

EKS: Elastic Kubernetes Service

JSON: JavaScript Object Notation

YAML: Ain’t Markup Language

TBD: Trunk-based development

Abstract

This thesis consists in an analysis of a project developed in collaboration with ***Akka Technologies Company***. The implemented software is a backend service built with the support of Amazon Web Services technology, which is the leading cloud provider service. The purpose of this work is to review the ***best strategies*** and ***technologies*** used during the development of the project. However, the thesis can't enter either the specifics code details or its future applications due to the customer required secrecy. During the implementation of the project, the first decision taken is to employ the cloud to host the architecture in order to lower the cost, improve the scalability and reliability. Therefore, the second decision is related to the choice of the best cloud provider. ***Amazon Web Services*** (AWS) is a slightly better option than the competitors. The development paradigm used is called ***serverless***, that means the developer does not need to build his own server infrastructure but he can exploit the one furnished by the cloud provider. This is a big advantage in terms of deployment rapidity, cost and easiness of making the software scale because it does not require further investments in hardware or time. In fact, to change the service options, only few actions are needed. AWS offers a long lists of services that can be used to realize cloud projects. After the analysis of the most important ones, the basic services for realizing the project are EKS and Lambda. ***Lambda*** is the best choice because its pricing is based on usage and its running time must not overcome 15 minutes which is more than enough to run an API REST that save and retrieve data on the database. The others main architectural components used are the ***API Gateway*** that exposes the endpoints necessary to reach the lambda functions, the ***Secret Manager*** which keep stored environmental variables, ***Cognito*** that is the authentication service and **RDS** which is the database one. The process of development is realized using the best practices proposed from ***DevOps***. Therefore the app is a ***microservices*** one, in fact each lambda works independently from the others. The job is organized using ***Scrum*** to try

to keep the a good visibility of the tasks and following an idea of learning culture and improvement of the team members. The strategy of continuous delivery, testing, logging systems and cloud security are realized at state of art. Also the Infrastructure as Code paradigm is implemented with Terraform service that let you create a cloud infrastructure only using code.

Thesis goal

This thesis was carried out in collaboration with the company AKKA Technologies mainly operating in smart working under the guidance of the enterprise tutor Stefano Amedeo and with the help of supervisor Luciano Lavagno. The company, which has offices throughout Italy and Europe, is interested in various sectors of technology and deals both with the realization of projects and with providing assistance to other industries through its experts.

The main purpose of this thesis is the resolution of a problem raised by the company itself, which needs to develop software that is able to receive data from a Bluetooth device installed on a wearable accessory, visualize and manage all these data on an app and store them in cloud. If we want to analyze the project, we can say it needs a large gamma of competences to be built to its fullest. At the start the company offered me different options to work with:

- ✓ *The creation of the communication protocols between the app and the firmware;*
- ✓ *The creation of the app;*
- ✓ ***The creation of the backend service using cloud technologies;***
- ✓ *Machine Learning and/or Deep Learning on the data gathered by the app to improve the service or analyze some key information about the devices;*
- ✓ ***Keeping account of the optimal cloud architecture, develop following the best-practice DevOps principles** and the study of laws concerning data backup in various countries of world.*

All the options were very interesting but at the end i opted for the last three points excluding the laws study about the data in the world. Unluckily the development level of the project at the start was only at its initials stages, so only after some time we discovered that the time required to acquire app data to make a Machine Learning or Deep Learning job would be too long for my thesis deadline. So at the end the main focus of this thesis is around cloud technologies, their best architecture setup and all what concerns state of art practices of developing in a team following the DevOps principles.

Chapter 1

1 Cloud development history

Image 1:1: Data flows from the cloud directly into the devices we keep in our hands.

Initially, companies and businesses with the need to store data had their own servers located in their respective offices. Later, to achieve their protection and to make management easier, the servers were moved to places suitable for their storage called ***datacenter***. It was then decided to install only one application per server due to the difficulty of isolating the services provided by different software. Due to this rule, analyzing a data center, it was observed that many servers were inactive most of the time. To overcome the rule of one application per server, three problems had to be solved:

1. *Isolation of shared configurations: two different applications may require different versions of the same library to cite an example;*

2. *Performance isolation: if an application consumes a lot of CPU it could cause slowdowns in the use of the others;*

3. *Isolation for security and reliability: breaching an application could compromise the entire server and therefore also the security of the other applications it contains.*

Thanks to virtualization it has been possible to overcome these limits by allowing the simulation of many completely isolated servers on the same physical machine, also simplifying their management. The annexation of large quantities of servers in data centers combined with the ease of management provided by virtual machines has led to the birth of ***cloud computing***. The cloud is a technology that allows remote access to software and hardware resources, the use of which is offered as a service by a provider. So in the cloud the user does not buy the product, hardware or software, but pays to use it. Cloud computing is becoming more and more successful, and this has favored the born of various types of cloud services with different levels of control, adaptability, and management:

1. *Software as a Service (SaaS): The product that is furnished with it is fully administered by the service provider. In the case of SaaS, the user's only concern is learning how to use the software offered by the provider without worrying about the management of the underlying infrastructure or service.*

2. *Platform as a service (PaaS): The provider offers to the user a basic set of services that can be used to create new software. The user doesn't have to worry about hardware management but only about application development. Developers can then focus on deploying and managing applications and don’t have to manage infrastructure (hardware and operating systems). There may be some development limitations regarding the types of programming languages accepted by the platform.*

3. *Infrastructure as a service (IaaS): This type of service encompasses the basic elements of cloud-based IT and usually guarantees access to network functions, servers (virtual or on dedicated hardware) and data storage space. Through the* IaaS *you get the highest level of elasticity and monitoring of IT assets. So in this type of service the provider offers the infrastructure, while the user creates the virtual machines and loads the service he wants to provide onto them.*

4. ***Function as a service (FaaS)****: is a type of cloud computing service that allows developers to construct, compute, operate, and manage application packages as functions without having to maintain their own infrastructure.* FaaS *is an* **event-driven** *execution paradigm that operates in* ***stateless containers****, and those functions handle server-side logic and state by utilizing services from a* FaaS *provider.* FaaS *provides developers with an abstraction for executing web applications in response to events, while avoiding the need to manage servers.* FaaS *infrastructure is often metered on-demand by the service provider, largely via an event-driven execution approach, so it is available when needed, but does not necessitate any server processes running continuously in the background, as platform-as-a-service (*PaaS*) would.*

1.1 <u>Cloud advantages</u>

The reasons to use the cloud services are various:

1. *Variable costs based on consumption indeed of fixed investments. The Cloud generally has lower prices than the ones required by the management of a local server (thanks to its aggregate administration).*

2. *Resolves the need to guess the size of the service prior to its creation and allows you to access exactly the required quantity by scaling it up or down as needed.*

3. *The distribution of the service is fast and agile because you need only few clicks to send online a new service without being worried about the server management. You can rapidly distribute a service in different parts of the world in an easy and fast way reducing the latency problems.*

4. *Releases the developer from server management and server expenses allowing to shift the focus to user needs.*

The main reason that cloud computing has been so successful remains the ability to lower initial investments for IT infrastructure by replacing it with modest variable costs, which change according to the needs of the business. Thanks to the cloud, enterprises no longer have to plan the purchase of servers and other IT devices in advance. Now they can create all the assets they need in minutes, significantly reducing deployment time. The global Covid pandemic that has damaged many markets has instead shown, in the case of cloud computing, the importance of being able to quickly access IT resources with affordable prices. The cloud services are provided by various suppliers. The following data have been extracted from the **Ref "Statista website"**. The AWS supplier in 2022 has a 32% market share in the field of cloud computing, confirming itself as the leader in the sector. Microsoft Azure is still the second giant on the market with a share of 23%. Far behind the other big players in the sector are Alibaba and GPC with about 9% each. We will analyze below the two main competitors in the cloud computing sector.

1.2 <u>Amazon Web Services</u>

Image 1.2: AWS Cloud logo.

*Amazon Web Services (*AWS*):* Amazon provides its cloud-based services through this platform. Amazon is the current global market leader in the sector and continues to invest in the improvement of the services offered by AWS. One of the main benefits of the platform is the customer's friendly pricing based on the amount of usage of the AWS services. AWS cloud is also the supplier that has servers in the most high number of various geographical areas of the world and this allows to decrease the latency of its services. The availability of the service is also another positive point for AWS and, in fact, it is around 100%. This is the reason many companies including Spotify, Netflix and Airbnb host their data on AWS.

1.3 Microsoft Azure

Image 1.3: Microsoft Azure Cloud Logo.

Microsoft Azure: formerly known as Windows Azure, it’s Microsoft's cloud computing platform. It offers several cloud services, including computing, analytic, storage, and networking. Organizations can leverage these services to build new applications or launch existing software in the cloud. Azure seeks to support all industries and is compatible with open source technologies. With Azure you can start servers, balance them, scale them, launch serverless functions, generate virtual networks and store large amounts of data. The main disadvantage of Azure is the cost of its services which are relatively higher than the competitors. However, it is currently the main competitor of AWS and has an almost equally efficient geographical coverage with more than 40 datacenters scattered all over the world.

1.4 AWS vs Azure

If we want to compare these two service providers, the first point to analyze concerns the virtual machines (VMs) which constitute the backbone of the cloud environment for the virtualization of IT systems. We can say that in this case the technologies put in place are different, but similar in terms of capabilities and functionality, even if AWS seems to have slightly better performance. As for the network difference characteristics, both AWS and Azure provides very similar services but AWS, having more datacenters, is able to lower the latency of the service a little more than Azure (If we consider an average of the latency of the two service providers). AWS and Azure also have parallel and cutting-edge characteristics in data storage and cybersecurity. In terms of costs, AWS seems to be a little cheaper than Azure. Even if the differences are not so marked and both rivals are valid choices, it was decided in agreement with the company to deepen the study of AWS which seems to be a generically better option than the competitor.

Chapter 2

2 Amazon Web Services

Image 2.1: AWS is like a toolbox for Cloud.

Starting in 2006, Amazon Web Services (AWS) began providing businesses with cloud computing web services through its IT infrastructure. AWS despite already being an industry sector leader continues to extend the global infrastructure to provide its users with ***low latency*** and ***high throughput*** services and allow data storage in the desired regions. The AWS infrastructure is based on ***Regions*** and ***"Availability Zones" (AZ)***. In a region there are many AZs in which there are data centers in variable numbers, each arranged in a different structure. The goal of this widespread arrangement is to obtain ***high performance***, ***scalability*** and ***resistance to failures***. Each AZ is located in areas with low risk of environmental catastrophes and is powered by networks that are different from the others in order to limit the possibility of the collapse of several AZs at the same time. Thanks to these characteristics, AWS has an extremely stable, adaptable and economical cloud infrastructure chosen by more than one hundred thousand companies worldwide. AWS provides all different types of services: Infrastructure as a Service, Platform as a Service, Software as a Service, Function as a Service. One of the main reasons because AWS is chosen is the pricing which is very clear.

Each AWS instance has a cost based on its usage time or its disk usage amount in case of a data storage service. There is also a payment window which shows the customer's costs for each individual service. The registration process is also designed to be very accessible and does not require signing any agreements. It is sufficient to enter the email and credit card and you can work immediately.

2.1 AWS list of services

AWS offers a multitude of services designed for every customer need:

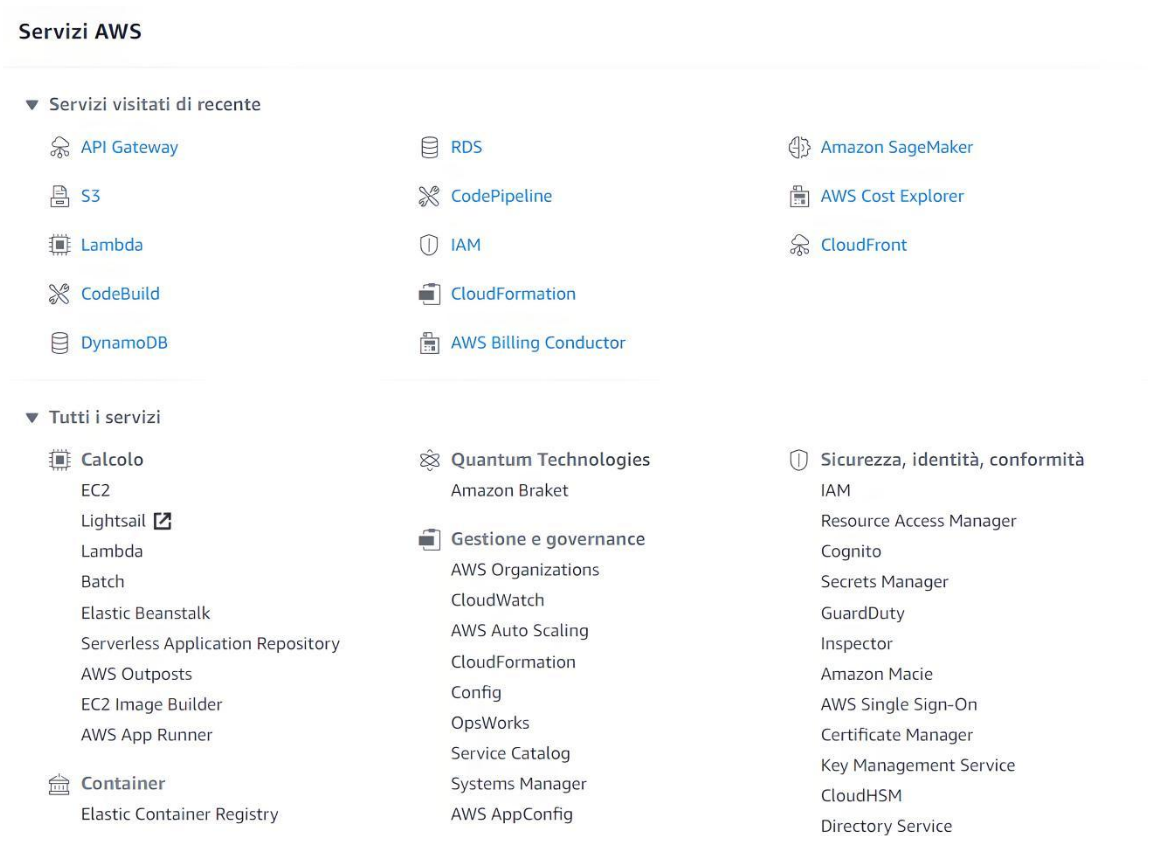


Image 2.2: AWS has a wide list of services (in this image we see only a part of them).

The amount of services provided by AWS is so large and out of our scope that we will only analyze the ones seen during the development of our project:

- ✓ *Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure and adaptable computing power within the* AWS *Cloud. It is designed to make large-scale computing resources more accessible to users. Thanks to the EC2 graphical interface, we can easily select the required configuration of the virtual machine we will run into the Amazon environment. You can choose from many instances with different operating systems and software packages, as well as select the best amount of memory, storage space, CPU for your application. Through Amazon EC2 we are able to change the amount of resources available quickly and easily allowing our projects to scale without difficulty, and it is also possible to have the service automatically scale as needed thanks to auto-scaling which can change the number of available EC2 instances based on the load. Through Elastic Load Balancing you can automatically distribute traffic among multiple EC2 instances, and this provides higher resistance to errors and allows you to distribute traffic evenly. EC2 can integrate with most other* AWS *services and is very reliable, in fact* AWS *guarantees higher than 99% availability in each region. In addition to being reliable, the EC2 service is also secure and works in conjunction with Amazon VPC to provide robust network features for users. It is also possible to choose between different payment plans which are more or less expensive depending on the performance and availability required by the user.*

- ✓ ***AWS Lambda** is a service that allows you to launch your own programs without having to provision or manage servers. Payments are structured so that charges reflect usage time, and non-executed code is free. Thanks to Lambda, you just need to upload the code, and the service itself will take care of the execution with high availability. You can have Lambda functions* *called from any app or from other AWS services thanks to the trigger functionality.*

- ✓ *Amazon Simple Storage Service (Amazon S3) allows you to store and retrieve data with a simple and intuitive service thanks to the graphical interface. The availability and durability of the data contained in the service is almost 100%. In fact, the data is saved so that there are multiple copies of it in different availability zones and in multiple devices within the different structures. S3 has different offers depending on how often the saved data is retrieved, and allows us to scale the data storage space easily, and thanks to this we pay only for the space we need without having to allocate resources for future needs. S3 is secure in fact the data transfer takes place using SSL and subsequently the archived data is automatically encrypted, moreover you can decide which IAM users can access the data.*

- ✓ ***Amazon Relational Database Service (Amazon RDS)** permits the developer to use relational databases within the cloud by choosing from Oracle, Microsoft SQL Server, MariaDB, MySQL, Aurora and PostgreSQL. The service is easily configurable and scalable at an economical price. With its simplicity of management, Amazon RDS allows you to get to distribution faster. As with the other AWS services, RDS is also secure in fact the database always has at least one replica in another Availability Zone (AZ) and you can configure access using a VPC. If this is not enough all data are encrypted. RDS prices are low and calculated according to usage.*

- ✓ *Amazon Virtual Private Cloud (Amazon VPC) allows you to logically isolate a cloud network from which you can launch certain AWS resources. The user can completely manage the virtual network thus created, choosing which is the gateway, designing the routing tables and any subnet, and choosing a* *range of IP addresses to assign and whether to use IPv4 or IPv6. Customizing the network configuration of VPC is very simple. You can also generate hardware Virtual Private Network (VPN) connections by merging the corporate datacenter with the VPC then using AWS as an extension of the corporate network.*

- ✓ *Amazon Route 53 is a highly available and scalable Domain Name System (DNS) web service. It is designed to give users the ability to route clients to applications by switching literal addresses to IP addresses (and is also IPv6 compatible). Amazon Route 53 allows you to easily communicate user requests to infrastructure inside or outside AWS and to set controls on DNS traffic and on the endpoints of your applications. Amazon Route 53 allows you to choose from various routing types to lower latency and decrease errors. Amazon Route 53 also allows you to purchase domain names, after which Route 53 sets up the DNS configurations of the registered domain.*

- ✓ *AWS CodeCommit allows you to manage enterprise hosting of Git repositories in a scalable, private and secure way. With AWS CodeCommit you can archive both source, code and binaries, and it works perfectly with existing Git tools.*

- ✓ *AWS CodeBuild is a source code building, testing, and distribution-ready software package service. Through CodeBuild we can speed up code compilation times as the service continuously recalibrates resources and processes and reworks different versions simultaneously.*

- ✓ *AWS CodeDeploy is a service that automatically deploys code to instances, both for EC2 instances and locally running instances. AWS CodeDeploy accelerates the deployment of new features and performs necessary application update tasks. AWS*

CodeDeploy automates software deployment, which helps avoid the errors that manual operations are prone to.

