Integration of business systems optimized with Cloud

A Degree Thesis
Submitted to the Faculty of the
Escola Técnica Superior d'Enginyeria de
Telecomunicació de
Barcelona
Universitat Politécnica de Catalunya
by
Sergi Roura Saez

In partial fulfilment
of the requirements for the degree in
TELEMATIC ENGINEERING

Advisor: Jose Luís Muñoz

Barcelona, July 2018

Abstract

The purpose of this project is the analysis and realization of proofs of concept of several solutions for communication's optimization between systems, based on APIs, to determine which scenarios offer the most benefits in the business ambit.

To attain this purpose, several studies for the different technologies have been done by using them on the development of a couple of web applications, to then compare and test their behaviors.

The technologies that have been studied are Webhooks, HTTP Compression, Reactive APIs, Server-sent Events, GraphQl, Kafka Streams, Tyk API Gateway, Kong API Gateway and iPaaS.

We have seen that all of them have their advantages and disadvantages. Some like HTTP Compression should be used as often as possible, while others like GraphQL or Server-sent Events have more specific use cases. There are also those which have similar behaviors such as Tyk or Kong where we have to choose which of them use depending on our expectations, while others like Reactive APIs and Webhooks complement each other.

At the end, deciding what to use and what not to depends on the systems we want to integrate. The solutions studied in this document have all proven to be useful to solve specific problems.

Resum

L'objectiu d'aquest projecte es l'anàlisis i realització de probes de concepte de diverses solucions per la optimització de comunicació entre sistemes, basats en APIs, per tal de determinar quins escenaris ofereixen els més grans beneficis en l'àmbit empresarial.

Per aconseguir cumplir aquest objectiu, s'han fet diversos estudies per les diferents tecnologíes usant-les en el desenvolupament d'un parell d'aplicacions web, per comparar i testejar el seu comportament.

Les tecnologies que s'han estudiat són Webhooks, HTTP Compression, Reactive APIs, Server-sent Events, GraphQL, Kafka Streams, Tyk API Gateway, Kong API Gateway i iPaaS.

Hem pogut veure que totes elles tenen els seus avantatges i desavantatges. Algunes com HTTP Compression s'haurien de fer servir tant sovint com sigui possible, mentre que d'altres com GraphQL o Server-Sent Events tenen casos d'ús més específics. Hi han també aquelles que tenen comportaments similars, com Tyk i Kong en les que hem de triar la que fer servir depenent de les nostres expectatives, mentre que d'altres com Reactive APIs i Webhooks es complementen entre elles.

Al final, decidir què fer servir i què no depèn dels sistemes que volem integrar. Les solucions estudiades en aquest document han demostrat ser útils per solucionar problemes específics.

Resumen

El objetivo de este proyecto es el análisis i realización de pruebas de concepto de diversas soluciones para la optimización de comunicaciones entre sistemas, basados en APIs, con tal de determinar qué escenarios ofrecen los mayores beneficion en el ámbito empresarial.

Para conseguir cumplir este objetivo, se han hecho diversos estudios para las diferentes tecnologías usándolas en el desarrollo de un par de aplicaciones web, para comparar y testear su comportamiento.

Las tecnologías que se han estudiado son Webhooks, HTTP Compression, Reactive APIs, Server-sent Events, GraphQL, Kafka Streams, Tyk Api Gateway, Kong API Gateway y iPaaS.

Hemos podido ver que todas ellas tienen sus ventajas y desventajas. Algunas como HTTP Compression se deberían usar tan a menudo como sea posible, mientras que otras como GraphQL o Server-Sent Events tienen casos de uso más específicos. Hay también las que tienen comportamientos similares, como Tyk y Kong en la que tenemos que escoger la que usar dependiendo de nuestras expectativas, mientre que otras como Reactive APIs y Webhooksse complementan entre ellas.

Al final, decidir qué usar y qué no depende de los sistemas que queremos integrar. Las soluciones estudiadas en este documento han demostrado ser útiles para solucionar problemas específicos.

I want to dedicate this project to my family, who has always supported me and been by my side throughout all my degree and life.

Acknowledgements

I would like to specially thank both my advisor José Luís Muñoz and my friend Enric Ruhi for taking their time to read the complete project and having given advise about how to improve it. I would also want to thank Miguel Matarín for suggesting the study of the technologies listed on this document and Sergio Nogueiras for giving a helping hand on learning how to work with the AWS account.

Revision history and approval record

Revision	Date	Purpose
0	01/03/2018	Document creation
1	04/05/2018	Document revision
2	18/05/2018	Document revision
3	8/06/2018	Document revision
4	22/06/2018	Document revision
5	29/06/2018	Document acceptance

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Sergi Roura Saez	sroura15@gmail.com
Jose Luís Muñoz	jose.munoz@entel.upc.edu

Written by:		Reviewed and approved by:	
Date	01/03/2018	Date	29/06/2018
Name	Sergi Roura	Name	Jose Luís Muñoz
Position	Project Author	Position	Project Supervisor

Table of contents

Abstract 1

Resum 2

Resumen 3

Acknowledgements 5

Revision history and approval record 6

Table of contents 9

List of Figures 10

1 Introduction 11

2 State of the art of the technologies used in this thesis 12
2.1 Webhooks . . . 12
2.1.1 Background . . . 12
2.1.2 Why use Webhooks? . . . 12
2.1.3 Securing Webhooks . . . 13
2.1.4 When to use webhooks . . . 13
2.1.5 Webhooks in companies . . . 13
2.2 HTTP Compression . . . 13
2.2.1 Background . . . 13
2.2.2 Benefits of HTTP compression . . . 13
2.2.3 Support of HTTP compression . . . 14
2.2.4 Problems preventing use of HTTP compression . . . 14
2.3 Reactive APIs . . . 14
2.3.1 Background . . . 14
2.3.2 What is Reactive Programming? . . . 14
2.3.3 Why adopt Reactive Programming . . . 15
2.3.4 Understanding Reactive Programming . . . 15
2.3.5 Reactive APIs in companies . . . 15
2.4 Server-Sent Events (SSE) . . . 15
2.4.1 Motivation . . . 15
2.4.2 Server Sent Events or Websockets? . . . 16
2.4.3 Server-sent Events in companies . . . 16
2.5 GraphQL . . . 16
2.5.1 Background . . . 16
2.5.2 Main characteristics . . . 17
2.5.3 API's Schema definition . . . 17
2.5.4 Solving API operations . . . 17

2.5.5 Best Practices . . . 18
2.5.6 OData. Another way to REST . . . 18
2.5.7 GraphQl in companies . . . 18
2.6 Kafka Streams . . . 19
2.6.1 Background . . . 19
2.6.2 What is Kafka Streams . . . 19
2.6.3 Kafka basics . . . 19
2.6.4 Kafka in companies . . . 20
2.7 Kong . . . 20
2.7.1 Background . . . 20
2.7.2 Key Concepts . . . 20
2.7.3 Kong in companies . . . 21
2.8 Tyk . . . 21
2.8.1 Background . . . 21
2.8.2 How Tyk MDCB Works . . . 22
2.8.3 Tyk MDCB Logical Architecture . . . 22
2.8.4 Tyk in companies . . . 22
2.9 Integration Platform as a Service (iPaaS) . . . 23
2.9.1 Background . . . 23
2.9.2 Cloud Service Models . . . 23
2.9.3 Why adopt iPaaS . . . 23
2.9.4 iPaaS or ESB? . . . 24
2.9.5 CloudReach iPaaS Service . . . 24
2.9.6 IPaaS in companies . . . 24

3 Project development- Proofs of Concept 25
3.1 Webhooks . . . 25
3.2 HTTP Compression . . . 25
3.3 Reactive APIs . . . 25
3.4 Server-Sent Events (SSE) . . . 26
3.5 GraphQL . . . 26
3.6 Kafka Streams . . . 26
3.7 Kong . . . 27
3.8 Tyk . . . 27
3.9 Integration Platform as a Service (iPaaS) . . . 28