- ✓ *AWS CodePipeline is a continuous integration and continuous delivery service that is used to update software and infrastructure in an automated way. Every time the code is edited CodePipeline builds, tests and deploys it following the path created by the developer. This makes development processes faster.*

- ✓ *Amazon CloudWatch is a monitoring service for AWS cloud resources and software running on AWS. Amazon CloudWatch is used for the purpose of collecting and managing log files and for setting alarms and responding automatically to changes in AWS resources. The instances that CloudWatch can monitor are RDS, DynamoDB, EC2, Lambda, and log files created by applications. It can also be used to analyze application performance. With the data collected in this way you can improve your own software.*

- ✓ *AWS CloudFormation provides developers with a quick service to generate an AWS architecture, that can be deployed rapidly and repeatedly. To program AWS resources and their parameters we can use the example models provided by AWS or generate models from scratch. After the AWS resources are generated, you can edit them normally as with those created manually. CloudFormation also provides a graphical interface through which we can immediately view our architecture and possibly add new elements that will then be editable in the code editor in case we need to add new parameters or dependencies.*

- ✓ ***AWS Identity and Access Management (IAM)*** *allows you to securely manage access to AWS resources and services for your users. Through IAM, you can administer user and user group permissions by providing or denying access to AWS resources.*

The entities that IAM works with are users and roles. Users have credentials for accessing AWS services while roles allow AWS resources to interact with other services.

- ✓ ***Amazon Cognito*** *is the service that allows users of our applications to carry out the process of registering, logging and accessing the application depending on the permissions they have. Authentication can be set up to use social providers like Facebook or Amazon. Cognito also allows you to keep the data locally in order to be able to start the application even offline. The synchronization function between multiple devices of the same user is also included. With the services offered by Amazon Cognito, you can devote yourself to the development of your application, leaving synchronization, registration and access control in the hands of the Amazon service.*

- ✓ ***Amazon API Gateway*** *allows you to create API that can be easily monitored, secured, modified, and published. With a few simple steps you can generate an API that acts as an entrance to our applications, data access and AWS services.*

- ✓ *Amazon Simple Notification Service (Amazon SNS) is capable of sending push notifications in the form of single or multiple messages (even SMS) to a large number of devices. It is fast and adaptable and can send to any device. In addition to other SNS devices, it can also send to various Amazon services and HTTP endpoints.*

2.2 Serverless architecture

Serverless computing is a cloud development model in which the developer can build and launch applications without the need to manage servers. The literal meaning of the word serverless is in fact "without server" because it exempts the developer from coming into contact with the latter during the performance of his duties. The servers in a serverless architecture are in fact managed by cloud service providers who take care of all scalability, management and procurement operations. Hereafter we will see the advantages and disadvantages of the serverless architecture:

1. *The developer's work is accelerated because he* ***no longer has to deal with the server infrastructure****. Since resources are allocated dynamically, it is no longer necessary to invest in advance on any additional resources for future use, nor to organize plans and costs regarding scalability. Serverless is inherently a model that aims to free the programmer from all aspects other than writing code.*

2. *Development price and infrastructure* ***costs are low*** *compared to other cloud development paradigms. In fact, the developer does not have to deal with the infrastructure and the time needed to create the app is reduced, consequently the production costs are lowered. Furthermore, thanks to the fact that the serverless architecture dynamically recalls the necessary resources, they will be paid only during use. This differs from other cloud architectures in which resources are allocated statically, and further reduces costs and simplifies the management of work peaks for which we will not be forced to allocate other fixed resources. After the distribution of the application, the developers will only have to correct any bugs in the application and no personnel will be needed to control the servers, further reducing costs.*

3. ***Resource optimization and auto-scaling*** *are another huge advantage of serverless architecture. In fact, processing and storage resources are dynamically assigned at the time of need*

and this makes it possible to easily adapt a service based on the user base.

4. ***Eventual latency problems** are instead a disadvantage that can occur if the stretch of code to be executed is requested infrequently, this because it is necessary to allocate new resource each time a request is made.*
5. *The main problem of serverless architecture is **being able to integrate the software into cloud structures**. Because of this, if you decide to change supplier it will probably be necessary to redevelop everything adapting it to its platform and its limits.*

2.3 Monolithic architecture and microservices architecture

The architectural structure of modern applications is divided into two large families, monolithic and microservices. The monolithic structure implements the application through a single component in which the entire software code is written. The microservices architecture, on the other hand, organizes the application as a set of multiple components, each of them is called a service, capable of speaking with each other through communication protocols. The monolithic structure was the first to be conceived and is still used **today** if the applications to be developed are of modest complexity. Its advantages are:

- ✓ *Ease of development, in fact the development environment and programmers are focused on creating a single application.*
- ✓ *Simplicity of modification, any element of the application can be altered without having to deal with other development teams because it is not necessary to put the application in contact with*

other components that could cause interaction problem in case of latter modifications.

- ✓ *Simple to test, you don't need to make the application communicate with other components but simply launch and test it individually.*
- ✓ *Simple to distribute because you only need to upload a single application to the server.*
- ✓ *Simple to scale because you just run multiple instances of the same application behind a load-balancer.*

Unfortunately, more the application grows in size, more the monolithic architecture complicates and slows down the development process, so agile development and application distribution become impossible. The bigger the software gets, the less developers understand it. As a result, finding bugs and fixing them becomes difficult and time consuming. Furthermore, with the advent of new technologies, the libraries used in the development of the product could become obsolete and make the application vulnerable. Testing the application and deploying it also become a complicated process because an error in a single module can cause the entire software to malfunction. Even scaling the application could become difficult because it is possible that different modules have different resource requests (it may be that one module requires a lot of storage space while another needs to process and therefore requires intensive utilization of the CPU). Consequently, when we are developing applications that are excessively growing, it would be ideal to migrate to a microservices architecture.

2.4 Microservices advantages and disadvantages

When we decide to structure the application with a microservices architecture, the keyword is modularity. In fact we will decompose the software into many independent modules that are developed and understood by different teams of developers. In order to isolate the different modules each of these must have its own database. Microservices architecture has several advantages:

- ✓ *The most important is that it allows you to implement continuous delivery and distribution of large and complex applications. The continuous delivery practice is part of* DevOps, *which is a set of techniques aimed at making software delivery fast, frequent and reliable. There are many reasons it allows you to implement continuous software delivery. First of all, since the application modules are modest in size and independent from each other, they can be analyzed with automated tests that are easy to write and quick to execute, as a result the application will contain fewer errors. Since each service can be deployed independently, this saves developers on a team from having to deal with other teams to make changes to the module, making it easier to push frequent changes into production. Furthermore, the fact that each module is isolated allows the organization to structure development teams as small units made up of a few elements that manage one or more modules. Each team can therefore scale, develop and deploy their services independently from other teams, resulting in much faster development speed.*

- ✓ *Services are small in size and are easy to manage because they don't slow down the development environment, start up quickly, and are simple to understand and modify.*

- ✓ *Each service can be resized as desired and therefore the scalability of a module is independent from that of the others. Furthermore, each service can be deployed on the hardware with the best specifications for it.*

- ✓ *There is greater error isolation, in fact, in case of a failure of one service the others will still continue to function normally.*

- ✓ *It is easier to experiment and adopt new technologies, even rewriting a service from scratch is not excessively expensive if the need arises, while in the case of a monolith structure the entire application would have to be rewritten.*

Unfortunately, the microservices architecture, although practically mandatory in the case of developing complex applications, still has some disadvantages:

- ✓ *Decomposing the system into microservices is not simple, and there is no well-defined algorithm to do it. If the system is decomposed incorrectly, there is a risk that the various components became dependent on each other, creating a "distributed monolith" in which all the advantages of microservices architecture are lost.*
- ✓ *Distributed systems are complex and the developer has to put more effort into the development also because each module has to communicate with the others through communication mechanisms between services. Furthermore, each service must be designed to be able to handle errors of other services whether they are unavailable or have high latency. Writing automated tests that span many services is also tricky. As a result, the developers in the organization must have software development and delivery skills to use the microservices architecture correctly.*
- ✓ *Deploying features that span multiple services requires careful coordination between the teams working on those services.*
- ✓ *Another difficulty lies in deciding when to adopt the microservices architecture. This is a problem of newborn applications where it is not known how quickly the software will evolve. In fact, initially it could be more demanding to structure the application with a microservices architecture and therefore you could opt for a monolithic one. Later, when the problem*

becomes complexity, however, it will necessarily be required to think about how to decompose the application into microservices.

- ✓ *To correctly exploit all the advantages of the microservices architecture it is essential that the organization applies* DevOps *practices or the advantages will be very limited.*

In the past, the goal of software architectures was scalability, reliability and security. In modern architectures, however, it is essential to have fast and secure software delivery. The microservices architecture provides this quality along with excellent testability and maintainability.

2.5 Building microservices architecture

Every application has two categories of requirements, the functional ones that define what the application must do and those for quality of service such as scalability, reliability, maintainability, testability and deliverability. The choice of architecture decides the quality of the service. To define the microservices architecture of an application we must proceed in steps:

1. *We write the requirements in the form of user stories identifying the operations to be carried out (therefore the functional requirements). After defining the functional requirements, we derive the base classes and from these the operations performed by the application. The operations performed by the application can be of two types: commands or requests, the commands create, update or delete data while the requests are read functions. Each operation must then be described through the required input fields, the returned value, the preconditions and the post-conditions. The requests also provide information on the composition of the graphical interface.*

2. *From the functional requirements we identify the services. Unfortunately there is no mechanical process to perform this step but there are several decomposition techniques that try to solve the problem from different perspectives. The main technique is the decomposition by business capability in which each operation is divided by category (for example all logistics operations will be grouped together), after which the functions that are closely related can be in one service, otherwise they have to be managed by one service aside. The advantage of this decomposition strategy is that the architecture should remain virtually unchanged even if some aspects of the business change. There are two fundamental principles to follow when decomposing the application into services: the single responsibility principle and the common closure principle. The principle of single responsibility requires that each service has only one responsibility so that it has maximum stability. The common closure principle says that in the hypothesis of making any single change to the application, this must only alter the code of a single service. Consequently, elements that change for the same reasons or factors must be brought together in the same service. Doing this will reduce the number of services to modify when changes need to be made. Ideally a change will affect only one team and a single service. This rule should avoid the birth of distributed monolithic architectures (for example microservices architectures with the same problems as monolithic structures). Other problems in the decomposition of services that need to be taken into account are:*

    ✓ *Service latency, if multiple services have to communicate too frequently slowdowns could occur, the solution could be to combine these services into one.*

- ✓ *Synchronous communications between processes reduce the availability of services, the ideal is to implement asynchronous communications.*

- ✓ *Some operations update the data of multiple services, this update must be done using a "saga", that is a sequence of local transactions coordinated using messages.*

- ✓ *Obtaining a consistent view of data across different databases is generally not feasible in a microservices architecture, fortunately this need is very rare.*

- ✓ *Most applications have a central class that is used in any operation, in this case it is necessary that each service has only the elements of that class necessary for its operations so that the central class (belonging to the central service) is not invoked each time a single operation takes place.*

3. *Now we have a list of system operations and a list of potential services, at this point we need to decide what the APIs are and how the services interact with each other. First we need to figure out which service each system operation belongs to. In general, it is convenient to assign the system operation to the service that has the information to satisfy it. Next you need to figure out which operations need to invoke multiple services. To interact with a client, services can use synchronous or asynchronous communications. In synchronous communications, the person making the request stops waiting for a response, in asynchronous ones, instead, he continues to work and expects a response in the future. Today, APIs are developed with the REST style, a communication style between processes that uses HTTP. A*

key concept in REST are resources, which typically represent a single object or a collection. REST uses HTTP to manipulate resources that we refer to with a URL. For communication between processes instead we can use an asynchronous communication strategy. You can adopt a strategy that uses a broker and who acts as an intermediary or one in which the services communicate directly with each other. One way to asynchronously handle communication between services is to make a service that receives a synchronous request, responds with data from its local database and then asynchronously sends the requested information to the other services to check if the data matches and if necessary, update your database.

The major problem of distributed systems such as microservices ones is the complexity of the interaction between the various components. In fact, a single request from the client may require the response of several different services, each with its own database which must be consistent with respect to the others. Sagas are a mechanism for maintaining consistency between data in distributed transactions. You need to define a "saga" for each command that needs to update data in multiple services. In reality, a "saga" is nothing more than a sequence of local transactions. Each local transition updates the data in a single service using ACID transactions. Terminating a local transaction in a saga starts the next local transaction. The coordination logic of a "saga" can be of the "choreography" or "orchestrated" type. The first typology distributes the decisions among the participants of the "saga", in fact they communicate by exchanging events. The second typology centralizes the coordination logic of the "saga" in a single class that sends command messages to the participants of the "saga" telling them which operations to undertake. Unfortunately the "saga" mechanisms are not isolated from each other and in case of use it is necessary to write countermeasures to remedy this problem.

Chapter 3

3 DevOps

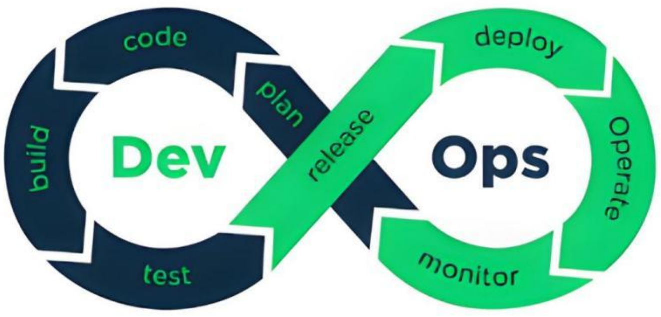


Image 3.1:DevOps techniques improves the process of software development.

DevOps is the set of techniques adopted to speed up the processes that transform an idea (in our case in the software field) into a product delivered to the user and ready for use. In the ideal DevOps system, the developer receives rapid and constant feedback that let him implement and validate his code quickly in the development environment. We achieve this by continuously checking every small code change through automated testing, and this gives us the confidence that the changes we make will work properly in the production environment, and that any problems will be found and fixed quickly. The described situation is more easily achievable when the application architecture is modular, well encapsulated, and loosely coupled so that small teams of developers can work on it autonomously with easily contained small errors that should not cause global malfunctions.

3.1 Visibility, focused team, util job

One of the first steps necessary to implement DevOps is to make the work visible by dividing it into short tasks (by creating long tasks the number of errors in the code increases and also the time to find and fix them, furthermore the testability of the code also decreases) and possibly parallel ones so that each developer can choose which one to carry out and move all those already completed in a common place in order to make the work carried out tangible as if it were the construction of a house in which everyone can see at a glance what stage the work reached. Also, the team of developers working on the project is ideally small and focused. Small to reduce the number of project handovers (which implies update times for each element of the team with each task performed). Focused because every time a developer is assigned to multiple projects, he wastes time (even just to contextualize) when he switches from one to another. It is also necessary to understand where are the bottlenecks in our organization that can slow down the development (they can be both hardware and software or lack of personnel). It is important to eliminate unsolicited work such as the development of features not requested by the user and solve problems as soon as they are discovered without ever postponing to later dates in which it becomes more difficult to trace the source of the problem.

3.2 Learning culture and company transformation process

Image 3.2: The growth of a company is enfatized by the growth of his employees.

To reach a stage where DevOps is truly effective, it is necessary to spread a culture of updating and safety throughout the company. It is necessary that the company does not adopt a punitive philosophy towards those who make mistakes so that the error is found more quickly and those who committed it can understand the problem without being denigrated, and a ***culture of sharing*** is generated through the organization. The process should start from the leaders who aim to guide employees. To test the resistance of the team to errors, some bug can be injected in a controlled way into the application to train the resilience of the developer group. You can also create more automated checks and tests to increase resilience to failure and look for the root of the problem, then disseminate the acquired knowledge within the organization through appropriate mechanisms. By changing the culture of terror into one of ***societal improvement*** the result is continuous staff development and a ***higher rate of problem solving***. When you start the process of transforming a company in order to modernize it and make it follow DevOps processes, you have to proceed step by step. Initially we need to apply the change efforts on small groups of developers willing to change, ideally they should be people respected in the society and who have a strong influence on the organization in order to make our initiative more credible.

Secondly, it is necessary to expand the change to a large part of the groups of the organization so that the improvements of the DevOps process become visible to all. The third phase is to convince groups that still want to keep the old approach. We need to start the third phase only when the DevOps results are excellent and most of the groups are already putting it into practice and can show the benefits of the change. At this point we need to understand how to assign tasks and how to improve workflow. In general, at the end of the process, we would like all team members to implement DevOps practices in their daily routine. One of the best ways to stimulate the improvement of the company is to establish a growth goal to be achieved within a preferably short time limit through DevOps techniques. It also pays to ***reserve 20% of iterative cycles for solving non-functional requests*** (maintenance, manageability, scalability, reliability, testability, deployment and security) and for reducing technical debt so that it does not reduce the ability to develop and resolve errors quickly. In this phase it is also necessary to gather information on the development process in general to lay the foundations for the next steps.

3.3 Different types of business organization

After having collected the information necessary to modernize the company we must start organizing the enterprise to improve the workflow. There are 3 types of organization that can be given to your company based on the improvement we want to make:

- ✓ *Functionally oriented organizations that tend to optimize workflow to have specialized staff, tend to practice the division of labor and cost reduction. They aim to help employees grow and have generally vertical structures. Their main problem is that, by dividing the departments by specialization, the efforts to coordinate the groups are complex and there are multiple hands-offs of the work, which slows down the workflow. Even if* *these are not the most suitable companies for* DevOps *they can still adapt to it but each sector must have a policy of high trust in each other and develop mechanisms to ensure that priority jobs are developed immediately by all. A key step when using this type of organization is to work on projects with loosely coupled components so that a single error does not create a ripple effect causing malfunctions in the entire project.*

- ✓ *Matrix oriented organizations would like to optimize both the functional and have a good response to the market. However, this type of organization is usually composed of complex structures often leading to controversial outcomes in which sometimes neither functional results nor an excellent response to the market are obtained.*

- ✓ ***Market-oriented*** *organizations tend to optimize the response to customer needs. They usually have a flat structure, composed of departments that deal with cross-functional disciplines which, however, sometimes lead to redundancies in the organization. However, this is the structure most adopted by organizations that practice* DevOps*. In fact, these organizations should have many small teams able to work independently and quickly on projects, even if this way costs are slightly increased (as opposed to the functional model which tries to cut costs).The goal is to reduce the times requested to generate value for the customer. To make the idea of* DevOps *work, every developer group should also be capable of testing, securing, deploying, and supporting the service they provide. It would also be ideal that every member of the team knows how to carry out, or at least has knowledge of, every part of the development process, so that they are able to complete any activity in that area, in order to avoid changes of hands during development or when you are maintaining the application.*

In general, a necessary step to modernize the company, whatever the type of organization of the enterprise, is the ***inclusion of operational engineers*** in the development teams in order to integrate DevOps practices with daily work and improve development planning.