4 Results 29
4.1 Webhooks . . . 29
4.2 HTTP Compression . . . 29
4.3 Reactive APIs . . . 30
4.4 Server-Sent Events . . . 30
4.5 GraphQL . . . 30
4.6 Kafka Streams . . . 31
4.7 Kong . . . 32
4.8 Tyk . . . 33
4.8.1 Kong vs Tyk . . . 33
4.9 Integration Platform as a Service (iPaaS) . . . 33

5 Budget 35

6 Conclusions and future development 36
6.1 Conclusions . . . 36
6.2 Future development . . . 36

Bibliography **37**

Appendix **43**

Glossary **47**

List of Figures

2.1 Webhooks flow. . . . 12
2.2 Comparison between compressed and non-compressed HTTP requests. . . . 13
2.3 Example of request headers in Chrome. . . . 14
2.4 Reactive APIs flow. . . . 15
2.5 Comparison between polling and server-sent event's flows. . . . 16
2.6 GraphQL schema. . . . 17
2.7 GraphQL resolver. . . . 17
2.8 Companies that use Apollo and GraphQL. . . . 18
2.9 Kafka Streams diagram. . . . 19
2.10 Comparison between Kong and legacy architecture. . . . 21

3.1 HTTP compression response. . . . 25
3.2 Result of defining a route to an API inside Kong. . . . 27
3.3 Tyk dashboard. API Management screen. . . . 28
3.4 AWS API gateway. Backend connection . . . 28

4.1 Result of a GET request. . . . 29
4.2 Result of a compressed response. . . . 29
4.3 Reactive application . . . 30
4.4 GraphQL based webpage. . . . 31
4.5 Synchronized applications using Kafka Streams. . . . 32
4.6 Result of GET request through API Gateway. . . . 33
4.7 Running Tyk API. . . . 33
4.8 Result of GET request to public API Gateway. . . . 34

Chapter 1

Introduction

The purpose of this project is the analysis and realization of proofs of concept of several solutions for communication's optimization between systems, based on APIs, to determine which scenarios offer the most benefits in the business ambit.

All proofs of concept have been developed under the same conditions, using Windows 7 as operational system and working with the framework Angular on the frontends and NodeJs for the backends.

The project requirements that this document had to accomplish were the following ones:

1. Benchmarking of solutions.
2. Proof of concept of solutions.
3. Generate analysis documentation.
4. Redaction of conclusions.

In order to fulfill them, the procedure followed was to search information about each of the solutions to then generate an introduction for each of them with an explanation of their most relevant characteristics. Then study the required tools needed to test our technologies and explain the parts of each of them we would be using during the project as well as the procedure to install them under our conditions. The last steps were to generate custom proofs of concepts explained in detail so that they could be reproduced and analyze the technology aspects while doing so, to then extract some conclusions for each of them including in some cases a comparison between them.

This project has its origin at Everis Barcelona, where it has been partly developed. Its idea came from a reunion where we discussed the need to do some analysis of several technologies to decide which to use in the project we were developing. A long list of them came up from it and finally it was decided to narrow it to these 9: Webhooks, Http Compression, Reactive APIs, Server-Sent Events, GraphQl, Kafka Streams, Tyk API Gateway, Kong API Gateway, iPaaS.

The views used for the first web where developed by the author itself several months ago as a little part of a project developed in the company. They are a, now outdated, representation of digital architectures, that the author has completely modified for them to work for the purposes of this document with the consent of Everis' project leader.

The work plan, milestones and Gant diagram can be found on **[Appendix A]**.

The complete project, including all the required tools, studies and proofs of concept is included in a separate document called **Technical Appendix** of 134 pages length.

Chapter 2

State of the art of the technologies used in this thesis

2.1 Webhooks

2.1.1 Background

Over recent years more and more services are offering the ability to configure **webhooks** that notify you when something interesting has happened. It is increasingly being used by companies to improve their customer experience.

A webhook is a user-defined HTTP callback which retrieves and stores data from an event, usually from outside of your software application.

If you want to use it, you need to provide a callback URI, and the service offering the webhook will make a HTTP request to that URI whenever the event of interest occurs. This allows you to write your own code that responds to the event when it happens. All you need is the ability to listen for HTTP requests to the callback URI. It is also possible to configure it to work on a NAT network.

2.1.2 Why use Webhooks?

The main reason we need webhooks is that we don't want to have to continuously poll a service to discover what has happened recently. Polling is a waste of resources both for the client and server. Giving the server the ability to push notifications via webhook callbacks solves the problem.

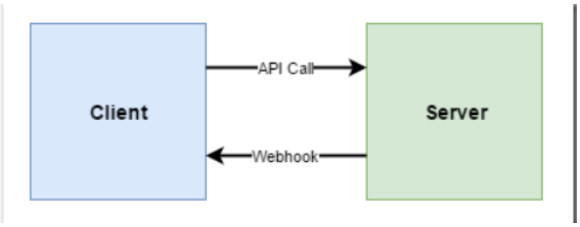


Figure 2.1: Webhooks flow.

With webhooks, the issuer of a webhook is the one who owns the contract, they not only define the incoming API but they also define the payload of the webhook. So one of the services doesn't need to know anything about who the other service is (i.e: in REST API development, the backend doesn't need to know the url of the frontend, it just responds automatically to the one who called its services).

2.1.3 Securing Webhooks

Webhooks should always use HTTPS. No-one should be able to intercept a webhook payload and examine it. They should also include a pre-shared secret somewhere in the HTTP request. It would be even better if they include a hash calculated using a pre-shared secret and the JSON payload so that the secret is never transmitted, but that would introduce more complexity for the recipient of the webhook. Another important security step is to sanity check the webhook payload and limiting the scope information shared in the callback.

2.1.4 When to use webhooks

If the service let users initiate long-running operation, or there are events that the users might like to be notified about in real-time , offering webhooks will make the system much easier for them to work with. However, the use of webhooks should be made optional, and not provide any information that could not also be retrieved through polling. This way, the users who don't want to (or can't) host a webhook will still be able to use the service.

2.1.5 Webhooks in companies

A good example of webhooks in action is **GitHub**, who allows to set up webhooks to subscribe to various events. So if someone wants to be notified when a pull request is created, he can tell GitHub where the webhook is hosted, and then whenever a pull request is created on his project, GitHub will make a HTTP request to the callback URI, posting a JSON object that contains information about the pull request. Webhooks are also commonly used on payment providers.

Other examples of webhooks use would be in chatbots implementations such as **Inbenta**'s Veronica, **Sendgrid**'s Event Webhook that notifies whenever a mail is received or **Shopify**, that offers them to keep the user's e-commerce data updated.

2.2 HTTP Compression

2.2.1 Background

HTTP compression is a capability that can be built into web servers and web clients to improve transfer speed and bandwidth utilization.

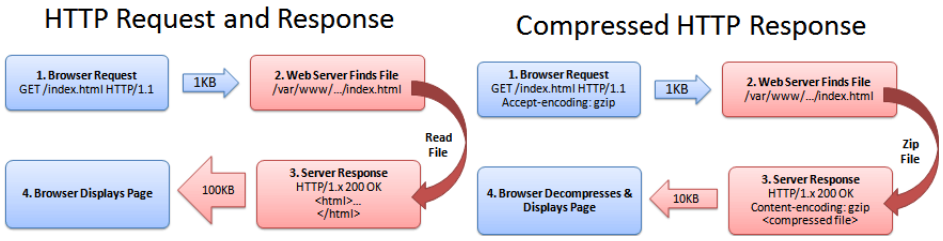


Figure 2.2: Comparison between compressed and non-compressed HTTP requests.