3.4 Continuous delivery, testing, secure deployment

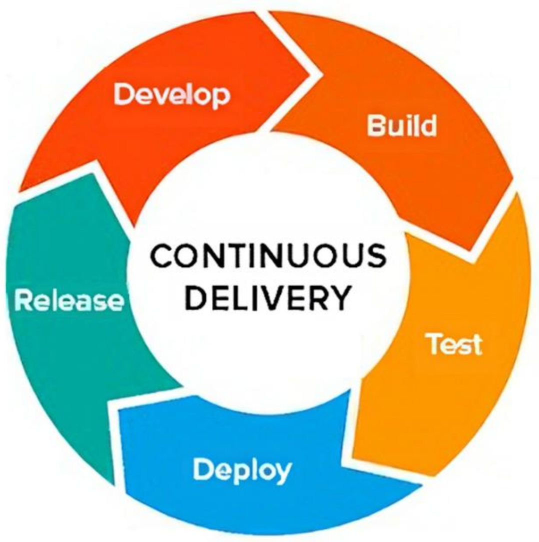


Image 3.3: Using DevOps techniques we can improve delivery, testing and deployment of the application.

After placing functional engineers in the various developer groups, we need to automate the transition from development to production as much as possible in order to avoid errors during deployment by implementing a "continuous delivery" strategy that is realized through a pipeline that tests and releases new low-risk versions. Implementing this strategy shortens delivery times, provides immediate results on code functionality, and ***makes code deployment a part of daily routine.*** To implement this pipeline, the first step is to create different environments: one for development, one for testing and one for production, linked together with a compilation mechanism that allows everyone to update the application without asking permission from third parties. It is also necessary to use a common project repository so that each developer can refer to the updated code and be able to make changes including rolling back to previous versions of the application if required. To verify the proper functioning of the application in all its parts there are different types of automated tests that can be written:

1. *Utility tests are needed to control single methods;*
2. *Acceptance tests controls that the application works correctly in its fullness;*
3. *Integration test are used to verify that the application works well with other applications or services.*

The idea is to find errors with the most specific tests possible, as the unit tests are faster while the slower ones (acceptance and integration) are launched before performing the manual ones. So most of the problems should be solved with unit tests while the more complex ones should be used only at the end for confirmation, also because they require the commitment of more resources and time, and cannot be used in parallel by multiple groups. The primary method of automating application release is to implement feature enable/disable which allows us to activate or deactivate them without needing to deploy to production. In practice, this strategy is applied by inserting the code to enable/disable within a controlled conditional block in the application configurations. The reason why it is implemented is that it allows you to return the application to a functional state only by disabling the new blocks. It also allows the improvement of high-performance features by disabling extra services that are not essential for the application to function. It also increases our resiliency through a service oriented architecture as we can perform the release of incomplete services by simply hiding them inside a conditional block. We can also release complete features of the application in an "obscure" way by simply making them invisible via conditional blocks.

3.5 Microservices architectures

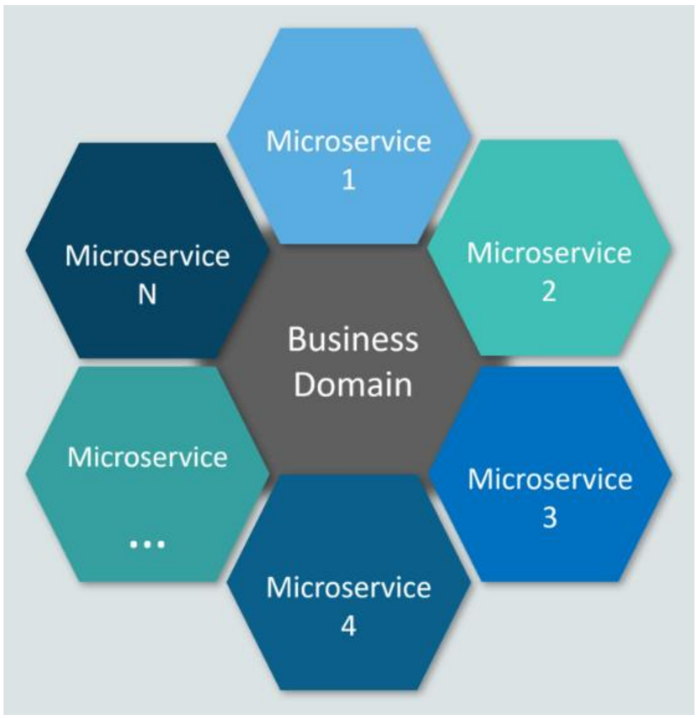


Image 3.4: Splitting the application into smaller service is the way to go to avoid difficulties in deployment, testing and teamwork on an app.

From the point of view of software architecture, a necessary practice to increase productivity, testability and security is to create applications in which the ***different components are loosely coupled*** so that each single component can be made separately distributable changes, allowing to divide the work between different groups. Initially the applications had a monolithic structure in which the various components were closely linked together. In reality this type of architecture is not intrinsically wrong and is optimal when the application is small. However, when the software provides more functionality it is better to structure it as a set of separate services that work together. This type of architecture is defined as microservices. If we are adopting a monolithic architecture and we realize that our application is expanding too much we can convert some features into APIs to correct the project error.

3.6 The error as value and starting point

An organization that knows how to solve problems, maintaining a policy of no denigration for those who make the mistake, and spreading knowledge within the company will certainly achieve better results. In general, whenever a problem is found, a check should be conducted without judgment against the culprits and then explain the problem in a document accessible to all. We must consider each work that takes place as a new source of information also oriented to improve the development process, standardizing a process is fine but we must continually decrease the tolerance for error in order to detect any defects in the strategy we use and consequently improve it. Organization leaders should put employees at ease about making mistakes and learning from them. It is also important to enter errors into the system sometimes to test the resilience of our recovery processes. This is so important that in some companies "game" days are set up. In these occasions a catastrophic event is hypothesized and its recovery is planned, after which, at a specific moment, the event is actually carried out and the company's employees should respond to it as it was decided. By executing these events, you can create even more resilient services and improve staff performance in case that problem actually occur. In general, a culture of learning should be encouraged by putting everyone in the position to talk about their mistakes. **You can use public and automated chats** (whose purpose is to create documentation) within the company to spread knowledge among developers. Furthermore, it is advisable to **move all the company's documentation to a central repository** which makes it accessible to everyone quickly and simply, otherwise there is a risk that at each new job, the developers will have to redesign all the architecture and the code from scratch, even if they are working on a similar projects.

3.7 Logging systems and control metrics

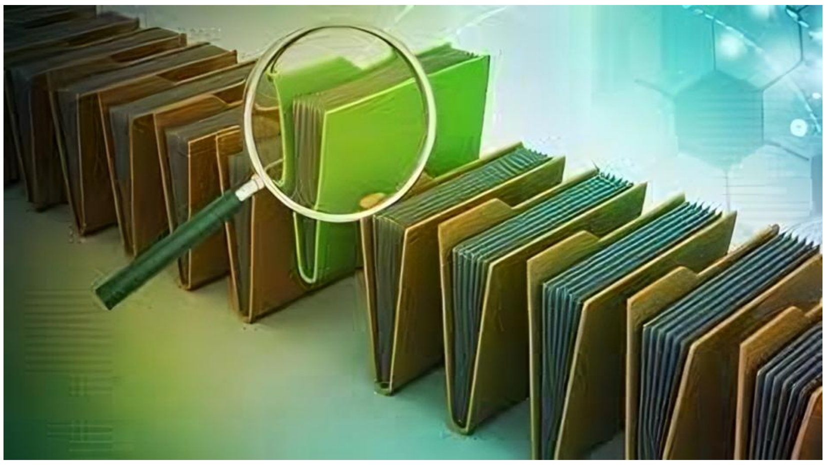


Image 3.5: Analyzing the application in the correct way is really important.

To have total control over the application it is essential to develop systems for monitoring the architecture in order to quickly detect any problem. The elements of monitoring systems are component for collecting data in the form of logs regarding the business logic, the application and the host environment, and a storage point to contain these events. Every developer should, during day-to-day development, create auditing metrics (for both semantic and performance errors). To detect problems, we can use statistical techniques such as mean and standard deviation that allow us to understand when a metric is different from usual. One methodology to improve controls that detect problems is to choose a range of recently detected errors and design dedicated controls that would allow them to be discovered more readily. When the analyzed data does not have a Gaussian distribution we can use anomaly detection techniques with strategies other than standard deviation to detect errors.

3.8 <u>Cloud security</u>

Image 3.6: Security in cloud is a bit particular and imply a shared management between provider and developer.

One of the major issues in DevOps with cloud computing is security management. Security management difficulties are various and can be divided into several categories:

- ✓ *Restriction imposed by industry standards on the use of cloud for certain applications.*
- ✓ *Where they are and where they are archived, who has access to them and backups, how they are monitored and managed, including resilience.*
- ✓ *Controls required for managing firewalls and security configurations for cloud-based applications and environments.*
- ✓ *Concerns about high reliability and service loss in the event of an interruption.*

As with the majority of new technology paradigms, the security issues surrounding cloud computing have become the most debated impediments to widespread adoption. To ensure the trust of businesses, cloud services must provide levels of security and privacy that equal or exceed what is available in traditional IT environments. In a cloud environment, access expands, responsibilities and controls shift, and the speed of delivering resources and applications increases with implications for all aspects of IT security. The approach is to develop security in accordance with each stage of a project or cloud initiative. The security measures implemented in Cloud Security, in addition to requiring specific actions to protect data privacy, are embodied in a set of rules aimed at limiting access to company accounts and regulating the use of external devices, both in cases in which they are the property of employees and used for professional purposes, and in the event of devices of the company used outside the corporation. In particular, Cloud Security policies, must include, among other things, the following elements:

- ✓ *Authentication of accesses, both internal and external to the system, for which the option of registering has been provided.*
- ✓ *Filtering traffic: Because of the presence of appropriate control mechanisms, it is possible to filter traffic from and to the enterprise Cloud system, allowing for the identification and neutralization of any type of information security breach.*

The manner in which the cloud security service is made available in a specific business depends, in large part, on the cloud provider chosen individually, or on the security solutions implemented. The cloud Security plays a key role in internal strategies, as well as providing a number of benefits, the most important of which are the following:

- ✓ *The cloud's security is centralized: because Cloud computing is based on the centralization of data and applications, the Cloud's security is also centralized, ensuring greater process reliability and a lower resource expenditure.*

- ✓ *IT costs are significantly reduced: current Cloud security systems do not require the purchase of expensive hardware, but rather of software dedicated to and managed by specialized providers, who base their primary business on the implementation, updating, and functionality of such systems. As a result of the significantly reduced costs born by the businesses that use Cloud security, the services are of high quality, ongoing, and constantly updated.*

- ✓ *Management and administration reduced to the bare necessities and without compromises: by selecting a provider with a solid track record in the industry, administrators and those in charge of internal security can entrust complete Cloud protection to someone who has been doing so for years and has achieved the best results possible. It will also be the time to say goodbye to manual configurations, and at that point, all action related to software, application, and process updates will be managed at the central level, in fully automated mode.*

- ✓ *Reliability: the Cloud Computing security services provide excellent reliability, provided that appropriate measures are implemented to meet the needs of the business. Because cloud computing is a highly scalable and replicable product, the benefits derived from its use are linked to the reduction of technological costs and the use of smart systems, which provide a significant competitive advantage.*

To ensure cloud security, it is necessary to adopt the following measures:

- ✓ *The first rule in determining the suitability of an organization's internal infrastructure, which consists of hardware and software, is to ensure that the combination of these structures is reliable, appropriate for the purpose, and provided by certified providers. At this point, technicians that specialize in the deployment of* *such systems must understand the whole architecture of the platform, as well as the combination of services and tools available, to ensure that an equitable division of responsibilities exists between the company and the supplier. Simultaneously, when testing the development of new applications, it is critical to follow the provider's instructions. If migrations of existing systems or applications are carried out, it is recommended that the necessary modifications be carried out to ensure optimal distribution.*

- ✓ *Management of the access, to analyze this component, it is critical to have a holistic view of the Cloud, with a focus on the data stored in the system. The Cloud Server* API *connection is used by service managers to verify the consistency and integrity of data stored in the Cloud. It will also be possible to see which users accede to the data, what are the flows, and what processes are used to save, post, send, and share the data. In this sense, the concept of access via strong authentication is a key, given that multi-factor authentication is now the best system available for ensuring a high level of security when accessing cloud-based applications.*

- ✓ *Other security-related actions like the concern the blocking of IP addresses and/or the control of unauthorized access to the company's network.*

- ✓ *Data protection is a topic that deserves its own treatment, but to achieve data protection in the context of cloud security, it is necessary to pay special attention to the flow of information that passes through the cloud. To provide adequate data protection, it is not possible to use very restrictive filters or, in the worst-case scenario, blockages, because daily, routine, and extraordinary operations must always be guaranteed. In this perspective, data protection is one of the most difficult* *challenges in the context of cloud security, as it will be necessary to strike the proper balance between data protection and continuous access to data. Similarly, providers must ensure maximum protection in the event of unintentional disclosures, including the provision of special safeguards in this regard.*

- ✓ *Monitoring and defense are crucial tasks, in fact the IT infrastructure as well as all business devices, must be subject to continuous and costly monitoring, depending on the size of the company, its core business, or specific characteristics that make it unique in its category and worthy of special attention in terms of cloud security. In fact, not all the data that is processed has the same reliability, and the risks may vary depending on the circumstances. Here's why it's critical to revert to a commercial reality that understands how to identify errors in the printing process while avoiding unnecessary duplications and/or systemic flaws.*

To operate a complete monitoring of the Cloud system, for example, testing activities on the reliability of applications are very effective, even before their actual distribution and placement on the market.

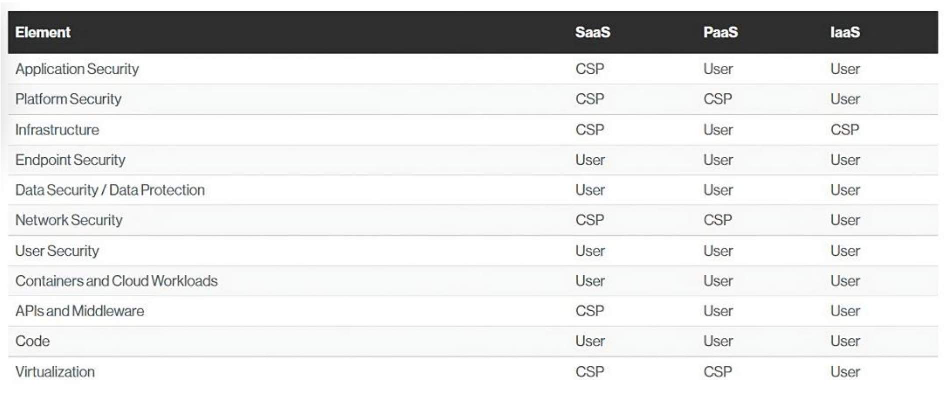
3.8.1 Shared security model:

Although security in the cloud model is mainly delegated to the cloud service provider, have been developed different kind of cloud services for users, each one with different responsibilities for the owner of the infrastructure and the user. This kind of management of the cloud is called Shared Responsibility Model. The Shared Responsibility Model states that the cloud provider, such as Amazon Web Service (AWS) or Microsoft Azure, must monitor and respond to security risks connected to the cloud and its underlying infrastructure. Meanwhile, end users, including individuals and companies are charged with protecting data and other assets stored in any cloud environment. Unfortunately, this concept of shared responsibility is often misinterpreted, leading to the mistaken belief that cloud workloads, as well as any related apps, data, or activities, are totally safeguarded by the cloud provider. As a result, users may unknowingly run workloads in a public cloud that are not fully secured, leaving them open to assaults on the operating system, data, or apps. Because zero-day vulnerabilities are susceptible, even safely setup workloads can become a target during runtime. Shared responsibility rules are different depending on the kind of cloud delivery model we are using:

- ✓ *Software as a service (*SaaS*): is a software delivery paradigm in which a vendor centrally hosts an application in the cloud that a subscriber can utilize. The supplier oversees application security, as well as its maintenance and administration, in this paradigm.*

- ✓ *Platform as a service (*PaaS*): is a platform delivery mechanism that may be purchased and used to build, run, and manage applications. The vendor offers both the hardware and software often utilized by application developers in the cloud platform model; the service provider is also responsible for the platform's and its infrastructure's security.*

- ✓ *Infrastructure as a service (*IaaS*): is a paradigm for delivering infrastructure in which a vendor delivers a variety of computing resources such as virtualized servers, storage, and network equipment over the internet. The company is responsible for the security of anything they own or install on the cloud infrastructure, such as the operating system, apps, middleware, containers, workloads, data, and code, in this paradigm.*



Element	SaaS	PaaS	IaaS
Application Security	CSP	User	User
Platform Security	CSP	CSP	User
Infrastructure	CSP	User	CSP
Endpoint Security	User	User	User
Data Security / Data Protection	User	User	User
Network Security	CSP	CSP	User
User Security	User	User	User
Containers and Cloud Workloads	User	User	User
APIs and Middleware	CSP	User	User
Code	User	User	User
Virtualization	CSP	CSP	User

Image 3.7: In the image above we can see who has the responsibility for security in each type of architecture.

While the Shared Responsibility Model assumes that two or more parties share responsibility for guaranteeing the security of discrete parts inside the public cloud environment, it is vital to emphasize that the customer and CSP do not share responsibility for the same asset. Rather, regardless of the service model type, the CSP or client has full and complete responsibility for the security of any assets under their direct control.

3.8.2 AWS shared responsibility security model:

Because of every cloud provider has different rules regarding security and we choose AWS as our, we need to analyze how security is handled. Security and compliance are shared responsibilities between AWS and the customer. This shared approach may help to alleviate the client's operational burden since AWS operates, manages, and controls the components of the host operating system and the level of virtualization up to the physical security of the structures in which the service operates. The client is responsible for the operation and management of the guest operating system (including updates and security patches), other application software, and the configuration of the AWS security group's firewall. Clients should carefully consider the services they choose since their responsibilities will vary depending on the services used, the integration of those services into their IT environment, and the applicable laws and regulations.

The nature of this shared responsibility provides flexibility and customer control, making distribution possible. AWS is responsible for safeguarding the worldwide infrastructure on which all AWS services are delivered. The infrastructure is made up of hardware and software components, as well as networks and structures that provide AWS cloud services. The customer's responsibility will be determined by the Cloud AWS services chosen by the client. This determines the entity of the configuration work that the client is responsible for as part of his or her security responsibilities. Services such as Amazon Elastic Compute Cloud (Amazon EC2), for example, are classified as Infrastructure as a Service (IaaS) and, as such, require the client to do all necessary configuration and security management tasks. Clients who distribute an Amazon EC2 instance are responsible for the management of the guest operating system (including updates and security patches), any application or utility installed by the client on the instances, and the configuration of the AWS-provided firewall (known as a security group) in each instance. AWS manages infrastructure, operating systems, and platforms for services such as Amazon S3 and Amazon DynamoDB. Clients access the endpoints to archive and recover data. Clients are responsible for data management (including encryption options), asset classification, and using IAM tools to apply relevant authorizations.

Chapter 4

4 Development of the project

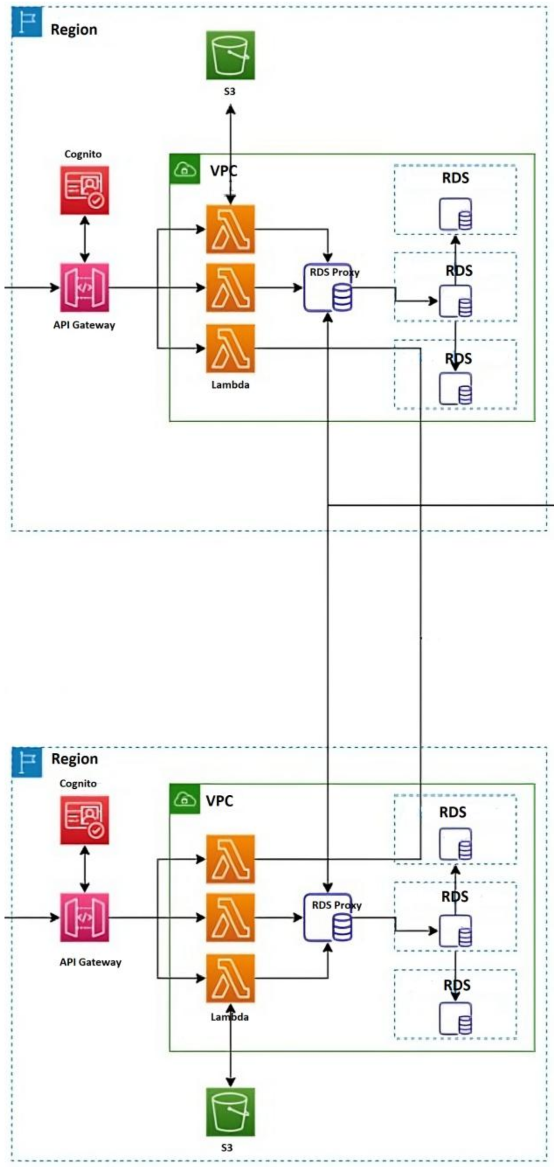


Image 4.1: This is the final architecture of the project

The architecture in the image is the final architecture of the project and makes use of Amazon cloud infrastructure. Here we will summarize the reasons for choosing cloud computing (and hence building a serverless architecture) that have already been discussed in previous chapters:

- ✓ The company is not required to host the service on its own servers, nor it is required to spend in their acquisition.
- ✓ The service may easily scale to the cloud, with a few clicks, or by changing a single setting without the need for further investments in business servers, or by calculating the final size of the service in advance.
- ✓ Instead of devoting part of the organization to server management, it is possible to focus attention on the needs of the user.
- ✓ Costs should be reduced since cloud services are only paid for when they are used (and are optimized with hardware dedicated to the needs).

After having recognized the importance of using Cloud computing we need to choose a Cloud service provider to use. As we can see in the image the service provider that has been chosen is Amazon Web Services (AWS).

The main competitor of AWS is Azure that provide similar performance but in terms of latency AWS is a better option. Also, as AWS is the largest provider of cloud services and has been providing them for a longer period of time than the competitor, it has more experience in the field and generally offers better performance. Another confrontation between AWS lambda and Azure function has brought the following results:

- ✓ Lambda are easier to manage than Azure functions, because they require less configurations to work.
- ✓ Lambda receive JSON formatted input and return a JSON formatted output so it’s easy to interact with them while Azure

functions can receive different input and return multiple and different output adding some complexity.

- ✓ Every lambda has a dedicated resource association that guarantee good performance while Azure functions can share resources each other so there is no guarantee that performance are always optimal.
- ✓ If we had to integrate with Microsoft technologies, which is not the case in our project, probably Azure Functions would be the better option than Lambda.

Another key point in the development of the project is to follow an agile development strategy. That's why we have followed the best DevOps practices. A key point in agile development is to use a microservices architecture indeed of a monolithic one. In small projects a monolithic structure can have its advantages (simple to develop, modify, test, distribute and scale) but if the application grows all these positive traits disappear proportionally. That's why when you develop big applications you need to use a microservices architecture that split a whole application into many components. In this way we can apply continuous delivery and deployment practices also for big applications. Services become easy to manage and to maintain. Moreover the scalability of a component and the hardware on which it is deployed can be varied for every specific need and in case of failure of one of the services that compose the app the others still continue to work. Another point in favor of microservices architectures is that is easy to experiment new technologies and rewrite a single service from scratch in case of need. A concealed benefit of microservices is that they may be reused in other projects if they are written in a generic manner. Microservices architecture bring also some disadvantages like the difficult of decompose the app into various component and make them speak each other, this require higher effort than developing a single monolith application. Another target of DevOps is to make code deployment a part of daily routine by implementing a "continuous delivery" system and build automated tests so that when the deployment occur the tests check immediately the correctness of the results. Therefore we have developed unit tests, acceptance tests, and integration tests to validate the code to follow the principle of DevOps. To help testing the Lambda functions during the development we used a program called Postman that let you send API calls using a simple programming interface. Then we realized continuous integration and continuous delivery (CI/CD) using a set of tools: Gitflow, GitHub, CodeBuild, CodeDeploy and CodePipeline. We will analyze these services in the next chapter, here we will describe only their use:

1. Gitflow to upload the task on Git Repository considering the need of managing a team of developers which work on the project at the same time.

2. CodePipeline let you model the whole release procedure, including the creation of your code, deployment to test environments, testing of your application, and release to production. Every time there is a code change, Amazon CodePipeline then builds, tests, and deploys your application in accordance with the specified process. The creation of this workflow is possible thanks to the fact that CodePipeline can manage the other AWS services.

3. As our version control system, GitHub pulls the most recent modification and transmits the revised code version to CodeBuild whenever there is a change to the code in the GitHub repository.

4. CodeBuild executes the tests and installs the required dependencies. It will push the build artifacts to the S3 bucket once the testing and installation have been completed successfully. To be able to work CodeBuild need that the Lambda has inside its package a buildspec.yml file with all the commands that CodeBuild need to launch.

5. The most recent Build Artifacts are retrieved and transferred to a separate S3 Bucket by CodeDeploy. The newest Build file from the S3 bucket is used by another CodeBuild run to update the relevant Lambda function once the CodeDeploy deploys the code to the S3 bucket.

The tool to add the automated testing in our pipeline is CodeBuild. At point 4 of the preceding bulleted list we can use CodeBuild to run some tests. To setup the tests in our pipeline we need to correctly udpate the package of the Lambda we want to upload to GitHub. In fact after CodePipeline Invoke CodeBuild, to install the dependencies and run the tests, the Lambda package need to have a "buildspec.yml". Then we have to create a folder which contains all the tests we want to run (there are many node.js libraries to create tests that we can use at this purpose, you can choose the one you prefer based on your necessities). After this we add to the "buildspec.yml" a command line that execute these tests.

As we can see in the first image of this chapter the component of the architecture are:

- ✓ The API Gateway manages the flow of data that interacts with a backend service, and implements policies, authentication, and general access control for API calls to protect sensitive data. API Gateway let you create, publish, manage, monitor, and protect REST, HTTP, and WebSocket APIs at any level. It has been chosen because the request the backend receive are API REST and because it can trigger Lambda functions which are the core of our architecture.
- ✓ Cognito provides user authentication, authorization, and management for web apps and mobile devices. It also stores all users data.
- ✓ Lambda is the core service in this project, everything revolves around them. Each lambda is triggered by an API

Gateway call, and interact with S3 and RDS to store and retrieve data to return to the user.

- ✓ S3 is a storage service in which we store the images and firmware data.
- ✓ RDS is the relational database service in which we store the data we receive from the client app.

We will see more in detail each service in the rest of the chapter. We will start analyzing the first decision needed to implement a serverless architecture. Although Amazon has various services to achieve our goal, the most significant ones for our purpose are the "***Lambda***" service and the "***Amazon Elastic Container Registry (ECR)***". We will start evaluating the ECR service first.

4.1 Virtualization and containers

The advent of virtualization in the IT world has allowed for the creation of virtual machines, followed by direct virtualization at the operating system level. Thanks to this possibility, technologies like Linux containers have emerged. These technologies are entirely free of the need to virtualize the bulky hardware component and instead focus just on the functions required to run the microservices that compose cloud-based DevOps applications. These runtime environments are lighter and more agile to manage than virtual machines, and they may be executed directly as applications on the host operating system. These containers can be created and run locally but also managed, created, and launched by online services like Amazon ECR and Amazon EKS.

4.2 Amazon elastic container registry (ECR)

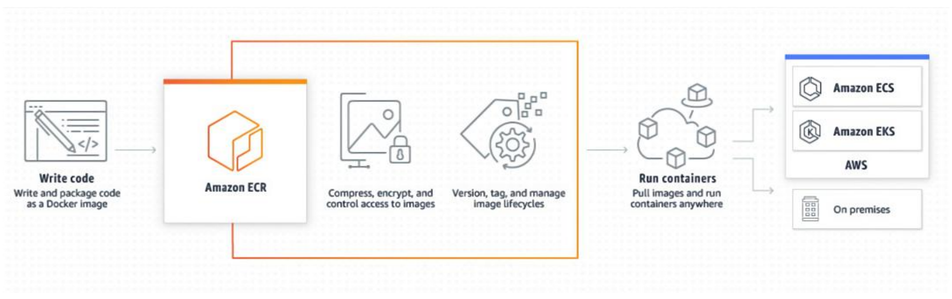

Write code
Write and package code as a Docker image
Amazon ECR
Compress, encrypt, and control access to images
Version, tag, and manage image lifecycles
Run containers
Pull images and run containers anywhere
Amazon ECS
Amazon EKS
AWS
On premises

Image 4.2: In the image we can see how ECR works in a complete application

Amazon ECR is a fully managed container registry that provides higher-level hosting services, allowing you to easily implement images and artefacts from any application. Amazon ECR can send container images to Amazon ECR without installing or scaling the infrastructure, and then upload the images using any management tool and let you publish containerized applications with a single command and easily integrate your self-managed environments and also keeps the most recent images and deletes those that are no longer useful. Use the rules and tag assignment to get to the images quickly. It is also used to share and download images in a secure manner using Hypertext Transfer Protocol Secure (HTTPS) with automated cryptography and access controls. One of the main advantages of Amazon ECR is that it accepts and distribute your images more quickly, reduce download times, and improve availability with scalable and durable architecture. ECR is also safety conscious and fulfill your requirements for image security by using Amazon Inspector's integrated vulnerability management service, which automates vulnerability assessment and ticket routing. Amazon ECR let you easily create containers but their orchestration need to be managed by another Amazon service like Amazon Elastic Kubernetes Services (Amazon EKS). But to speak about Amazon EKS we need to introduce Kubernetes technology.

4.3 Kubernetes

When developers begin to create hundreds, if not millions, of containers that are distributed across various cloud services due to their inherent portability we need to find a way to maintain control over this incredible variety while maintaining visibility over everything that happens in our applications. Kubernetes is an ***orchestration*** and open-source technology. It is used for ***deploying, scaling, and managing containerized applications***. With the development of cloud these kinds of products are destined to become a more widely used standard in businesses. Kubernates can orchestrate containers in private, public and hybrid cloud environment as well as to manage microservices architectures. Containers and Kubernetes are now considered industry standards for developing cloud native applications, and their technologies are available in the offerings of all Cloud Service Providers. Containers are implemented at the operational system level, with all of the benefits it entails in terms of lightweight and scalability. However, they remain isolated, necessitating the use of orchestration software like Kubernates to have visibility of all active containers across the various cloud services. In Kubernates architecture containers are run at the system level on various host machines (nodes) for various cloud services. The containers executed on each machine constitute a pod. Kubernetes oversees identifying all available pods and distributing them across various host machines, while also ensuring that the necessary computational resources are available through an agent called Kublet. Each pod is assigned a unique IP address, which Kubernetes uses to assign the node of choice for container execution. It is possible to automate container management or manually manage them using the Kubernetes API via a wide range of functions. Summing up, Kubernetes is made up of clusters, which are made up of various nodes (host machines) that are tasked with managing the pods that contain the containers to be executed. This type of architecture is associated with perfection in microservices because it allows for the rapid deployment of all clusters required to automate the management of a large number of containers distributed virtually anywhere in the cloud. This visibility is communicated to the end user via a single dashboard, allowing them to monitor the proper operation of the containers and plan all of the operations required to develop and maintain the applications. Kubernetes' operational logic provides a number of critical benefits for the ability to automate the following processes:

- ✓ ***Deployment**: the ability to create new container locations, as well as manage container migration from other environments and the elimination of those that are no longer required.*

- ✓ ***Monitoring** is a key function of Kubernetes-based orchestration, ensuring real-time visibility of the processes running on the many private, public, and hybrid cloud services on which the development team relies to build the software components. The automation allows you to evaluate whether or not the containers are working well in real-time, to reroute those who have been arrested for whatever reason, and to remove those who, based on various criteria, are no longer required in the various work assignments.*

- ✓ ***Load Balancing**: a very useful function for optimizing cloud network resources, thanks to the ability to distribute traffic based on the needs of individual containers, which comprise the workload to be managed.*

- ✓ ***Storage**: management of operations necessary for container archive in hybrid and cloud environments, in order to meet all requirements in an automated manner.*

- ✓ ***Optimization**: the deployment analysis allows you to optimize your computing resources based on the needs of each container, assigning them to the node whose availability is closest to their satisfaction.*

- ✓ ***Security**: automated management of all required authentication data such as passwords, tokens, SSH, and so on.*

As expected, the primary Kubernetes element is represented by the cluster that is obtained each time a deploy is performed. The cluster is made up of nodes that execute the users' containers. Every cluster, to exist, must have at least one Worker Node, which is managed by a Control Plane. The clusters generally are implemented to reach the desired high availability. A high availability to ensure the best possible business continuity. To avoid a cluster failover causing a disruption in the operation of the containers that run the applications, more Control Plane are being used at the moment.

4.4 Docker

Kubernetes is frequently associated with Docker in the container world. Docker is now available on nearly all of the major cloud service providers, beginning with AWS and Microsoft Azure. Unlike Kubernetes, Docker's main application does not focus on orchestration. Rather, Kubernetes and Docker complement each other well. Docker allows you to run, create, and manage containers on a single operating system, whereas Kubernetes allows you to automate provisioning, networking, load balancing, security, and scalability of work containers on available nodes. Everything is possible thanks to a single dashboard. In more recent times the Docker company has released Docker Swarm, a standalone application for managing Docker containers, which can obviously also be managed using Kubernetes. Swarm was created to provide a simpler alternative to Kubernetes, with fewer commands and the added benefit of being designed to manage a technologically inferior variety. Docker Swarm is obviously not the only alternative to Kubernetes, which has competitors such as Apache Mesos and Jenkins. Apache Mesos is an open-source cluster management known for its integration with machine learning tools such as Cassandra, Kafka, and Spark. Jenkins is an open-source platform focused on continuous integration and continuous delivery, which is typical of cloud native applications built using DevOps methodology. Among these technologies, Kubernetes is unquestionably the most popular and widely distributed, both in open source and commercial distributions.

4.5 Amazon Elastic Kubernetes Services

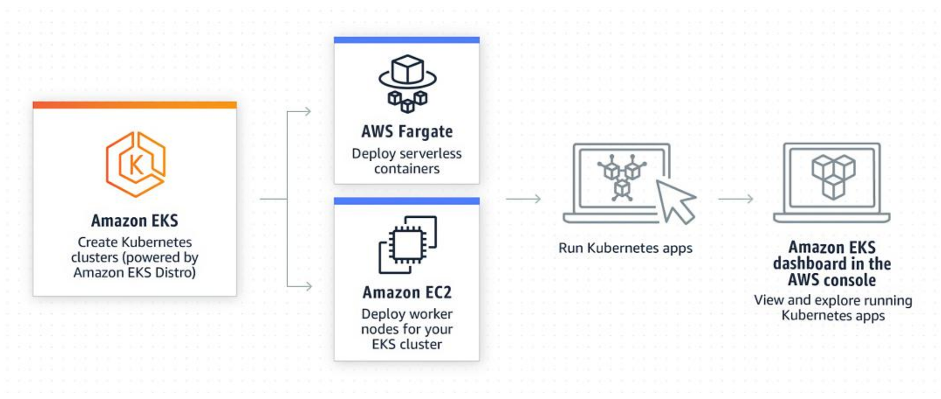

Amazon EKS
Create Kubernetes clusters (powered by Amazon EKS Distro)
AWS Fargate
Deploy serverless containers
Amazon EC2
Deploy worker nodes for your EKS cluster
Run Kubernetes apps
Amazon EKS dashboard in the AWS console
View and explore running Kubernetes apps

Image 4.3: In the image we can see the working principle of EKS

Amazon Elastic Kubernetes Services (EKS) is the most focused way to ***advance, execute, and scale Kubernetes on AWS cloud***. Amazon EKS automatically manages the availability and scalability of the Kubernetes control plane nodes in charge of container scalability, application availability management, cluster data archiving, and other key processes. With Amazon EKS, you can take advantage of AWS's performance, scalability, dependability, and availability, in addition to integrations with AWS's network and security services. On-premises, EKS provides a fully supported Kubernetes solution with integrated tools and a simple implementation in AWS Outposts, virtual machines, or bare metal servers. EKS can be used for different purpose. It manages your Kubernetes cluster and tasks in hybrid environments and run Kubernetes in your data center. It can be used for machine learning by executing distributed training processes efficiently using the latest GPU-powered Amazon Elastic Compute Cloud (EC2) instances. It can be used to create web applications that automatically size themselves and execute highly available configurations across various availability zones (AZ) with ready-to-use networks and security integrations.

4.6 Lambda review

Lambda is a service that lets you run your own programs without the need to provision or manage the servers. Payments are designed in such a way that charges correspond to use time, and non-executed code is free. You only need to upload the code to Lambda, and the service will handle the execution with high availability. Because of the trigger feature, Lambda functions may be invoked from any app or from other Amazon services. Assigning Lambda the due permissions they can interact with all other AWS services to completely handle them programmatically.