HTTP data is compressed before it is sent from the server. There are two different ways compression can be done in HTTP:

- At a lower level, a Transfer-Encoding header field may indicate the payload of a HTTP message is compressed.
- At a higher level, a Content-Encoding header field may indicate that a resource being transferred, cached, or otherwise referenced is compressed. This second one is more widely supported.

2.2.2 Benefits of HTTP compression

HTTP compression was created to be a solution for several of the performance issues of the Web by reducing the size of the resources transfered to a server, therefore lowering the user's perceived latency and making a better use of the network's bandwidth.

Many Web resources, such as the ones used in Angular and NodeJS (HTML, CSS, Typescript, Javascript...) are simply ASCII files with the same information repeated over and over again. HTTP compression proves to be especially benefitial in those cases, reducing a large amount of bytes and therefore enhancing the user experience. Also, for those services that are payed depending on the amount of bandwidth consumed, it becomes a way to save money.

2.2.3 Support of HTTP compression

Most navigators support some form of HTTP compression ever since HTTP/1.0 was out, but it was mostly forgotten and unused. It wasn't until HTTP/1.1 appeared that its use started to be more significant. The clear example is chrome, that uses gzip compression on both the requests and responses of our API rest.

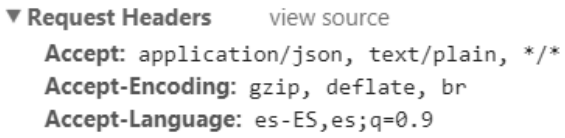


Figure 2.3: Example of request headers in Chrome.

2.2.4 Problems preventing use of HTTP compression

Antivirus software can interfere with connections to force them to be uncompressed. Also, the use of proxies, misconfiguration of servers and browser bugs can prevent this compression to be used. Internet Explorer 6 was the browser most prone to failing back to uncompressed HTTP. Another problem found while deploying HTTP compression on large scale is due to the deflate encoding definition, that made the deployment unreliable. For this reason some software only implement gzip encoding.

2.3 Reactive APIs

2.3.1 Background

Reactive programming is a really popular subject in the JavaScript and Android communities. The term itself has been around for many years but until Microsoft released the Reactive Extension[1] for .NET it wasn't quite well considered. Right now, RxJava and RxJs are widely adopted for Android and Javascript users respectively and some frameworks such as Angular from Google includes them as a hard dependency.

2.3.2 What is Reactive Programming?

Reactive programming bases on programming with asynchronous data streams using a toolbox of functions to combine, create and filter any of them. A stream, usually called Observable, is a sequence of ongoing events ordered in time that can emit a value, an error or a "completed" signal. You can listen to that stream by `subscribing` to it and create functions that execute when one of the results,called Observers, is emitted. All this conforms the `Observer Pattern`[47].

[1] Rx is one of the most known libraries for Reactive Programming.

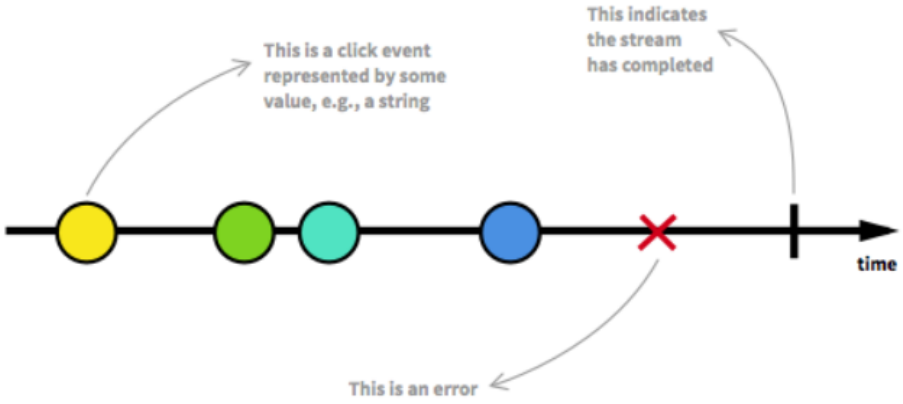


Figure 2.4: Reactive APIs flow.

2.3.3 Why adopt Reactive Programming

Reactive Programming raises the level of abstraction of the code so you can focus on the interdependence of events that define the business logic, rather than having to constantly fiddle with a large amount of implementation details. Code in RP will likely be more concise. In modern webapps and mobile apps the benefit is more evident when having a multitude of UI events related to data events.

Apps nowadays have an abundancy of real-time events of every kind that enable a highly interactive experience to the use by making the application more responsive.

2.3.4 Understanding Reactive Programming

There are interactive websites such as RxMarbles [50] that show in an easy and fun way how Rx Observables work. Also, there are many articles about the topic that explain in a really accurate and extent way the benefits of using these kind of technics. One of the most recommended ones is "Understanding reactive programing through an example with rxjs part 1" [51] from Potcharapol Suteparuk.

2.3.5 Reactive APIs in companies

The list of businesses that have adopted reactive programming is extremely long. Just using it with Angular or ReactJs for their projects we already have companies such as Everis, The Guardian, Google, Vevo, Weather Channel, Youtube, Paypal, Upwork, Freelancer, between other hundreds of them. It has become an extent and really popular way of writing code.

2.4 Server-Sent Events (SSE)

2.4.1 Motivation

SSE is a mechanism that enables server to asynchronously push the data via HTTP from the server to the user once the user-server connection is established by the user. Once the user established the connection, the server continously sends data and decides to send it to the user whenever a new piece of data is available.

With real-time communication, instead of requiring the user to refresh the browser, new information can be appended instantly to a web page. In early days, you would have to resort to polling via AJAX. This puts strain on both browser and server, and data was always at least seconds behind. Since then, new technologies have emerged that push information to the browser and back to the server as soon as it arrives.

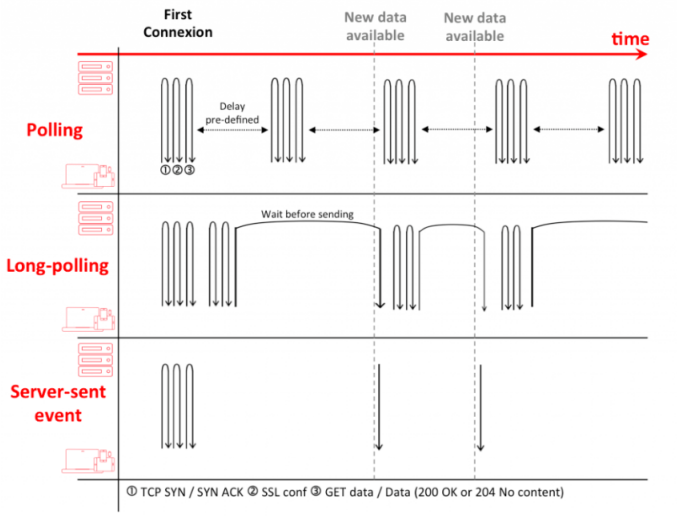


Figure 2.5: Comparison between polling and server-sent event's flows.

Because SSEs are streams of data, it is important for the user to have long-lived connections.The server used has to be able to handle large numbers of simultaneous connections. Some examples are Juggernaut, Node.js and Twisted.

2.4.2 Server Sent Events or Websockets?

One reason SSE has been kept in the shadow is because later APIs like WebSockets provide a richer protocol to perform bi-directional, full-duplex communication. Having a two-way channel is usually more attractive for cases where you need near real-time updates in both directions. However, in some scenarios data doesn't need to be sent from the client. You simply need updates from some server action. A few examples would be friends' status updates, stock tickers, news feeds, or other automated data push mechanisms.

SSE are sent over traditional HTTP. That means they do not require a special protocol or server implementation to get working. WebSockets on the other hand, require full-duplex connections and new Web Socket servers to handle the protocol. In addition, Server-Sent Events have a variety of features that WebSockets lack by design such as automatic reconnection, event IDs, and the ability to send arbitrary events.