4.7 AWS container app development vs AWS function as a service

So, having analyzed the technologies, the first confrontation in development technologies is between AWS function as a service and containers. After some researches we got the best situation in which you should use one or the other technology. You should use lambda if:

- ✓ ***You have a modest application that runs in 15 minutes or less on demand****. Lambda functions have a timeout value that may be set anywhere in 15 minutes. Lambda terminates functions that are running for longer than their time-out value.*
- ✓ *You are unconcerned about or require sophisticated EC2 instance setup. Lambda maintains, provisions, and protects your EC2 instances, as well as offering target groups, load balancing, and*

auto-scaling. It ***removes the complication of maintaining EC2 instances****.*

- ✓ ***You want to pay only for the capacity that is utilized****. Lambda costs are calculated based on the number of milliseconds consumed and the number of times your code is executed. Costs are proportional to consumption. Lambda also offers a free tier of service.*

And you should use EKS in the following situations:

- ✓ ***You are running Docker containers****. While Lambda now has Container Image Support, EKS is a better choice for a Docker ecosystem, especially if you are already creating Docker containers.*
- ✓ ***You want flexibility to run in a managed EC2 environment or in a serverless environment****. You can provision your own EC2 instances or Amazon can provision them for you. You have several options.*
- ✓ *You have tasks or batch* ***jobs running longer than 15 minutes****. Choose EKS when dealing with longer-running jobs, as it avoids the Lambda timeout limit above.*
- ✓ ***You need to schedule jobs****: EKS provides a service scheduler for long running tasks and applications, along with the ability to run tasks manually.*

Because of our app has only the need to call for API REST that save and retrieve data from the database they doesn't need to last more than 15 minutes. Moreover being our Lambda really fast it should be better to pay for their usage only when they are running and this is another point in favor of Lambda, in fact EKS need to reserve space in a cluster that costs a fixed price based on the amount. EKS require a lot more configurations than Lambda so if don't want to add further complexity to the management of the project Lambda is a better option. With Lambda it is harder to make misconfiguration errors and they have a really high reliability. So at the end our architectural choice is to use the ***Lambda functions as base of the project***.

4.8 Architecture

Following the most modern development strategy we decided to adopt a ***Function as a Service serverless architecture***. The "function as a service" or "*FaaS*" is a type of cloud service that allows you to manage the application as a set of functions. In practice, we implement this architecture by exploiting the versatility of Amazon AWS services and in particular of the Lambda service. In Amazon AWS services, the central development tools of a generic Function as a service serverless architecture are certainly the API Gateway, the Lambdas, and the RDS database.

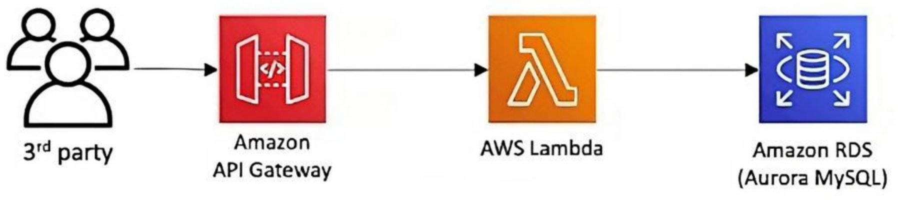
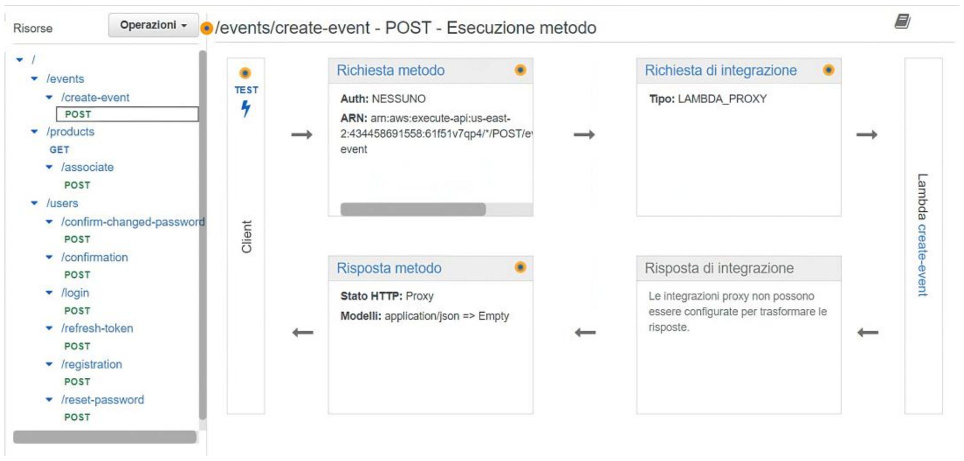


Image 4.4: This is the basic architecture of our application

4.9 Base architecture

In the architecture described, the client sends requests to the API Gateway which executes the Lambda function requested by the user and returns a response. Depending on whether it is necessary to communicate with the database, the

Lambda may have to connect to the Aurora RDS service. Starting from the API Gateway we will analyze this infrastructure and its capabilities more in detail:



/events/create-event - POST - Esecuzione metodo

Richiesta metodo
Auth: NESSUNO
ARN: arn:aws:execute-api:us-east-2:434458691558:61f51v7qp4/*/POST/e
event

Richiesta di integrazione
Tipo: LAMBDA_PROXY

Risposta metodo
Stato HTTP: Proxy
Modelli: application/json => Empty

Risposta di integrazione
Le integrazioni proxy non possono essere configurate per trasformare le risposte.

Image 4.5: This is an example of how an API rest is visualized in the API Gateway

Here we have an average stage of API Gateway service. We can see that authentication and some products and events resources are managed by API Gateway calls. Each of these API is associated with a lambda developed specifically to realize the task requested by the user. API Gateway methods can require different types of header to correctly answer the user. In particular the authenticated requests, usually the ones used after authentication, need the user to send back the authentication access token to verify that the user is authorized to access the services he requests.

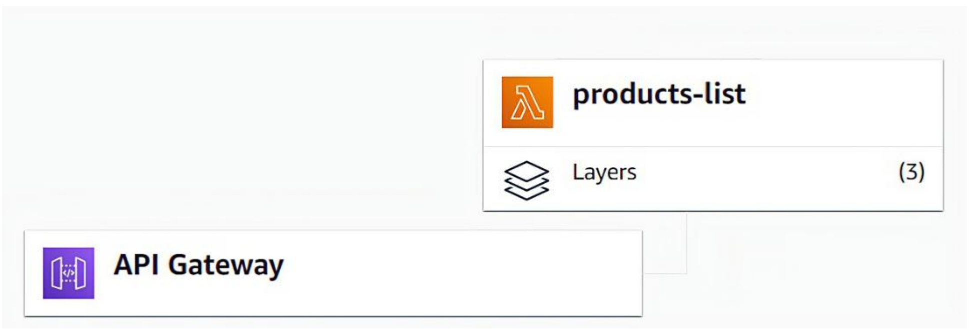


Image 4.6: Visualization of a lambda and its API Gateway trigger in AWS lambda panel.

In the upper part of Lambda panel we can see the Lambda function name and the number of layers that the Lambda can use and the trigger, which is the element that makes the Lambda launch. In our case the Lambda are started by an API Gateway call initiated by a user. Then in the code panel, we can see the code that compose the Lambda. The code can be written directly on the Lambda or uploaded from zip file or using other injection techniques that exploit other services. The Lambda can also be tested on place using the testing functionalities and its logs can be seen directly on Lambda service console or in CloudWatch service. In the lower part of the Lambda panel we can find the runtime configurations and the code properties but most important we can manage the layers, which works like libraries for the Lambda function, adding new layers give the ability to the Lambda to use the code inside the layer itself. The Lambda configuration panel can further assign other options to the Lambda. The ones we used are the authorization, the environment variables, the VPC and the Server Proxy ones. The authorization panel let you assign the Lambda a role that grant it the permissions necessary to access the AWS resources needed by the function. The environment variable panel let you declare some key-value variable that could be necessary for the Lambda to work, in general is better to declare these variables inside the Secret Manager which is a centralized resource pool in which the variable can be written once and read from all the Lambda, differently than environment variable which can be reached only by the Lambda that declares them. The VPC can be used to move the Lambda in a specific subnet for infrastructure compatibility reasons. The Server proxy panel let you add a proxy to the database, it will be automatically linked to the Lambda and the chosen database, if created in this panel. Other two important Lambda panels are the versioning and aliasing usable for version control. In the version panel we can create new versions of the Lambda, so the code inside the Lambda will be saved and kept secure as that version of the Lambda, we won’t be able to edit that Lambda code but we will be able to edit the actual Lambda code until we feel enough sure to deploy another Lambda version. The alias panel can associate any version of the Lambda to a named pointer that indicate the important versions of the Lambda, like the actual in deployment or the one that is being tested.

4.10 Secret Manager

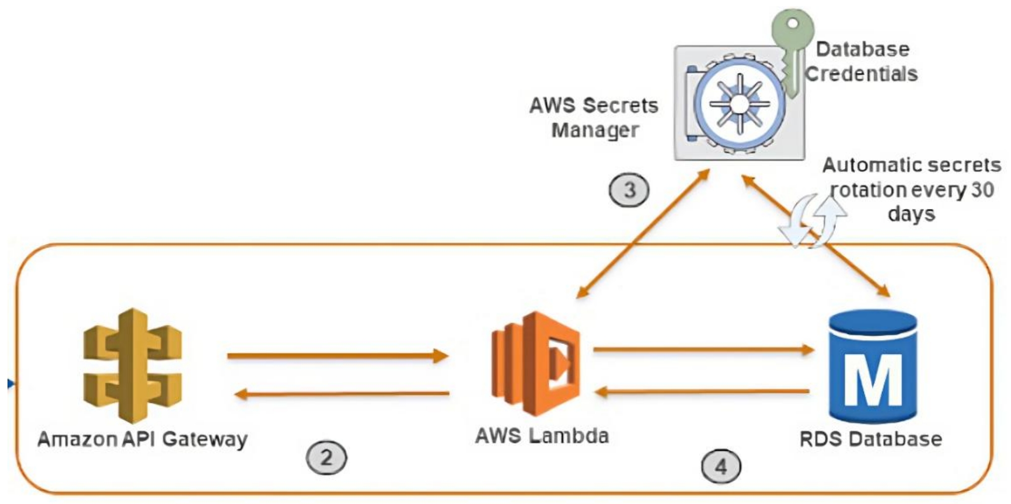


Image 4.7: In the image we added to the basic infrastructure the secret manager component

The next piece of the architecture to add is the Secret Manager. This service let you store “key-value” pairs and full access credentials to database making easy for developers to avoid the necessity to write in clear these values on the functions they create. Another advantage of Secret Manager is the automatic secrets rotation every 30 days for database access credentials, this is an optional service and can be selected during secret creation, and will automatically update the database password every a fixed amount of days, actually this is not implemented in the project but probably it will for security reasons.

4.11 Test with API Gateway

To manage efficiently the versions of microservices we need a way to handle the new updates. In our situation ***each Lambda is a microservice***. Our develop chain starts locally using Visual Studio Code, that edit a local repository of our functions. Then to share the work with other developers we use Gitflow to upload on Bitbucket the new updates. Thanks to Gitflow each developer can work on different tasks by creating new feature branches. After this process we can upload the updated Lambda also on AWS. Here the work does not end. In fact, each time we successfully add new features to a Lambda we can publish a new version of the Lambda. Then we can create new alias to which we can assign a specific lambda version. Generally, in a project the ***two most important alias used are*** “***development***” and “***production***”. The version to which we will associate production is usually the one fully working and exposed to the users. The “development” alias is generally associated to the “LATEST” version (so the one we are editing now) and is used to apply new modifications to the microservice. In the future when the “development” alias will be stable and perfectly working we will produce a new version of the lambda and we will associate it with the alias “production” exposing it to the public. But this process is not completely automatic. In fact, here we will need to interact with the API Gateway. At the actual state of art illustrated on AWS documentation we will need to deploy two stages of the API gateway, one called “development” and one called “production”. To each of these versions will be given a stage variable that will let to call the correct alias of the Lambda selected. This way of working is really important to follow the DevOps principles, in fact we will have at disposition two environments, one that is the stable version for the user and one we can work to create new updates without damaging the production environment.

4.12 <u>CloudWatch</u>

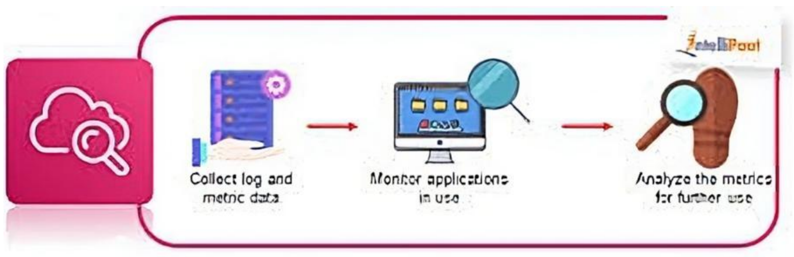

Collect log and metric data
Monitor applications in use
Analyze the metrics for further use

Image 4.8: CloudWatch is the AWS monitor service

Amazon CloudWatch is a cloud AWS resources monitoring service. With Amazon CloudWatch you can collect, monitor and keep trace of parameters and logs and also set alarms. Setting an alarm is needed when you want CloudWatch notify you if some event happens. In the metrics panel of CloudWatch you can trace the parameters of the application and also draw graphs by choosing some of them. The trace of logging is the resource we used more in our project, in fact to test results of a Lambda after it has been launched by a service the only way is to search the Lambda on CloudWatch log panel.

4.13 <u>Relational Database Service (RDS)</u>

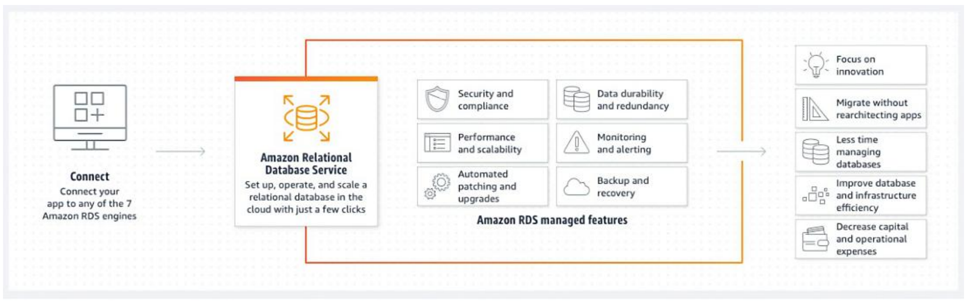

Connect
Connect your app to any of the 7 Amazon RDS engines
Amazon Relational Database Service
Set up, operate, and scale a relational database in the cloud with just a few clicks
Security and compliance
Data durability and redundancy
Performance and scalability
Monitoring and alerting
Automated patching and upgrades
Backup and recovery
Amazon RDS managed features
Focus on innovation
Migrate without rearchitecting apps
Less time managing databases
Improve database and infrastructure efficiency
Decrease capital and operational expenses

Image 4.9: Amazon Relational Database Service let you manage the most used database engines

Amazon Relational Database is a relational database service that let you choose between six of the most used database engines: Amazon Aurora, MySQL, MariaDB, PostgreSQL, Oracle and Microsoft SQL Server. Amazon RDS manages all the standard database activities like provisioning, backup, application of patches, recovery, error detection and restoration. Thanks to RDS is possible to use replication to improve availability and reliability for high workloads. There are two possible implementations:

1. *Use an RDS read replica instance which is an asynchronous read-only replica, it can be used by your application for any query that does not require a change in the data. In this way we are reducing the load from our master database.*

2. *Use a Multi-AZ means that the database has a standby spare server in another availability zone of the same region. This is a synchronous replica but it cannot be directly accessed until the active server fails, if it happens the spare server takes over and start handling traffic quicker than it would be possible without the spare one. Multi-AZ improve the deployment reliability since version upgrades, backup snapshots and creation of replica can be done using the computational power of the spare server. The major problem of Multi-AZ is that it doubles the costs of the*

service to reserve the spare server resources. The main use of Multi-AZ is therefore the fast recovery generally reserving a Multi-AZ spare server only for the master server.

3. *Actually, there is the option to mix the use of Multi-AZ and replicas. This will give the server the ability to recover faster from failures thanks to Multi-AZ properties and the ability to easily scale when read-heavy workloads happens thanks to replicas. In this implementation the replica are update asynchronously like always but a replica can be also converted into a standalone database instance implementing the Multi-AZ architecture.*

The advantages of RDS are various:

- ✓ ***Easy to use**: to interact with the database you can use the command line interface of Amazon RDS or API calls or even the AWS management console. Databases are optimally pre-configured basing on the input you chose when creating the database. So in minutes you are ready to start using your fully working database. With Amazon RDS the software of the relational database is always updated to the last version.*

- ✓ ***Better performance**: It is possible to regulate the performance of our database during the creation of it. We can choose the speed of input/output operation that we prefer. Read and write operations on RDS database perform better than a normal database instance, this happens thanks to the technologies put in place by Amazon, this service has no extraordinary costs.*

- ✓ ***Scalability**: It is possible to modify the computational power and the memory that sustain the database in few minutes. The dimensions of database storage are automatically augmented by AWS RDS if needed until limits imposed by the service or set* *by the user are reached. Moreover, the storage recalibration happens in real time without inactivity moments.*

- ✓ ***Availability and Durability**: AWS RDS dispose of an automatic backup function that let you recover the database in its preceding state until 35 days earlier. It is also possible to make snapshots of the database on user requests, these are full backup that stay in memory until user decide so (cost increase depends on the storage occupied by snapshots). In case of hardware problems Amazon RDS automatically change the computational instance that sustains the distribution.*

- ✓ ***Security**: all Amazon RDS database data can be encrypted using the keys managed with AWS Key Management Service (KMS). If the user decide to encrypt the data all database storage, replicas, multy-AZ, snapshots and backups are encrypted. Amazon RDS also support SSL protection of data in transition. RDS let you execute database instances on Amazon VPC, that let you isolate each database instance and connect them through a virtual infrastructure using VPN IPsec using state of art encryption methods. It is also possible to decide the kind of traffic that can reach the database editing the firewall configurations. Amazon RDS let the access to its resources only to those which have the correct IAM roles configurations, it is also possible to assign tags to database instances that let only certain roles to edit them.*

- ✓ ***Manageability**: using Amazon Cloudwatch it's easy to control RDS database parameters, like computation capability, memory and storage, I/O activities and connections to the instance. RDS can also send you notification about database events using Amazon SNS.*

- ✓ ***Costs reduction**: costs are computed exclusively on resource usage. Amazon RDS let the user to reserve instances till 3 years* *in exchange of discounts. It also stops the database instances for 7 days, after that time the resource is automatically restarted, and you pay again the price to keep it online. This is done to encounter the need of developers which can enable databases only when developing and disable them when they stop working.*

4.13.1 AWS RDS MySQL:

Image 4.10: MySQL is the most famous database engine

When we create a database service with AWS RDS we can choose between many database engines. MySQL is one of the most important. It is coded in C and C++ and is the most famous and used relational database. It is really quick and user friendly, open source and freely available, moreover is fast and reliable. During the first tests, we started using MySQL mainly for its simplicity. Amazon RDS makes configuration, administration, working and scalability of the cloud MySQL database easier so that in minutes you can start developing with a fully functional database. The MySQL database was only a test one in our project, in fact to satisfy the requests of the customer we require high reliability and scalability capacity. Therefore we have to move to Amazon Aurora.

4.13.2 AWS Aurora:

Image 4.11: Aurora is a really fast and scalable relational database engine

For database administrators and software developers, the current database landscape is a frustrating one. You are forced to choose between open source databases that are challenging to scale and old guard databases that offer enterprise features but are expensive. With Amazon Aurora, a relational database developed for the cloud, you can get the speed and availability of commercial databases with the cheap cost and flexibility of open source databases. Aurora is MySQL and PostgreSQL compatible, making it an ideal choice for your new or current applications. It's up to five times quicker than MySQL and three times faster than PostgreSQL. It also scales automatically to keep up with your applications. Aurora produces six copies of your data that are scattered across several places and continually backup them to Amazon S3 so that your data are safe and internationally distributed. Aurora can duplicate your data across different regions for improved local performance and catastrophe recovery. You may use Aurora's serverless to autonomously start, scale, and shut down a database to match application demand or to easily manage unexpected workloads. Aurora provides enterprise-grade performance, scalability, availability, and security with the ease and cost-effectiveness of an open source database migration to the cloud.

4.14 Adding the RDS Proxy:

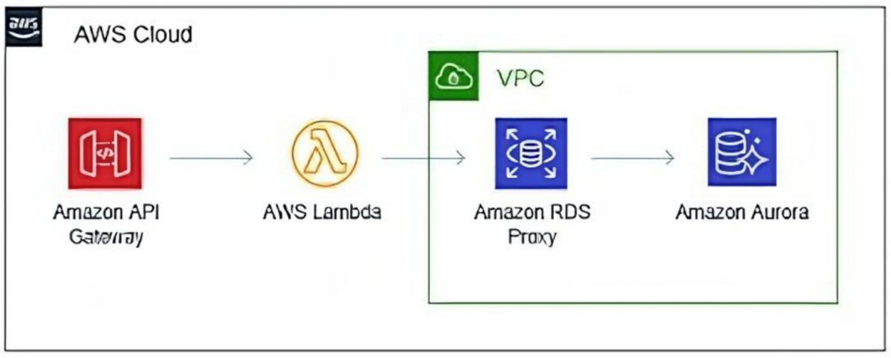


Image 4.12: We added the RDS Proxy to better manage the connection to the database

The use of Amazon Aurora database cause the next update in our architecture. In fact to fully exploit the capabilities of Aurora we need to connect it to Lambda functions via the Amazon RDS proxy. Amazon RDS Proxy is a fully managed, highly available database Proxy for Amazon Relational Database Service (RDS) that improves the scalability, resilience to database failures, and security of applications. Many applications, especially those based on contemporary serverless architectures, might have a large number of active connections to the database server and may initiate and drop database connections at a rapid pace, draining database memory and computation resources. Amazon RDS Proxy enables applications to pool and share connections created with the database, boosting database efficiency and application scalability. RDS Proxy reduces failover times for Aurora and RDS databases by up to 66%, and database credentials, authentication, and access are controlled via interaction with AWS Secrets Manager and AWS Identity and Access Management (IAM). Most apps can be enabled to use Amazon RDS Proxy with no coding modifications. To begin using RDS Proxy, you don't need to provision or manage any extra infrastructure. The pricing is straightforward and depends on the capacity of the underlying database instances. Amazon RDS Proxy support all Amazon RDS database engines.

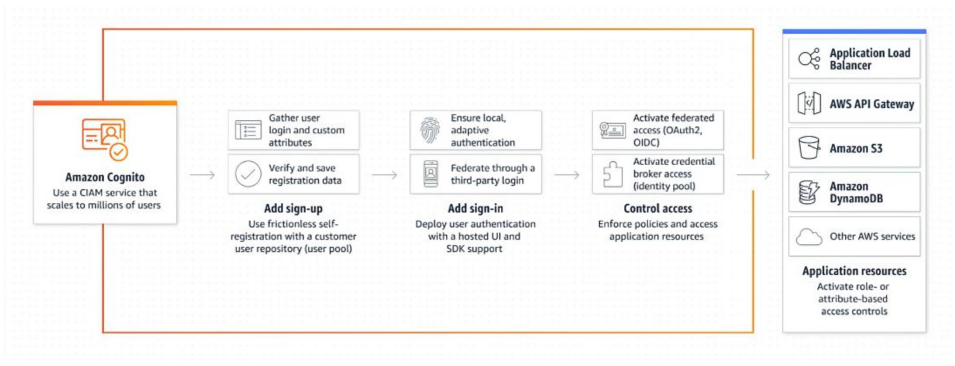
4.14.1 Performance improvement of using and RDS Proxy with serverless applications:

With Amazon RDS proxy, you can create serverless apps that are more scalable and available because they make better use of your relational databases. Modern serverless apps handle highly changing workloads and may seek to initiate a burst of new database connections or maintain a large number of open but idle connections. A spike in connections or a large number of open connections may put a load on your database server, resulting in slower queries and limited application scalability. RDS Proxy enables you to effectively grow to many more connections for your serverless application by pooling and sharing already existing database connections. RDS Proxy also allows you to restrict the total number of database connections that are established, allowing you to maintain predictable database performance. Finally, RDS Proxy protects the availability of your serverless application by blocking unserviceable application connections that may affect database performance.

4.15 Cloud security:

The services that implement the security of the cloud architecture that we are going to exploit are Cognito, the Virtual Private Clouds (VPC) and the Secret Manager. Another key element for the development of cloud architectures provided by Amazon is AWS CloudFormation which allows you to implement the "Architecture as code" development paradigm. Through this tool you can configure and launch entire cloud architectures only through code without having to manually design every single element of the architecture. For reasons of versatility, however, we have decided to exploit a tool similar to CloudFormation but external to Amazon called Terraform which is able, starting from an already built architecture, to obtain the corresponding code ready to be redistributed.

4.15.1 *Authentication and Cognito configuration:*



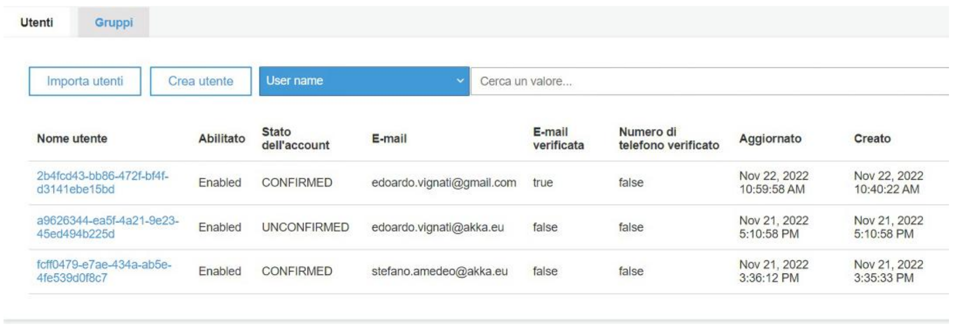
Amazon Cognito
Use a CIAM service that scales to millions of users
Gather user login and custom attributes
Verify and save registration data
Add sign-up
Use frictionless self-registration with a customer user repository (user pool)
Ensure local, adaptive authentication
Federate through a third-party login
Add sign-in
Deploy user authentication with a hosted UI and SDK support
Activate federated access (OAuth2, OIDC)
Activate credential broker access (identity pool)
Control access
Enforce policies and access application resources
Application Load Balancer
AWS API Gateway
Amazon S3
Amazon DynamoDB
Other AWS services
Application resources
Activate role- or attribute-based access controls

Image 4.13: Cognito has everything you need to manage the authentication flow

Cognito is the AWS service designed to enable the creation of services with authentication, authorization and secure user management. After completing the authentication process Cognito provides the user with a token. This token will be necessary for the user to interact with Cognito, that, once received the token back, will provide access credentials to the requested AWS services. This process takes place thanks to the presence of two data pools saved in Cognito: the first is called "user pool", the second is called "federated identity". Amazon Cognito User Pools allow you to quickly create the directory for managing user registration, access, and retention. The "user pool" provides a default interface for registration but also allows you to use a custom one external to the service with the necessary development adjustments. Through the "user pools" we will also be able to easily integrate our application with social intent providers such as Google, Amazon and Facebook or with corporate identity providers such as Microsoft Active Directory via SAML. You will also be able to choose from various security mechanisms such as multi-factor authentication (MFA) which allows for checking for compromised credentials, account theft protection, and email and phone number verification. You can also set up custom authentication flows and manage user migration taking advantage of the AWS lambda triggers provided within Cognito service. Cognito's “Federated Identities” allow you to manage access control to resources. Thanks to this Cognito feature, we will be able to ensure that registered users of the service can only access the AWS resources and *APIs* specified by limiting users' access to the services assigned to them. The accesses provided to different users can be differentiated by assigning different roles and authorizations (even only temporary).

4.15.2 Cognito from theory to practice solutions:

Now that we have clear the options provided by Cognito we will analyze them from the practical side. The image below shows a pool containing 3 users.



Nome utente	Abilitato	Stato dell'account	E-mail	E-mail verificata	Numero di telefono verificato	Aggiornato	Creato
2b4fcd43-bb86-472f-bf4f-d3141ebe15bd	Enabled	CONFIRMED	edoardo.vignati@gmail.com	true	false	Nov 22, 2022 10:59:58 AM	Nov 22, 2022 10:40:22 AM
a9626344-ea5f-4a21-9e23-45ed494b225d	Enabled	UNCONFIRMED	edoardo.vignati@akka.eu	false	false	Nov 21, 2022 5:10:58 PM	Nov 21, 2022 5:10:58 PM
fcff0479-e7ae-434a-ab5e-4fe539d0f8c7	Enabled	CONFIRMED	stefano.amedeo@akka.eu	false	false	Nov 21, 2022 3:36:12 PM	Nov 21, 2022 3:35:33 PM

Image 4.14: A sample of registered users in Amazon Cognito panel

As we can see the default fields of Cognito are the “username”, the “enabled” field, the “account status”, the “email” field, the “verified phone number”, the “updated ” and the “created” one. We will analyze their meaning below:

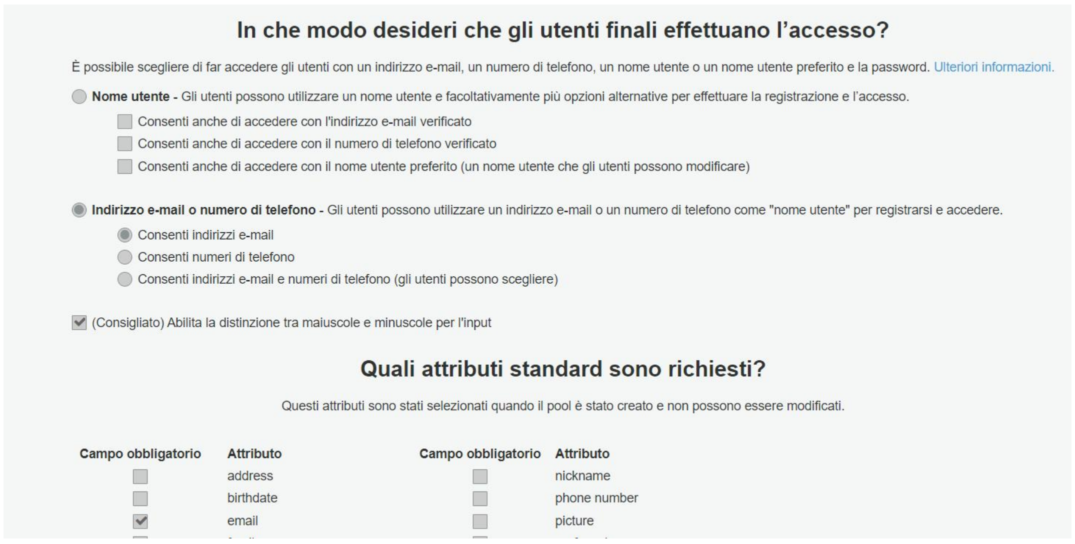
- ✓ *The "username" is an alphanumeric field used by the Cognito functions to identify the user during his requests to the* *authentication service, it can be said that together with the "email" is the main field.*

- ✓ *The enabled field shows, how the word says, if the user has the permission of the Cognito administrators to access* AWS *resources, at any time an administrator can decide to disable a user and even later to delete him.*

- ✓ *The "Account status" field shows if the user has confirmed the registration. Confirmation of registration can be done by email or telephone number. In our case we decided to proceed via email, in fact the "Verified Email" field shows "true" as a result while the "Verified phone number" field shows "false" to indicate that the phone number is not necessarily correct since it has not been used in the process of verification of the account.*

- ✓ *The "email" field contains the user's email and the "Updated" and "Created" fields are self-explanatory and indicate the respective dates of update and creation of the user.*

- ✓ *The second interface of the "user pool" that we are going to analyze is called "Attributes", and here we will be able to select which fields our user can register with, in our case we have chosen email and password. In fact, the only mandatory field (excluding the password) is the email* as shown in the image*. In this same interface but not present in the photo, it is also possible to introduce customized fields not set up by Cognito to complete the registration.*



In che modo desideri che gli utenti finali effettuano l'accesso?

È possibile scegliere di far accedere gli utenti con un indirizzo e-mail, un numero di telefono, un nome utente o un nome utente preferito e la password. Ulteriori informazioni.

○ **Nome utente -** Gli utenti possono utilizzare un nome utente e facoltativamente più opzioni alternative per effettuare la registrazione e l'accesso.

- ☐ Consenti anche di accedere con l'indirizzo e-mail verificato
- ☐ Consenti anche di accedere con il numero di telefono verificato
- ☐ Consenti anche di accedere con il nome utente preferito (un nome utente che gli utenti possono modificare)

● **Indirizzo e-mail o numero di telefono -** Gli utenti possono utilizzare un indirizzo e-mail o un numero di telefono come "nome utente" per registrarsi e accedere.

- ● Consenti indirizzi e-mail
- ○ Consenti numeri di telefono
- ○ Consenti indirizzi e-mail e numeri di telefono (gli utenti possono scegliere)

☑ (Consigliato) Abilita la distinzione tra maiuscole e minuscole per l'input

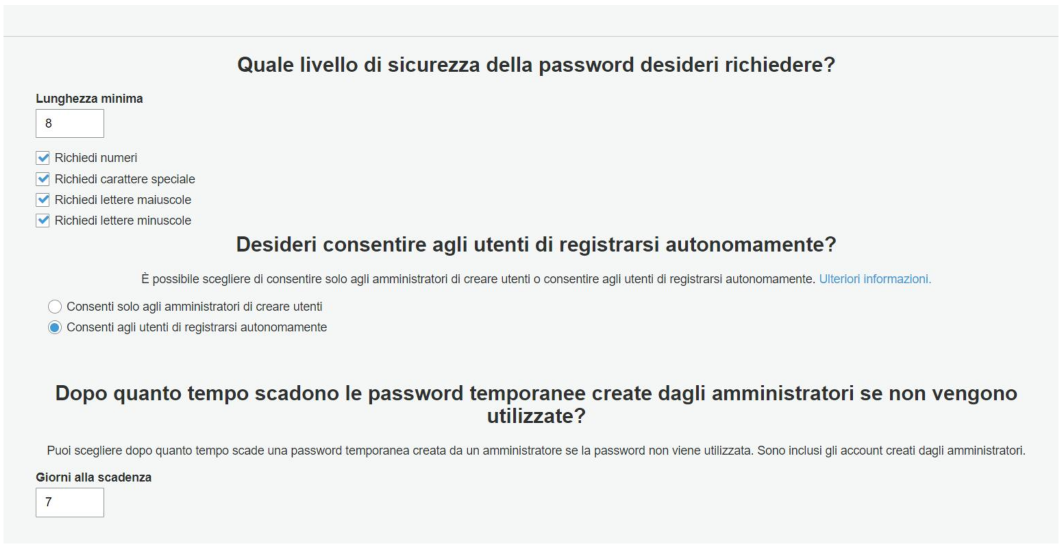
Quali attributi standard sono richiesti?

Questi attributi sono stati selezionati quando il pool è stato creato e non possono essere modificati.

Campo obbligatorio	Attributo	Campo obbligatorio	Attributo
☐	address	☐	nickname
☐	birthdate	☐	phone number
☑	email	☐	picture

Image 4.15: Cognito access method panel

After that, the password settings can be set in the Policy screen. The main element of this interface is the password security level.



Quale livello di sicurezza della password desideri richiedere?

Lunghezza minima: 8

- ☑ Richiedi numeri
- ☑ Richiedi carattere speciale
- ☑ Richiedi lettere maiuscole
- ☑ Richiedi lettere minuscole

Desideri consentire agli utenti di registrarsi autonomamente?

È possibile scegliere di consentire solo agli amministratori di creare utenti o consentire agli utenti di registrarsi autonomamente. Ulteriori informazioni.