2.4.3 Server-sent Events in companies

There are several uses for these types of events. Nowadays the most popular ones are:

- Real-time chart streaming live stock prices
- Twitter/Fb walls receiving live update's by Twitter's streaming API.
- Monitor for server stats like health, uptime, and other running processes.
- A sports portal using SSE push live updates right to a browser.
- Arrival & departure tables for flights or trains.
- Remote health monitoring for elderly care.

2.5 GraphQL

2.5.1 Background

GraphQL is a tool developed by Facebook that intends to substitute RestFull. It is available for javascript, typescript, ruby, python, java, scala, c# (between others) and for clients like React, React Native, Angular 2, Android and iOS.

2.5.2 Main characteristics

- With GraphQL, the application is the one that has control over the resource, reducing the bandwidth and times. If only one field of a resource is needed, its use is recommendable.
- It is possible to access several resources with just one call.
- `Endpoints` are deleted, being substituted by types that only indicate what is needed.
- It tries to delete the versioning of APIs.
- Disengages the dependency from database access.

2.5.3 API's Schema definition

The Schema defines the operations and types that our API exposes[58].

Analyzing the Schema we can observe 3 parts:

```
module.exports=new graphql.GraphQLSchema({
  query: QueryType,
  mutation: MutationType
});
```

Figure 2.6: GraphQL schema.

- **schema:** section where the operation's types are defined. The `Queries` will be the types of queries that can be done and the `Mutations` the way how we modify the server.
- **Query:** They are the defined queries, with the input and output parameters.
  `Example:` query1 (parameter1: [String]): [ReturnType].
- **Mutation:** They are the modifications that can create queries.

After defining the operations, we define the types that can be returned using the keyword `type`. Also, we define the input types with the keyword `input`.

2.5.4 Solving API operations

```
resolve: (root, args) => {
    var newArchitecture = new Architecture({
        _id: args._id,
        name:{defaultText:args.name, defaultLanguage:"en"},
        description: {defaultText:"", defaultLanguage:"en"},
        evaluable: false
    })

    return new Promise((resolve, reject) => {
        newArchitecture.save(function (err) {
            if (err) reject(err)
            else resolve(newArchitecture)
        })
    })
}
```

Figure 2.7: GraphQL resolver.

To tell GraphQL how to resolve the operations after receiving resource petitions, we use `resolvers`. This indicates which functions should them call to receive a certain query. With this we would have a defined and resolved API and we could start using it.

2.5.5 Best Practices

HTTP

GraphQL is typically served over HTTP via a `single endpoint` which expresses the full set of capabilities of the service. While GraphQL could be used alongside a suite of resource URLs, this can make it harder to use with tools like GraphiQL.

JSON+GZIP

GraphQL services typically respond using JSON, however the GraphQL spec does not require it. The reason why JSON is used is because it is mostly text, and compresses exceptionally well with GZIP. It's encouraged that any production GraphQL services enable GZIP and encourage their clients to send the header *Accept-Encoding: gzip*.

Versioning

Most APIs are versioned because, when there's limited control over the data that's returned from an API endpoint, any change can be considered a breaking change that requires a new version. In contrast, GraphQL only returns the data that's explicitly requested, so new capabilities can be added via new types and new fields on those types without creating a breaking change.

Nullability

Most type systems which recognise `null` provide both the common type, and the `nullable` version of that type, where by default types do not include null unless explicitly declared. However in a GraphQL type system, every field is nullable by default. By doing so, most errors may result in just a field returned `null` rather than having a complete failure for the request. Instead, GraphQL provides non-null variants of types which make a guarantee that if requested, the field will never return null. When designing, it's important to keep in mind all the problems that could go wrong and if nul is an appropriate value for a failed field.

Server-side Batching & Caching

GraphQL is designed in a way that allows you to write clean code on the server, where every field on every type has a focused single-purpose function for resolving that value. You can also use a batching technique, where multiple requests for data from a backend are collected over a short period of time and then dispatched in a single request to an underlying database or mircroservice by using a tool like Facebook's DataLoader.

2.5.6 OData. Another way to REST

There are other technologies similar to GraphQl that are also emerging to try to revolve REST programming.
OData (Open Data Protocol) defines a set of best practices for building and consuming RESTful APIs. It helps you focus on your business logic while building RESTful APIs without having to worry about the various approaches to define request and response headers, status codes, HTTP methods, URL conventions, media types, payload formats, query options, etc. It also provides guidance for tracking changes, defining functions/actions for reusable procedures, and sending asynchronous/batch requests.
Even though it was not possible to start an approach to this technology during the construction of this document, it is recommended to check it's main website `[70]` and test it.

2.5.7 GraphQl in companies

The companies from `Figure 2.8` use Apollo and GraphQL to power their most important applications [66].

Figure 2.8: Companies that use Apollo and GraphQL.

One of the most relevant companies on this list is **The New York Times**. As they explain in these articles [67] [68], apollo offered many improvements and options that React alone couldn't match.
Another one is **Airbnb**, as shown in the article [69], which ended up using it in its search page.

2.6 Kafka Streams

2.6.1 Background

Kafka is a fault tolerant, distributed publish-subscribe messagin system that is designed for fast processing of data and the ability to handle hundreds of thousands of messages. Stream processing is the real-time processing of data continuously, concurrently, and in a record-by-record fashion. In terms of kafka, real time processing typically involves reading data from a topic (source) doing some analysis or transformation work, and then writing the results back to another topic (sink).

2.6.2 What is Kafka Streams

Kafka Streams is a library for building streaming applications, specifically applications that transform input Kafka topics into output Kafka topics (call external services, update databases, etc.). Kafka Streams allows you to do this with concise code in a way that is distributed and fault-tolerant.
It can be a good alternative in scenarios where you want to apply a stream processing model to your problem, without embracing the complexity of running a cluster. The Apache Team provides excellent documentation for people for people who want to get introduced into this streaming platform [73].

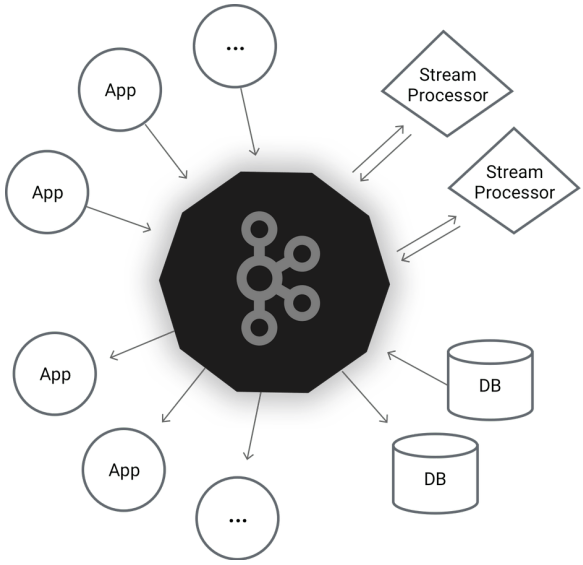


Figure 2.9: Kafka Streams diagram.

2.6.3 Kafka basics

To understand Kafka, let's first list some of the basics:

- Applications that send data to Kafka are called producers.
- Applications that read data from Kafka are called consumers.
- Producers send records. Each record is associated to a topic. A topic is like a category. Each record consists of a key, value and timestamp.
- Consumers can subscribe to a given topic, receive a stream of records and be alerted whenever a new record is sent.

- In the event that a consumer goes down, it is able to restart streaming from where it left off, by keeping track of the topic's offset.
- Kafka guarantees the order of messages in given topics, regardless of the number of consumers or producers.

Through Kafka's architecture, we are able to decouple the production of messages from the consumption of them.

2.6.4 Kafka in companies

There are more than 300 companies that use Kafka on their projects nowadays. Some of the most known are Spotify, Hubstop, Code School, Tumblr, WePay, between others.

2.7 Kong

2.7.1 Background

Kong is an open-source API gateway and microservice management layer based on Nginx. Its pluggable architecture makes it flexible and powerful. It's like a security layer which sits in front of the application and enhances it's performance.Kong provides full control over architecture and it's currently used by many organizations including small and large ones.

2.7.2 Key Concepts

- **API Object:** wraps properties of any HTTP endpoint that accomplishes a specific task or delivers some service.
- **Consumer Object:** wraps properties of anyone using our API endpoints. It will be used for tracking, access control and more.
- **Upstream Object:** describes how incoming requests will be proxied or load balanced, represented by a virtual hostname.
- **Target Object:** represents the services implemented and served, identified by a hostname(or IP address) and a port.
- **Plugin Object:** pluggable features to enrich functionalities of our application during the request and response lifecycle. Kong provides very powerful plugins.
- **Admin API:** RESTful API endpoints used to manage Kong configurations, endpoints, consumers, plugins, and so on.

`Figure 2.10` depicts how Kong differs from a legacy architecture:

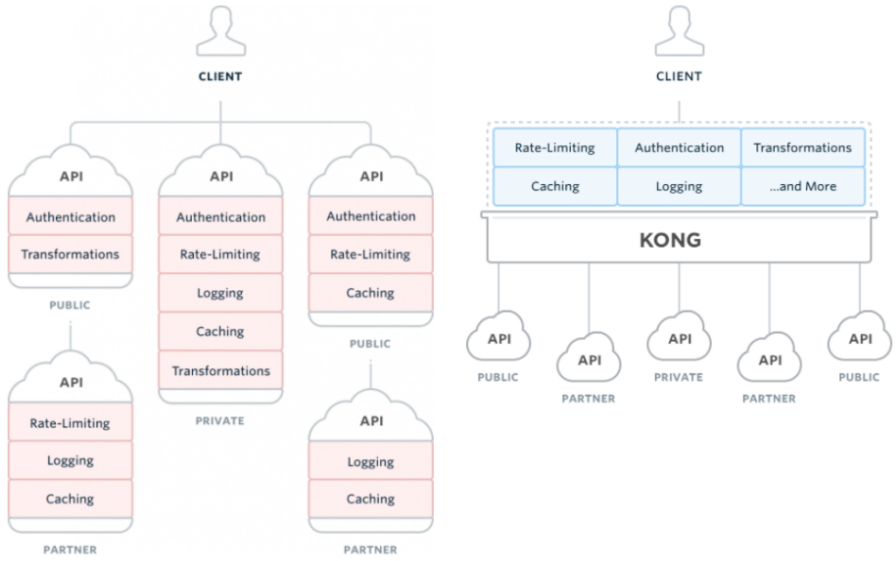


Figure 2.10: Comparison between Kong and legacy architecture.

2.7.3 Kong in companies

Kong is used in production at hundreds of organizations from startups to large enterprises and government departments. Some of them are:

- Mashape MarketPlace
- DataBC
- EuroStar
- Expedia
- Harvard University
- IBM
- Intel
- OpenDNS
- The Guardian
- The New York Times

2.8 Tyk

2.8.1 Background

Tyk Multi Data Center Bridge (Tyk MDCB or Tyk Sink) acts as a broker between Tyk Gateway Instances that are isolated from one another and typically have their own Redis DB.
In order to manage physically separate Tyk Gateway clusters from a centralised location, Tyk MDCB needs to be used to provide a remote "back-end" for token and configuration queris.

2.8.2 How Tyk MDCB Works

Tyk MDCB creates a bridge between a configuration source (MongoDB and a centralised Redis DB) and multiple Tyk gateway instances, this bridge provides API Definitions, Policy definitions and Org rate limits, as well as acting as a data sink for all analytics data gathered by slaved Tyk gateways.
The communication between instances works through a compressed RPC TCP tunnel between the gateway and MDCB, it is **incredibly fast** and can handle 10's of thousands transactions per second.

2.8.3 Tyk MDCB Logical Architecture

The Tyk MDCB Logical architecture consists of:

- A master Tyk cluster which must not be tagged or sharded or zoned in any way. It stores configurations for all valid APIs to facilitate key creation.
- MDCB instances to handle the RPC connections.
- The Tyk Slave clusters, which consist of Tyk Nodes and an isolated Redis DB.

Since Tyk instances connected to MDCB are slaved, to first get slave clusters set up it is needed a master. It can be an existing Tyk Gateway setup, it does not need to be separately created but it will hold a copy of all tokens across all zones. It needs to consist of:

- A Dashboard instance
- A Master Tyk gateway instance
- A master Redis DB.
- A MongoDB replica set for the dashboard and MDCB.
- One or more MDCB instances, load balanced with port 9090 open for TCP connections

The slave clusters are essentially local caches that run all validation and rate limiting operations locally instead of against a remote master that could cause latency. The procedure to handle a request is the following:

1. Request arrives
2. Auth header and API identified
3. Local cache is checked for token, if it doesn't exist, attempt to copy token from RPC master node.
4. If token is found in master, copy to local cache and use.
5. If it is found in the local cache, no remote call is made and rate limiting and validation happen on the local copy.

The slave clusters consists of:

- Tyk gateway instances specially configured as slaves
- A redis DB.

2.8.4 Tyk in companies

Tyk is a fairly new technology and as such, doesn't have yet an extent number of affiliates. However, its usage is increasing and little by little it is gaining more and more influence as an API Management solution.
Some of the companies that have started integrating their APIs with Tyk are Senz, Twig and Veris.

2.9 Integration Platform as a Service (iPaaS)

2.9.1 Background

Nowadays, data is no longer being stored centrally; it's scattered and distributed across multiple apps, databases and data centers.
The biggest challenge companies face today is integrating the data that is coming in and out of various web-based and on-premise apps.
iPaaS is a platform that allows users to connect various cloud and on-premise apps and then deploy these integrations without writing any code or installing additional software or hardware.
It's most important value is the functionality it offers. Any user can use an iPaaS to move data from any database or application to another app, automatically. This allows enterprises to automate complex business processes that span across web-based apps and on-premise resources.

2.9.2 Cloud Service Models

There are several types of cloud service models, such as:

- **IaaS** stands for Infrastructure as a Service. It provides users with the capability to provision processing, storage, and network connectivity on demand. Using this service model, the customers can develop their own applications on these resources.
- **PaaS** stands for Platform as a Service. The service provider provides services like databases, queues, workflow engines or e-mails to their customers. The customer can then use these component for building their own applications. The services, availability of resources and data backup are handled by the service provider that helps the customers to focus more on their application's functionality.
- **SaaS** stands for Software as a Service. The third-party providers provide end-user applications to their customers with some administrative capability at the application level, such as the ability to create and manage their users. Also some level of customization is possible.
- **iPaaS** stands for Integration Platform as a Service. It is now emerging as the next generation integration platforms for integrating cloud applications with one another and with on-premices and legacy applications.

2.9.3 Why adopt iPaaS

- iPaaS brings an organization's unique needs together into a cloud-based toolset.
- **Cloud-based Integrations:** Modern iPaaS solutions are cloud-friendly and more secure. Integrations between cloud applications happen in the cloud, thereby eliminating the risk of exposing a local network and data to the internet.
- **Cost-efficiency:** iPaaS eliminates the need to hire expensive developers to write custom integrations. The required platform is available as monthly/yearly subscriptions, and is mostly in the cloud. No hardware or software installations are required to integrate applications.
- **Agility:** iPaaS offers speed and flexibility which enables greater business agility. Required integrations can be created, edited, and disposed of when needed, without worrying about additional costs or provider permissions.
- **Updates Over Upgrades:** iPaaS provides reliable, automatic, and regular updates that reduce the cost of maintenance, improve reliability, and ensure that you make use of the latest features and enhancements as they become available. This becomes critical when merging with SaaS applications that provide updates to their API and security protocols regularly. A good iPaaS will update actions and activities on a bi-weekly basis so that a user doesn't have to worry about keeping up with updates.
- **Speed and Simplicity:** iPaaS enables you to connect and automate workflows for your business quickly and easily, whether you are connecting to new cloud solutions, to existing core on-prem systems, or starting up a zero-footprint enterprise.