- ○ Consenti solo agli amministratori di creare utenti
- ● Consenti agli utenti di registrarsi autonomamente

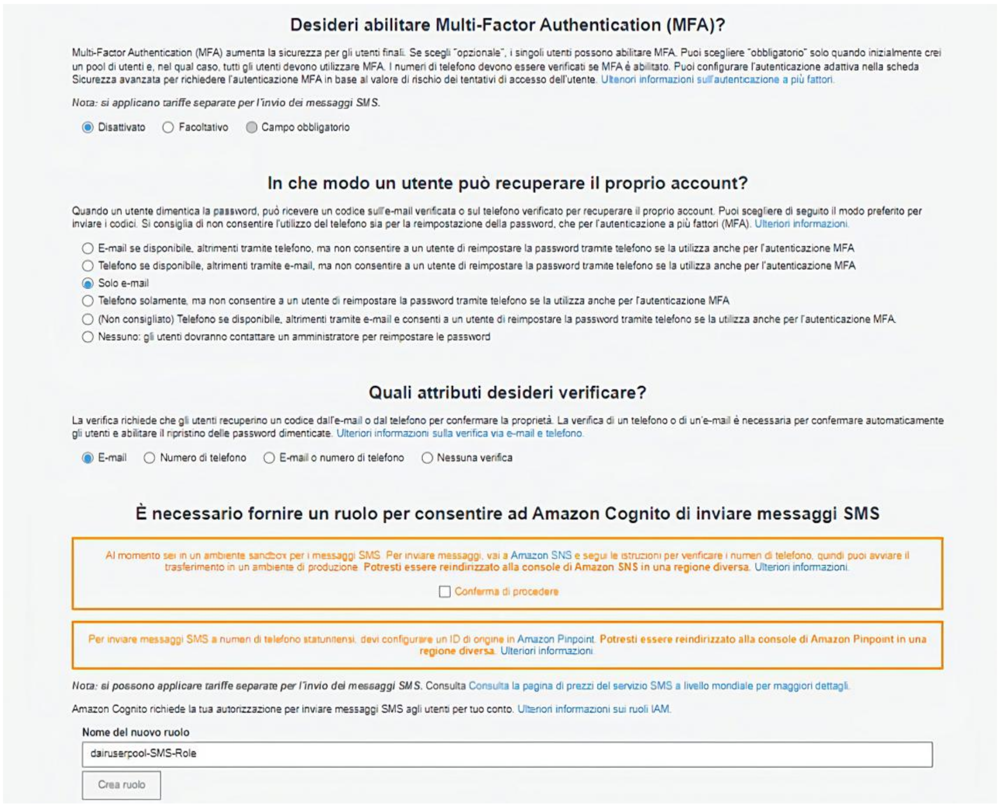
Dopo quanto tempo scadono le password temporanee create dagli amministratori se non vengono utilizzate?

Puoi scegliere dopo quanto tempo scade una password temporanea creata da un amministratore se la password non viene utilizzata. Sono inclusi gli account creati dagli amministratori.

Giorni alla scadenza: 7

Image 4.16: Continuation of the preceding panel

The MFA and verification interface allows you to set up multi-factor authentication, currently disabled in our case, and the password recovery method (we chose via email).


Desideri abilitare Multi-Factor Authentication (MFA)?

Multi-Factor Authentication (MFA) aumenta la sicurezza per gli utenti finali. Se scegli "opzionale", i singoli utenti possono abilitare MFA. Puoi scegliere "obbligatorio" solo quando inizialmente crei un pool di utenti e, nel qual caso, tutti gli utenti devono utilizzare MFA. I numeri di telefono devono essere verificati se MFA è abilitato. Puoi configurare l'autenticazione adattiva nella scheda Sicurezza avanzata per richiedere l'autenticazione MFA in base al valore di rischio dei tentativi di accesso dell'utente. Ulteriori informazioni sull'autenticazione a più fattori.

Nota: si applicano tariffe separate per l'invio dei messaggi SMS.

[✓ Disattivato] Facoltativo Campo obbligatorio

In che modo un utente può recuperare il proprio account?

Quando un utente dimentica la password, può ricevere un codice sull'e-mail verificata o sul telefono verificato per recuperare il proprio account. Puoi scegliere di seguito il modo preferito per inviare i codici. Si consiglia di non consentire l'utilizzo del telefono sia per la reimpostazione della password, che per l'autenticazione a più fattori (MFA). Ulteriori informazioni.

- E-mail se disponibile, altrimenti tramite telefono, ma non consentire a un utente di reimpostare la password tramite telefono se la utilizza anche per l'autenticazione MFA
- Telefono se disponibile, altrimenti tramite e-mail, ma non consentire a un utente di reimpostare la password tramite telefono se la utilizza anche per l'autenticazione MFA
- [✓ Solo e-mail]
- Telefono solamente, ma non consentire a un utente di reimpostare la password tramite telefono se la utilizza anche per l'autenticazione MFA
- (Non consigliato) Telefono se disponibile, altrimenti tramite e-mail e consenti a un utente di reimpostare la password tramite telefono se la utilizza anche per l'autenticazione MFA.
- Nessuno: gli utenti dovranno contattare un amministratore per reimpostare le password

Quali attributi desideri verificare?

La verifica richiede che gli utenti recuperino un codice dall'e-mail o dal telefono per confermare la proprietà. La verifica di un telefono o di un'e-mail è necessaria per confermare automaticamente gli utenti e abilitare il ripristino delle password dimenticate. Ulteriori informazioni sulla verifica via e-mail e telefono.

[✓ E-mail] Numero di telefono E-mail o numero di telefono Nessuna verifica

È necessario fornire un ruolo per consentire ad Amazon Cognito di inviare messaggi SMS

Al momento sei in un ambiente sandbox per i messaggi SMS. Per inviare messaggi, vai a Amazon SNS e segui le istruzioni per verificare i numeri di telefono, quindi puoi avviare il trasferimento in un ambiente di produzione. **Potresti essere reindirizzato alla console di Amazon SNS in una regione diversa.** Ulteriori informazioni.

☐ Conferma di procedere

Per inviare messaggi SMS a numeri di telefono statunitensi, devi configurare un ID di origine in Amazon Pinpoint. **Potresti essere reindirizzato alla console di Amazon Pinpoint in una regione diversa.** Ulteriori informazioni.

Nota: si possono applicare tariffe separate per l'invio dei messaggi SMS. Consulta Consulta la pagina di prezzi del servizio SMS a livello mondiale per maggiori dettagli.

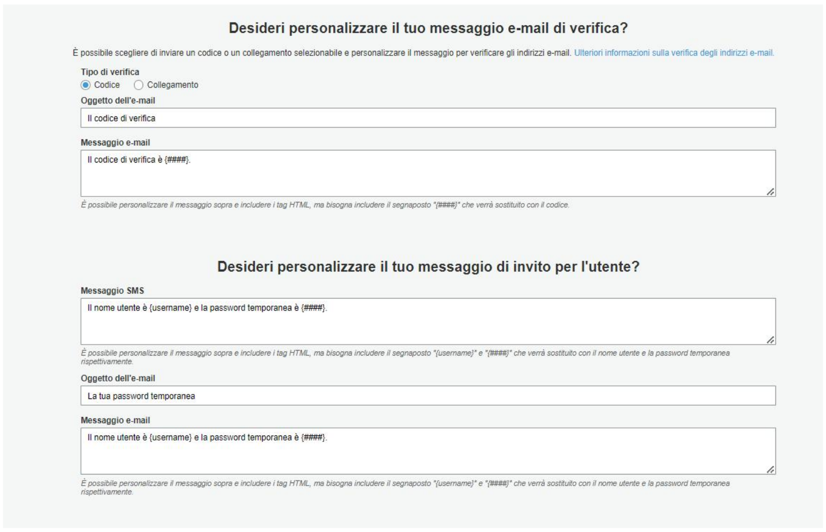
Amazon Cognito richiede la tua autorizzazione per inviare messaggi SMS agli utenti per tuo conto. Ulteriori informazioni sui ruoli IAM.

Nome del nuovo ruolo

dairuserpool-SMS-Role

Image 4.17: Multi-factor authentication panel

On the "Message Customizations" page we have chosen to verify the user's account via a verification code. The message can also be customized according to your choice.


Desideri personalizzare il tuo messaggio e-mail di verifica?

È possibile scegliere di inviare un codice o un collegamento selezionabile e personalizzare il messaggio per verificare gli indirizzi e-mail. Ulteriori informazioni sulla verifica degli indirizzi e-mail.

Tipo di verifica
[✓ Codice] Collegamento

Oggetto dell'e-mail
Il codice di verifica

Messaggio e-mail
Il codice di verifica è {####}.

È possibile personalizzare il messaggio sopra e includere i tag HTML, ma bisogna includere il segnaposto "{####}" che verrà sostituito con il codice.

Desideri personalizzare il tuo messaggio di invito per l'utente?

Messaggio SMS
Il nome utente è {username} e la password temporanea è {####}.

È possibile personalizzare il messaggio sopra e includere i tag HTML, ma bisogna includere il segnaposto "{username}" e "{####}" che verrà sostituito con il nome utente e la password temporanea rispettivamente.

Oggetto dell'e-mail
La tua password temporanea

Messaggio e-mail
Il nome utente è {username} e la password temporanea è {####}.

È possibile personalizzare il messaggio sopra e includere i tag HTML, ma bisogna includere il segnaposto "{username}" e "{####}" che verrà sostituito con il nome utente e la password temporanea rispettivamente.

Image 4.18: In this panel you personalize the verification code

In the "App Customers" interface, through a code, we will be able to provide access to the pool of users, only to those who provide the same code in their requests. In the "Trigger" interface, you can customize Cognito workflows by associating Lambda functions that execute custom code to certain Cognito events. Through the "App Integration" interface it will be possible to choose the settings to generate a web page provided by Cognito to perform authentication. In our case we do not use the one provided by Cognito because we prefer to create the authentication web page separately from the Cognito service. The last useful page is the "Federated identities" which allows authentication to the service through the use of accounts federated with AWS such as Facebook, Google and others (but also in this case at the moment our architecture does not use them).

4.16.3 Adding Cognito to the rest of the architecture:

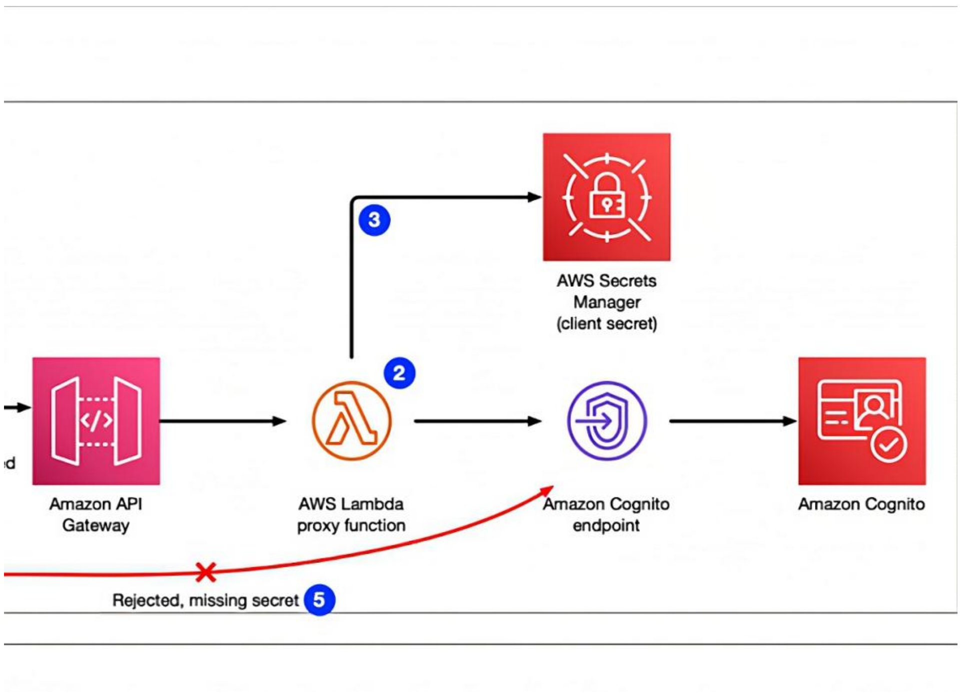


Image 4.19: Adding Cognito to the architecture

As with the procedure explained for the RDS Proxy, Cognito also makes use of Lambdas in our architecture. In fact, to access the Cognito service there are 2 previous steps that the data flow arriving from the client must overcome to reach its destination. Starting from the user's app, the data flow sends a request to the API Gateway via a series of REST-type Endpoints. Depending on the Endpoint to which the app makes the request, the API Gateway associates a Lambda function especially created to perform a specific function. The Lambda functions available for authentication are currently seven and include registration, confirmation of registration via email, login, sending email messages for account confirmation or password recovery. Most of the Lambdas in question have permissions (in AWS called roles) to access the Secret Manager, the proxy (and the MySQL database) and Cognito. Lambda must access the Secret Manager to obtain the database access credentials, the proxy to save some data on the database and Cognito for everything related to authentication and user access. The current code is written completely in node.js, but Lambda functions are a very flexible service and allow to be written in many different languages. This is a huge advantage because, since we work in a microservices architecture (each lambda can be modified in a completely separate way from the rest of the resources) if a new development team takes over and want to add new features to the service, they can leave the old microservices in their own language and write new lambda functions in other languages. Another benefit is the testing of assets and their modification. Since each lambda is built to be independent from the others, each element of the development team can concentrate on an element without having to wait for the conclusion of the work of the other developers (obviously by previously agreeing on the methodologies in case the Lambdas in question need to access common resources or communicate with each other).

4.16 S3

Image 4.20: Amazon S3 is a really elastic data container service

Another important component to our architecture is Amazon S3. Amazon S3 is able to simplify the management of large amounts of data in a scalable and cost-effective manner in comparison to any other traditional storage solution. Amazon S3 is a cloud computing service that provides scalable object storage with no quantity limits and 99% data durability. This astonishing percentage is respected due to the separation of data centers into several "availability zones" inside various "regions" of the AWS service. Our files are defined as objects in Amazon S3 and are saved within containers named buckets (they are like folders inside our Amazon S3 account). Because bucket can be reached through web there aren't two buckets with the same name. On Amazon S3, an object is a collection of data and metadata that is uniquely identified by a name and, if the "multiple version" feature is enabled by a name and a version id. A key in Amazon S3 is a string that uniquely identifies an object within a bucket and corresponds to the object's name or path. Amazon S3 is a really versatile service and let you replicate your data over different availability zones and regions (for safety and latency reasons). The multi-region feature happens only if the user requests it. This is due to different state legislation about data contained into servers. Another feature of Amazon S3 is letting the user keep different versions of a file (only if the option is enabled). This is done to be able to recover an old version of the data. There are also different Amazon S3 class of storage, characterized mainly by their trade-off cost and data access rapidity.

4.17 Infrastructure as Code

With the development of the cloud the development of the infrastructure of a project moved for the developer from the physical necessity to buy and configure a real server to the creation of a virtual environment in cloud. Over the time more complex virtual structures have born to let the developers follow new philosophies like DevOps or to build complex systems. With the increasing complexity of structures the ability to code them became a necessity. The main advantage of working with paradigm infrastructure as code is the ability to provide faster, repeatable and automatic provisioning. More in detail the advantages that could be implemented using Infrastructure as code are:

- ✓ ***It improves the speed of the work**: automation is faster than manually editing the interface when you need to implement and connect resources.*
- ✓ ***It increase the reliability**: if your infrastructure is big, it is easy to wrongly write the configuration of a resource or to execute the provisioning of the services in an incorrect order, with infrastructure as code provisioning and configuration of resources is always exactly as declared.*
- ✓ ***It prevents differences in the configuration**: a difference in the configuration can be caused by a mutable structure of the project due to new updates, if you create a different version of each updated function and, in this way you make "immutable" configuration of each update, letting it be recovered if some errors come with the new update.*
- ✓ ***It helps supporting experimentation, test execution and optimization**: because of infrastructure as code is simpler and faster then the manual creation of the single resources you recover time to make new experimentation with the infrastructure, testing it and rapidly expands the project for the production.*

Actually Infrastructure as code is considered a fundamental part of Agile development and DevOps practices.

4.17.1 CloudFormation

CloudFormation is a service provided by AWS that allows you to build your AWS infrastructure using code. CloudFormation was released in 2011 and has become indispensable for many AWS customers as it allows you to define reusable templates to generate AWS resources. Using CloudFormation you save time in building the infrastructure and you can focus more on other operations such as development. CloudFormation allows you to generate AWS infrastructures simply by using a JSON or YAML file in which you write the code of the resources to be created. To modify a CloudFormation model we can use the designer provided by the service itself which has a rich graphical interface and has nine main fields:

1. *Version identifies the capabilities of the model and if not specified CloudFormation assumes it is the latest version available.*
2. *The description which helps us include any comments regarding the model.*
3. *The metadata field which includes the details and resources used by the model.*
4. *The parameters that we can enter to customize the model for example by altering the input values.*
5. *The mapping that helps us match a key with the corresponding set of values.*
6. *The conditions, in which we can insert indications that indicate when a resource is created or when a property is defined.*
7. *The field is called transformation and specifies one or more transformations that we can use in our model.*
8. *Resources stating which AWS resources you want to include.*
9. *A field that helps sort the CloudFormation console output.*

4.17.2 Terraform

HashiCorp
Terraform

Image 4.21: Terraform let you implement infrastructure as code for most of the cloud providers

Terraform is an open source tool developed to realize the Infrastructure as Code (IaC). It is a declarative codification instrument that let you build the infrastructure of a cloud environment using the HCL language that is able to describe the infrastructure and provision it. Its main advantage is that is an independent platform service. This means that once learned it can be used for every provider of cloud services while the major part of Infrastructure as Code service usually works with only once. Another advantage is that Terraform is open source so it is supported by a large community that create plugins for the platform and as a result Terraform expands rapidly.

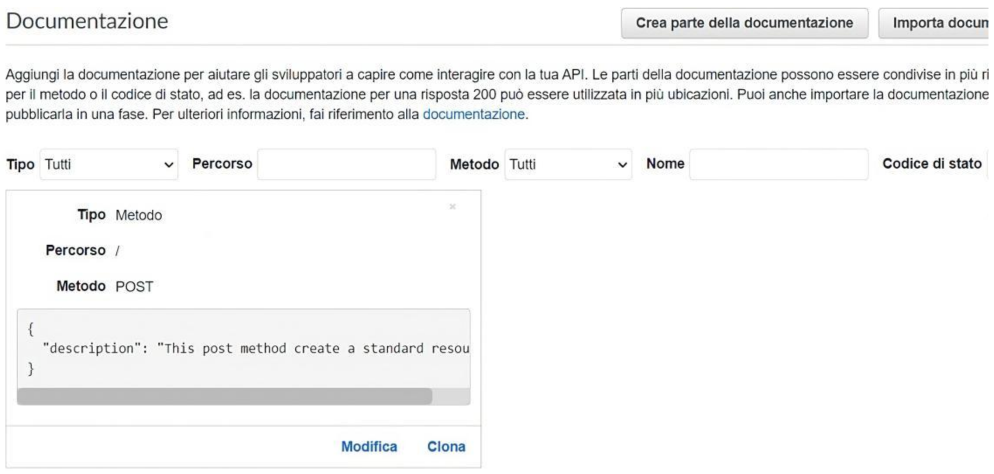
4.17.3 Working principle of Terraform

The base element of a Terraform infrastructure as code configuration is the “module”. A module can contain one or more components of the infrastructure. Each module can be called independently, recalled, and can call other modules, in this case called “secondary modules” to make the development process more granular. As we said earlier Terraform uses plugins that implement different resource types of a specific provider. The plugins contain all the necessary code to authenticate and connect to a service, the most part of cloud services are included, and AWS is surely one of them. Terraform can automatize the management of Infrastructure as a Service, Platform as a Service, Software as a Service and Function as a Service cloud providers infrastructures.