2.9.4 iPaaS or ESB?

iPaaS is often used in business-to-bussiness [97] scenarios where the speed of release times is a key requirement. The problem with this type of prepackaged integration is that it increases the threat of vendor lock-in [98].
Modern ESBs(Enterprise service bus) can handle SaaS applications and also prove useful when integrating legacy applications. On the other hand, iPaaS is cheaper than ESB, offers more scalability and the advantageous for business-to-bussiness integration outside an organization's own systems.

2.9.5 CloudReach iPaaS Service

Nowadays there are just a few services that allow iPaaS implementations, specifically made for companies who need integration of projects in a medium to large scale. We will be checking how to implement it in AWS since it is a known Cloud platform but there are others like CloudHub that are specifically intended for iPaaS implementations.
One of AWS's partners, provides one of the emerging services that allow such integrations,CloudReach [105].
It's pricing is around 1500-4500$ and allows an incredibly high amount of scenarios to integrate different systems from databases to services to on-premise applications.

2.9.6 IPaaS in companies

IPaaS vendors work towards integrating enterprise systems within the cloud and also between public and private clouds. There are two basic schools of vendors for iPaas:

- The **old school**, formed by existing integration companies that have modified their tools to work with cloud services and are using their experience to incorporate users' needs and create integration service platforms. Some examples of these are **TIBCO**, **Informatica** and **IBM**.
- The **new school**, formed by companies that were born within the cloud age and have found success based on and around cloud services.Some examples are **Jitterbit, Dell Boomi, SnapLogic** and **MuleSoft**.

Chapter 3

Project development- Proofs of Concept

All proofs of concept, procedures and methodologies are included in a separate **Technical Appendix** alongside with all the required tools used to develop this project. The next following lines include a short summary of the procedures followed for each technology.

3.1 Webhooks

To develop the webhooks proof of concept, we created a NodeJs server connected to a MongoDB database. We filled the Mongo database with some JSON documents with information about architectures and prepared the server to allow performance of CRUD operations to treat that information. We programmed it to implement webhooks, so that whenever a request was made, the data was retrieved and automatically sent back to the requester.

```
routes.get('/',(req , res )= >{
        Architecture.getArchitectures((err,architectures )= >{
                if(err){
                        res.send(err);
                }
                res.json(architectures);
        });
});
```

3.2 HTTP Compression

To develop the HTTP compression proof of concept, we added the dependencies required to perform it in our NodeJs server. Then we filled our database with thousands of JSON documents in order to make the response as heavy as possible, in order to trigger the use of compression. Finally, we used a navigator inspector to test the differences between applying it or not.

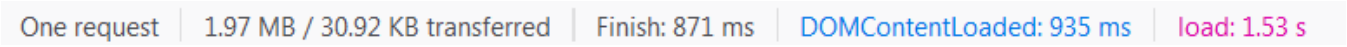


Figure 3.1: HTTP compression response.

3.3 Reactive APIs

To develop the reactive API proof of concept, we created some Angular views with typescript as the main language. We established a communication with the NodeJs server to retrieve data and display it using html. While doing so, we prepared our functions to retrieve the responses as Observable objects, in order to asynchronously wait for the response of the server and subscribe to its result. This way we obtained a full-fledged Reactive API web, with an Angular API as our frontend and a Node Server as the backend.

```
this.http.get(API_ENDPOINT+'/architectures').pipe(map(res =>res.json())).
subscribe(data=>{
        this . dataStore . architectures = data ;
        this . _architectures$ . next ( this . dataStore . architectures );
})
```

3.4 Server-Sent Events (SSE)

To develop the server-sent events proof of concept, we modified the code created to test reactive APIs and prepared the server to return the data from the database in a real-time stream and the views to listen to that stream.

```
app.get('/eventstream',(req, res, next)=>{
        res.set({
                'Content-Type': 'text/event-stream ',
                'Cache-Control': 'no-cache',
                'Connection': 'keep-alive'
        });
        app.on('message', data => {
                res.write(`event:message \n`);
                res.write(`data : ${JSON.stringify(data)}\ n\n `);
        });
});
```

3.5 GraphQL

To develop the graphql API proof of concept, first we created a new NodeJs server using the graphql syntaxis to program it, using a single endpoint defined by with an schema to perform all the mutations and queries required to retrieve JSON objects from a new database. After testing its performance with graphiql, we created a new Angular project using the apollo client's syntax to generate the queries required to perform the CRUD operations. Finally, we connected them both and tested its functionality by performing some operations getting the objects and the response time and displaying it on the views.

```
export class GraphQLModule {
        constructor(apollo : Apollo, httpLink : HttpLink){
                const uri ='http://localhost:3000/';
                const http = httpLink.create({uri});
                apollo.create ({
                        link : http ,
                        cache : new InMemoryCache()
                });
        }
}
```

3.6 Kafka Streams

To develop the Kafka Streams proof of concept, first we created some custom producers and consumers using terminals to test how the data was transmitted. Then, we modified the Node servers we created for both the reactive API and graphql tests to perform the functions of a producer and consumer respectively. Finally, we checked its behavior by performing operations on both frontends and synchronizing the data in order to update at the same time both databases and views.

```
const kafka = require('kafka-node'),
Consumer = kafka.Consumer,
client = new kafka.Client(),
consumer = new Consumer(client, [{topic:'Architectures', offset:0}],
{autoCommit:false});

consumer.on('message', function(message){
        console.log(message);
});
consumer.on('error', function(err){
        console.log('Error'+ error);
});
consumer.on('offsetOutOfRange', function(err){
        console.log('offSetOutOfRange:', err);
});
```

3.7 Kong

To develop the Kong proof of concept, we prepared a Docker Linux virtual machine in order to host it and uploaded our NodeJs server code on another one. Then we established a connection between both of them and used postman to administrate it, test its functionalities and check some of the different plugins it supports to allow some basic security implementations.


```
{
    "created_at": 1527798860,
    "strip_path": true,
    "hosts": [
        "tfgService"
    ],
    "preserve_host": false,
    "regex_priority": 0,
    "updated_at": 1527798860,
    "paths": null,
    "service": {
        "id": "d59c46e3-6108-41df-bcc9-87f09167e71d"
    },
    "methods": null,
    "protocols": [
        "http",
        "https"
    ],
    "id": "a4f5adea-339c-47d6-a99f-3592aae4171d"
}
```


Figure 3.2: Result of defining a route to an API inside Kong.

3.8 Tyk

To develop the Tyk proof of concept, we prepared a Docker Linux virtual machine in order to host it and uploaded our NodeJs server code on another one. Then, we created a new session on Tyk's dashboard and using its interface we established a connection between both of them and administrate it. Finally, we tested some of the functionalities it performed such as basic security implementations.

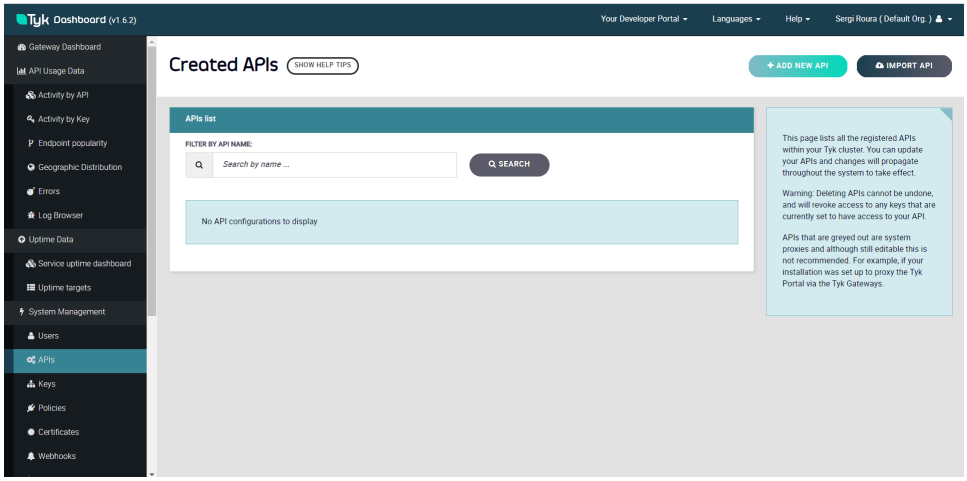


Figure 3.3: Tyk dashboard. API Management screen.

3.9 Integration Platform as a Service (iPaaS)

To develop the AWS API gateway's proof of concept, we uploaded our NodeJs server and database onto an EC2 instance within the Cloud. Then, after configuring it, we performed a connection between the AWS API gateway and said instance to test its performance, and included some client certificates in order to avoid foreign users from performing calls to it. Finally, we deployed our Angular application onto an S3 instance, set it up in order to have public access, and connected it to the API gateway using an API key, in order to have a live public webpage. Finally we accessed said web and performed operations to test its behavior.

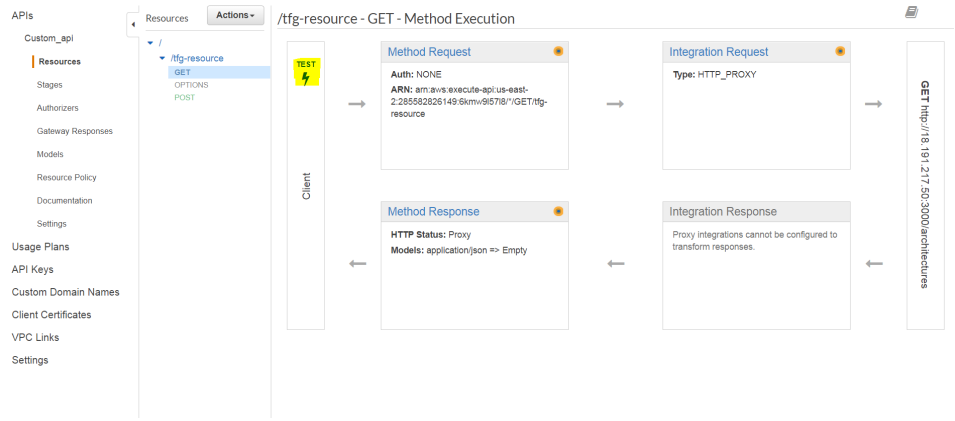


Figure 3.4: AWS API gateway. Backend connection

Chapter 4

Results

4.1 Webhooks

Webhooks have proven to be an easy and reliable way to let the server push notifications back to a client without the need to create extra code to do so. It helps us developers to not to worry about polling the system to get responses and thus let's us center ourselves on the correct development of functionalities.

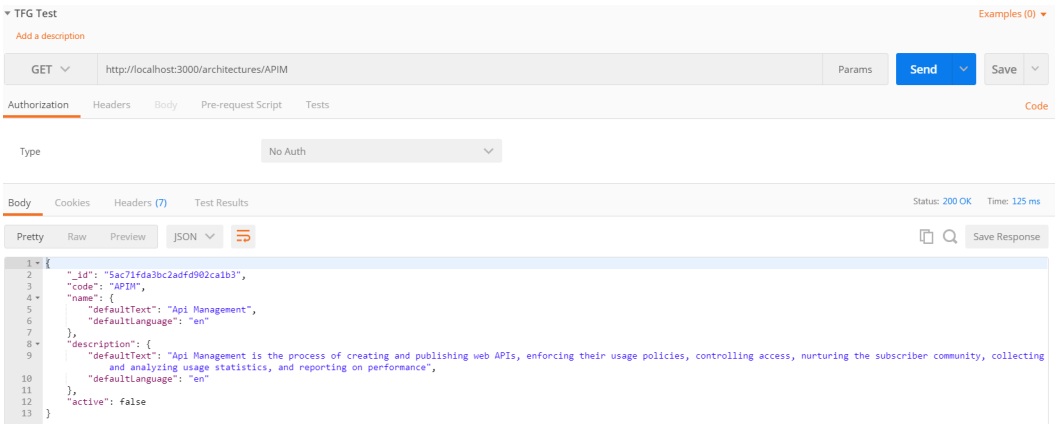


Figure 4.1: Result of a GET request.

4.2 HTTP Compression

The moment you can ensure your users won't have trouble at the moment of decompressing your responses, it should be a must to use HTTP compression while developing a backend of an application that requests big amounts of data. Not only it improves the user experience by decreasing its latency, but also translates onto a decrease of costs on serviced that tax you by how much you use a given bandwidth.

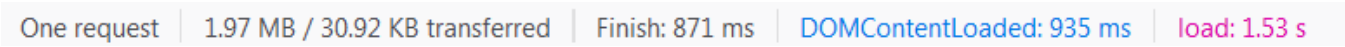


Figure 4.2: Result of a compressed response.

4.3 Reactive APIs

Nowadays, Reactive programming is no second to nothing when speaking of web development. Its an absolute great way to create interactive webs and platforms and, combined with Webhooks, can make the integration between a frontend and its backend a simple task of writing a couple of lines.
That said, it's important to note that when working with asynchronous code, you must be extremely aware of what's happening at each moment, so that you don't make your threads jump between lines of code while a necessary process is not finished yet.
In conclusion, it is a extremely powerful technology that need to be handled with care.

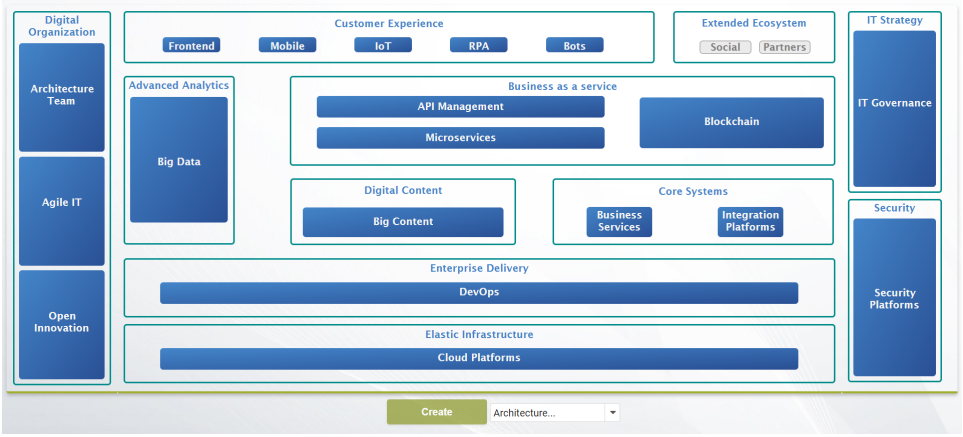


Figure 4.3: Reactive application

4.4 Server-Sent Events

Data streaming is a great way of performing real-time actions and monitoring, though it may have its downsides when using them on web applications. It's a great tool to use if you have to be constantly sending data, and it becomes more effective the more frequent it is. In example, you could use these tools in case you wanted to send a video file or a long text file from an application to another, so that its upload time becomes as short as possible. In general, its a tool to have in mind and that can solve problems when trying to send constant and big amounts of data.

4.5 GraphQL

GraphQL have proven to be a significantly better fit than REST for web applications with complex data requirements. Moving the queries for data into the components that consume it results in cleaner, more modular application code, though at the cost of some redundancy. Defining the shape of the returned data means that you get excellent tooling and guarantees about data structure without having to implement complicated response parsing and checking.
That being said, GraphQL doesn't have a good error handling system. Current logs don't include specific descriptions of the different errors. Also, even if it is not a big issue since most APIs already take into account a bit of redundancy on their object's creation, it must be noted that there is no way to escape from declaring each of them at least 2 times when using GraphQL, one as a type and the other as a database entity.
In general its a technology that can be interesting to take into account but that has a pretty high learning curve regarding its results. Luckily, Apollo gives some essential help to minimize the cost of understanding it.