4.18 <u>Swagger vs API Gateway documentation</u>

Image 4.22: Swagger let you describe the structure of your API

The process of creating documentation for an application is long and tiring, but particularly necessary when the development team is replaced and new developers come into contact with the application. In this situation clear and complete documentation is essential whether you need to complete the project or just want to make changes or maintenance. APIs (Application Programming Interfaces) represent the contact point between applications. All applications connected to the Internet rely on APIs that call specific application functions and return a useful result to the requester. As with the other components of the application, the APIs must also be documented and a standard called OpenAPI Swagger has been designed to achieve this purpose. In this way we can standardize the documentation process even for those who have not participated in the development process from the beginning. Before the OpenAPI standard, due to the different methodologies, technologies and programming languages, there was no standard for creating APIs. REST was later chosen as the standard for developing RESTful APIs. At the release of OpenAPI Swagger, the main REST API description competitor was WSDL 2.0 which, however, was considered rather complex, and for this reason quickly surpassed in popularity by the new competitor. The documentation created with Swagger consists of a single text file in JSON or YAML format. With the Swagger interface you can view the documentation in both textual and graphical form and you can also send requests to the API, which is why it has achieved great success. Swagger has several tools bundled and available online under the name SwaggerHub that facilitate documentation development. The first tool analyzed is Swagger Inspector which only by calling the server is able to generate part of the documentation which can then be edited with Swagger UI. Once completed, the documentation can finally be hosted on SwaggerHub, which however in the free version does not allow sharing projects with other users or even integrating with AWS for the automatic generation of documentation in case of editing of the API Gateway calls, moreover allows you to host up to 3 sheets of documentation.


Documentazione

Crea parte della documentazione | Importa docun

Aggiungi la documentazione per aiutare gli sviluppatori a capire come interagire con la tua API. Le parti della documentazione possono essere condivise in più ri per il metodo o il codice di stato, ad es. la documentazione per una risposta 200 può essere utilizzata in più ubicazioni. Puoi anche importare la documentazione pubblicarla in una fase. Per ulteriori informazioni, fai riferimento alla documentazione.

Tipo Tutti | Percorso | Metodo Tutti | Nome | Codice di stato

Tipo Metodo
Percorso /
Metodo POST

```
{
  "description": "This post method create a standard resou
}
```

Modifica | Clona

Image 4.23: API Gateway documentation panel

Taking into account the importance of documenting the APIs, AWS has provided the API Gateway service with a panel called "documentation" in which we can analyze or create the documentation related to each API call. For simplicity reasons we can also choose to write the documentation during the creation of the API Gateway and its APIs by clicking on the transparent gray book-shaped icons. The benefit of documenting with API Gateway is that you can leave notes for other developers and your work is instantly visible to everyone. After deploying the API Gateway you will be able to publish the documentation. In this case, publishing means generating a document in OpenAPI 2.0 or 3.0 format and exporting it in JSON or YAML format. It will contain all the documentation related to the API Gateway that respects the chosen format, in fact any comments that do not follow the OpenAPI rules will not be exported. In conclusion, the documentation task can be done with both API Gateway and SwaggerHub, the choice is whether you prefer the sharing allowed by API Gateway or the simplicity provided by SwaggerHub. After that the hosting can take place both on an S3 bucket or using the free SwaggerHub service.

Chapter 5

5 Agile development

During the development of an application, good management of the group is essential to optimize times and results. To accompany the different strategies, at the base, there must be technologies that are adequate for the job we want to do. One of the technologies used in many group projects is ***Git***, a system that allows version control and the collaboration of multiple developers on the same project. There are various platforms online such as GitHub, GitLab and Bitbucket that allow you to use the Git system to manage your projects. Git allows you to maintain multiple active versions of your project, each of which is called a “branch”. Each “branch” can be modified separately and then also reunited with the main “branch” at a later time through a “merge” operation. Every time, before changes are applied to a “branch”, a “commit” containing the changes is made. The conflicts between the new and the old version are resolved, after which the upload of the data to the repository is confirmed with a “push” operation. The “fetch” operation instead allows you to check if changes have been made to the “branch” (generally to check if other elements of your team have modified the repository), if so, before continuing to work on the project it will be necessary to download the new version with a “pull” operation, resolve any conflicts with our local work and then reload everything with a "push" operation. This is the basic principle of Git. Over time Git has gained more and more fame and new usage philosophies supported by libraries have been added. One of these is ***Gitflow***, initially a Git usage model, after adapted to the state of the art of modern DevOps strategies. Then it was supported by an installable toolset that simplifies the set of Git commands to perform the actions required by Gitflow (it includes those which would be a series of git commands in individual functions). Currently Gitflow is no longer considered the state of the art according to DevOps strategies, it has in fact been supplanted by trunk-based workflows, however for small development teams that do not need continuous merges it is still considered an optimal technology. So let's analyze how Gitflow works: Gitflow uses two branches, the first called “main” is the one in which there is the version of the software in distribution, the second called “develop” is the one in which tests and changes to the application are performed before being merged with the “main” branch (only if they are fully functional). Every time you want to make a change to the code, a new branch cloned from the “develop” branch is opened. In this branch we will make all the necessary changes (many new “feature branches can be opened in parallel”), at the end of which, the “feature branch” is merged with the “develop branch”. When we have made enough changes and we are satisfied with the result we can create a “release” branch, during the period in which the “release” branch is active, no new features can be added to the software and we only deal with testing and bug fixing, once satisfied with the result, you can also merge the “develop” branch with the “main” branch by putting the new changes into distribution. If an error is found in the “main” branch, an “hotfix” branch must be opened, used for a bug resolution sprint, at the end of which the “branch” is closed both on the “main” branch and on the “develop” branch and “main” must be tagged with a new version number.

5.1 Trunk-based development

Another alternative to Gitflow strategy, considered actually the state of art, as well as used by Google developers, is Trunk-based development. Actually Trunk-based development is being treated as the most advanced teamwork developer enforcement strategy. Trunk-based development (TBD) is a branching technique in which all developers integrate their changes every day directly to a shared trunk, which is always in a releasable form. Whatever a developer does on their local repository, they must integrate their code at least once every day. This method requires each developer to observe and react to changes made by their coworkers in version control on a regular basis, making collaboration on the quality and status of the codebase a near-constant activity. TBD permits the usage of additional branches, such as a short-lived release branch off the trunk for executing a release and local-only feature branches, although neither is necessary for TBD practice. TBD is the most closely aligned with current delivery methodologies like as Continuous Delivery and DevOps, which have becoming more important and even required for many software development teams. Surely, Trunk-based development is required for CI/CD. TBD prioritizes continuous integration in a developer's workflow. You simply must incorporate your modifications every day; comprehending this immediately causes you to reconsider how you approach your work and collaborate with your team members. You must approach your task as a succession of modest moves ahead, with each step potentially being launched to production. Smaller adjustments reduce the blast radius of any changes that cause a problem in production. As you gain experience using branch by abstraction, you'll learn that you occasionally want your code modifications to be deployed. For example, you may want your modifications to be verified in a non-production environment but not yet visible in production. Using feature flags, you may connect your branch to an externally controlled mechanism, such as an application configuration file or an external database or service. This provides you more control over releasing changes outside of the codebase. Using feature flags, developers may release incomplete work to production while keeping it hidden from end users. Since our team was modest in number we actually don't need to operate using trunk-based development in fact Gitflow, with little sized teams, perform as well as Trunk-based Development.

5.2 Deployment strategy

To create a full agile development environment we have just moved the first steps. Using Gitflow we were able to coordinate the work of the developers of the team. Now ***we need a deployment strategy that mechanically get the updates uploaded on Git and deploy them to AWS***. This is important to ***avoid deployment errors***, in fact when a project has a large dimension and the development process is complex and require many deployment, it is easy that during the deployment process some errors are made. So the developers will need to lose time fixing also deployment errors, and this is the part we can avoid using a good deployment strategy and the correct technologies. The three Amazon AWS services we will analyze for this purpose are Codebuild, Codedeploy, CodePipeline.

5.3 CodeBuild

Amazon CodeBuild is a fully managed continuous integration service that allows you to compile source code, run tests, and generate ready-to-implement software packages. CodeBuild eliminates the need to manage, scale and provision your development servers. It is necessary to specify the code's flow and build settings. CodeBuild will then run the build script to compile and test the code as well as create packages. CodeBuild automates continuous integration and delivery pipelines (CI/CD), resulting in a fully automated software release process that propagates changes to the code to many implementation environments. CodeBuild reduce the complexity of managing development servers, in fact it execute the Jenkins compilation tasks already present on CodeBuild to remove the need to configure and manage the Jenkins compilation nodes. CodeBuild automatically executes software builds using an existing GitHub repository and publishes the new results on GitHub. CodeBuild can also conduct unit tests on source code. Java, Ruby, Python, Go, Node.js, Android, and Docker are among the programming languages and frameworks supported by CodeBuild. AWS CodeBuild fixes and maintains server builds automatically, and it scales as volume grows. To offer a new isolated environment for each task, the service constructs temporary compute containers for server builds. It also runs many builds at the same time. When the build is complete, AWS CodeBuild discards containers and uploads build artifacts to S3 buckets or other storage sites. AWS CodeBuild interfaces with other AWS code services, such as AWS CodePipeline. A developer can use the AWS Key Management Service to encrypt build artifacts. AWS

CodeBuild works with the AWS Management Console, AWS CLI, software development kits, and application programming interfaces to provide specific information about each build, such as start and end times, status, commit ID, and branch.

5.4 CodeDeploy

The third element we need to create a continuous deployment environment is CodeDeploy. AWS CodeDeploy is a service that deploys application code from AWS S3, GitHub, or BitBucket to EC2 or on-premises instances. With hybrid infrastructure becoming the standard for many big projects, this is a capability that a cloud deployment tool must have. Continuous Integration (CI) and continuous delivery are two fundamental procedures in DevOps. Continuous builds and tests are insufficient to complete your DevOps shift, you must also deploy continually. The practice of delivering an app in short cycles, up to numerous times per day, is known as continuous delivery. This necessitates the development and testing of code such that it is release-ready from the outset. CodeDeploy allows you to deploy your code in two ways:

- ✓ *Deployment in-place: CodeDeploy delivers your code to the same set of EC2 instances by pulling the instances offline, executing the scripts that deploy your code, and then bringing the instances back online. This approach requires downtime and should be scheduled but, however, employs less EC2 instances than the other technique.*

- ✓ *Blue-green deployment: it entails developing two similar production setups that can manage equal production loads. While one environment in the blue environment is running the live application, the release is pushed to the other green environment to be setup and tested. Once the green environment is stable, a switch in the elastic load-balancer is* *all that is required to route traffic from blue to green. This approach consumes more EC2 resources, but it reduces downtime. This is the recommended technique for mission-critical applications. When the deployment on the green environment is complete, you can destroy the blue environment until your next release is ready. CodeDeploy can automate the entire procedure.*

Before you can begin the deployment, you must create an IAM user and guarantee that all of the components like repositories, EC2 instances, and CodeDeploy, can communicate with each other. You can deploy your app using CodeDeploy in two ways: through the CodeDeploy interface or using the AWS CLI.

5.5 CodePipeline

CodePipeline is a fully managed continuous delivery solution that aids in the automation of release pipelines for quick and dependable application and infrastructure updates. CodePipeline automates the build, test, and deploy parts of the release process whenever there is a code change. This allows for the quick and consistent release of features and upgrades. AWS CodePipeline integrates easily with third-party services like GitHub or any other custom plugin. CodePipeline works good also with other AWS tools like the three discussed before. In fact CodePipeline is like a junction that can connect all services together creating the real continuous deployment system. Continuous deployment is a software engineering technique that uses automated deployment to offer product functionality. It assists testers in determining whether or not the codebase modifications are proper and stable. By depending on infrastructure that automates several testing procedures, the team may accomplish continuous deployment. The program is updated with a new code after each integration satisfies the release requirements.

5.6 Scrum

Scrum is a management methodology that teams use to self-organize and work toward a common goal. We tried to apply Scrum strategies to our team management. Here we will explain how it works. Scrum is also built on empiricism, or the idea that knowledge comes from experience and that decisions must be made around our knowledge. ***Transparency, examination, and adaptability*** are the three pillars that support empiricism. The framework's structure is quite simple, consisting of:

1. *Responsability: there are three distinct roles within Scrum. Product Owner, Scrum Master, and Developers. All of them come together to form the Scrum Team, which has the characteristic of being self-managing and cross-functional, which means that it has all of the competencies necessary to deliver a software increment without relying on outside teams or individuals.*

2. *Artifacts refers to the methods through which Scrum visualizes work and value. The artifacts are Product Backlog, Sprint Backlog, and Increment. Every artwork includes a "promise" with the goal of improving transparency and allowing the team's progress to be measured. These are the commitments:*

   - ✓ *Product Goal, which is part of the Product Backlog and describes the product's future state. The Product Goal serves as a long-term goal.*

   - ✓ *Sprint Goal, which is part of the Sprint Backlog; describes a shorter-term goal and the reason why an iteration adds value to the product.*

- ✓ *Definition of Done, which describes when an Increment (or software increment) meets an acceptable quality level, allowing the final release to the user.*

3. *Event and Ceremonies: Scrum's events or ceremonies are as follows:*

    - ✓ *Sprint Planning occurs at the start of the Sprint (or iterate). It is, as the term suggests, a planning event.*
    - ✓ *Daily Scrum is a weekly 15-minute session in which developers update their plans for achieving the Sprint Goal.*
    - ✓ *Sprint Review, which closes a Sprint and allows you to reflect on your work.*
    - ✓ *Sprint Retrospective, the final meeting inside a Sprint with the purpose of evaluating processes, practices, and other aspects related to collaboration.*

In the Scrum framework, like in other agile methods, the entire team is self-organized, and the development team is cross-functional. According to agile and Scrum methodologies, if you have a team of prepared people, it makes sense to use everyone's full potential by sharing responsibilities and leading this group from the outside. The advantage is significant if you believe that this approach is effective in removing the potential bottleneck of having a single person tasked with visioning and making decisions on all aspects of the project. But the most essential result of this way of working is increased involvement and participation from all team members, which leads to an increase in productivity and well-being for all. As a result, in a Scrum team, there is shared leadership, and being a leader entails locating the problem and bringing the appropriate people together to solve it. Anyone on the team who is capable of anticipating an issue has the opportunity to serve as a leader in its resolution. Scrum is organized into Sprint cycles that last from one to four weeks. At the end of each iteration, the team releases one or more increments that include potentially releasable functionality. The cycles are time-boxed, which means they have a fixed duration, cannot be interrupted, and must end even if the work is not completed.

5.6.1 Scrum planning

At the start of each Sprint, at a session called Sprint Planning, the team selects its own tasks from a prioritized list of activities (Product Backlog) and commits to completing all activities that contribute to the achievement of the Sprint Goal by the end of the Sprint. The ultimate goal is not to complete as many activities as possible, but to produce increments of fully functional and usable software through the achievement of the Sprint Goal that is, the short-term goal that is desired throughout the iteration. This goal was agreed upon by the whole Scrum Team and was not predetermined by the Product Owner. It is critical to accept that everything that is planned at the start of the iteration can be changed as the team learns new skills during the development phase. The only conditions are that changes do not have a negative impact on product quality or the Sprint Goal which is fixed for the duration of the cycle.

5.6.2 Scrum execution

The team meets on a daily basis through the Daily Scrum Meeting (also known as the Daily Stand-up). It is common knowledge that during the daily stand-up ideas, problems, and potential dependencies on other teams emerge. This ceremony lasts a maximum of fifteen minutes if carried out correctly. All discussions, including the resolution of any problems or impediments discovered, are relegated to other sessions. At the end of the Sprint, the team releases an increment, which must always be complete and potentially releasable to the ultimate user. For example, in the case of a typical software application, a fully integrated, functional, and tested functionality. It is vital to note that under Scrum, there might be more than one increment every sprint. To put it another way, you don't have to wait until the end of the Sprint to celebrate.

5.7 End of the sprint

The Sprint concludes with two key ceremonies: the Sprint Review and the Retrospective Meeting. The Sprint Review is an informal meeting in which the team discusses the work done in collaboration with the Product Owner and any key stakeholders. The Retrospective Meeting, which is usually held at the end of the Review, allows the team to reflect on the just completed Sprint and serves for everyone to express their views. It’s important to execute this phase of the sprint to analyze what worked during the previous iteration and what needs to be improved.

Conclusions and future development

The project done in collaboration with the company is still being developed, but the most important tasks are already completed. The next updates are some networking architectural components and some Lambda functions to be written to complete the range of service needed to realize all the customer’s requests. The state of art development principles should be the one described in the thesis, but it is possible that the techniques used here are not the best one for every kind of project. In conclusion this thesis could be an introduction to the ones that try entering the cloud serverless world, both in matter of technologies and techniques used during the development.

Bibliography and sitography

[1] Gene Kim, Jez Humble, Patrick Debois, & Jhon Willis, The DevOps Handbook - How to Create World-Class Agility, Reliability, and Security in Technology Organizations.

[2] Chris Richardson, Microservices Patterns.

[3] https://docs.aws.amazon.com/ AWS Documentation.

[4] https://www.bmc.com/blogs/aws-ecs-vs-aws-lambda/ Confrontation between Lambda and ECS

[5] https://www.ionos.it/digitalguide/siti-web/programmazione-del-sito-web/che-cose-swagger/ Swagger

[6] https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflowm Gitflow Atalassian

[7] https://launchdarkly.com/blog/git-branching-strategies-vs-trunk-based-development/ Trunk-Based development

[8] https://www.ibm.com/it-it/cloud/learn/terraform Terraform

[9] https://www.statista.com/statistics/967365/worldwide-cloud-infrastructure-services-market-share-vendor/ Statista

[10] https://iamondemand.com/blog/aws-lambda-vs-azure-functions-ten-major-differences/ Lambda vs Azure Functions

Thanksgiving

On this page, I'd want to thank everyone who helped me with my thesis work and, in general, during my academic career.

In particular, I'd like to thank my supervisor, Luciano Lavagno for always being there to help me in times of difficulty and providing very useful suggestions for the completion of this project.

A heartfelt thank you also to the company Akka Technologies and to project managers Stefano Amedeo, Ippolita Jarenko, Rocco Affinito, and my coworker, Edoardo Vignati, who has always been available to assist me.

I want to thank my family for being there for me at all times, for assisting me at difficult times, and for asking me to do things that i would not have done otherwise; i want to thank them for accepting and supporting all of my decisions and projects.

Finally, I'd want to thank everyone who has played a role in this journey and in my life. Among them, a special thanks to Mattia and Edo for always be present, to my historical roommates Carlo, Simone, Riccardo with whom I shared almost my whole university career. I want to thank Sere and Matteo for the many coffee breaks, and my Turin adventure's comapanions Cristiano, Marco and Marco.