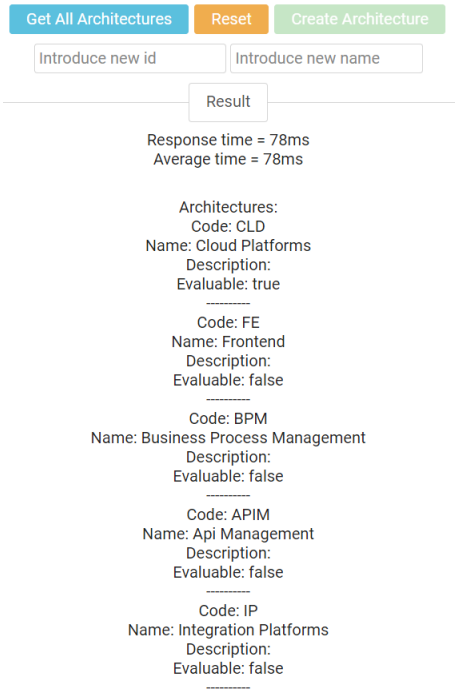


Figure 4.4: GraphQL based webpage.

4.6 Kafka Streams

Kafka Streams have proven to be a really interesting and powerful technology to intercommunicate servers and databases. The fact that it allows all of them to follow certain streams at a time makes it even better than using API gateways in many cases. Though it must be noted that it is not thought to be used to integrate frontends, it offers an easy way to keep track of operations done in other servers and to interconnect their codes.
That being said, it must be noted that Kafka Streams are real-time streams, and thus it must be taken onto account the same upsides and downsides that we considered when establishing those types of connections.

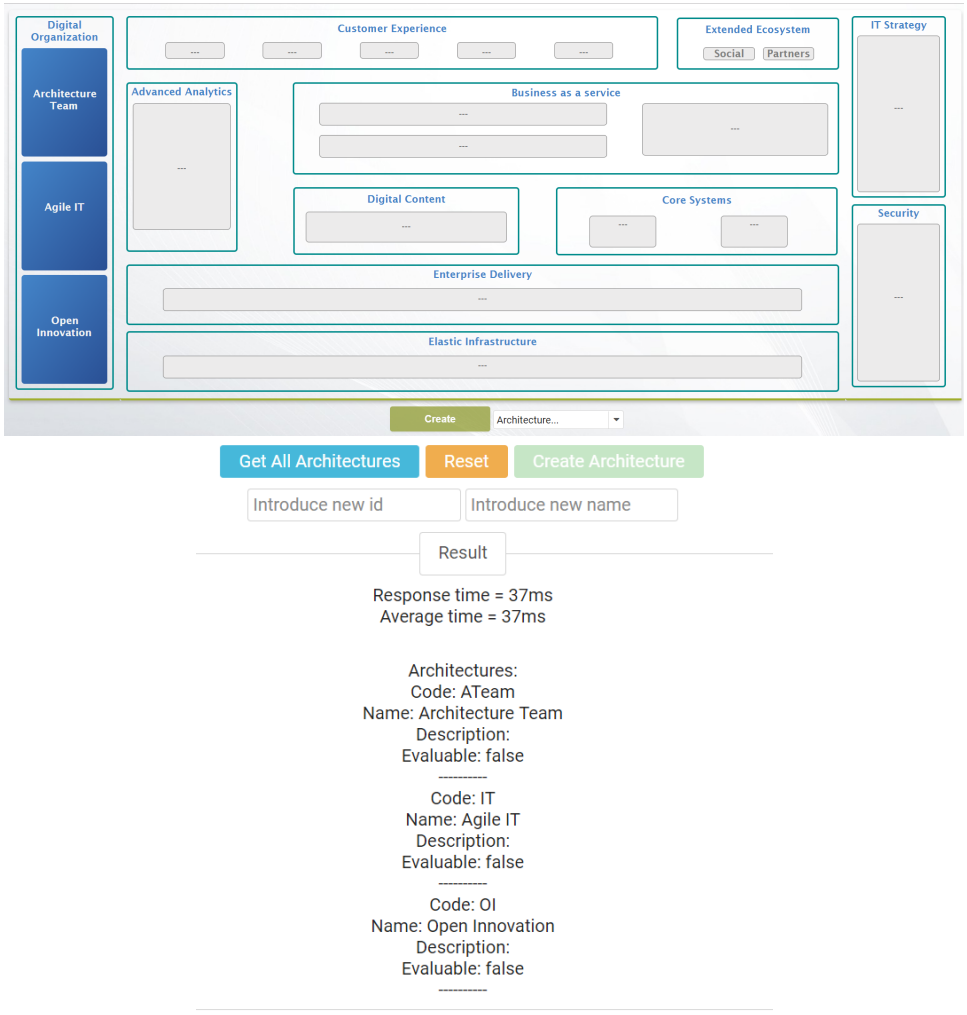


Figure 4.5: Synchronized applications using Kafka Streams.

4.7 Kong

Kong has proven to be a really versatile technology for an API gateway. The most remarkable thing of it is how easy it is to work with. After running the instance all you have to do is send a limited and very understandable set of requests to configure it and its plugins and you are ready to provide and monitor a service without much trouble. The only downside is the fact that it forces you to secure the different endpoints one by one and makes it a bit more long than just implementing their security on the server itself, but it's not that big of a problem and the upsides easily eclipse it. Definitely an easy and reliable technology.

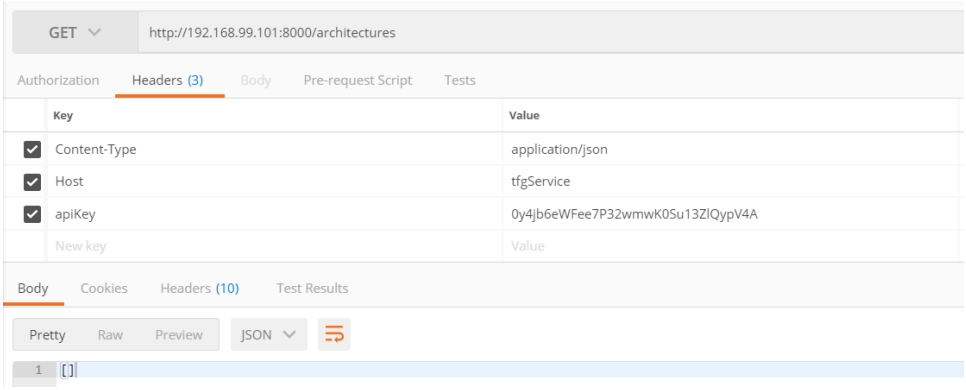


Figure 4.6: Result of GET request through API Gateway.

4.8 Tyk

Tyk has proven to be an easy an interactive gateway with many functionalities to connect and secure our APIs.

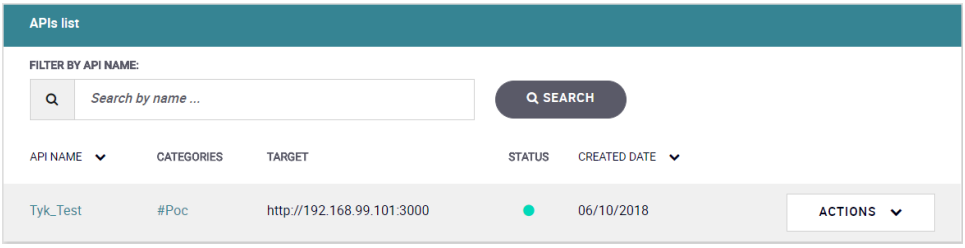


Figure 4.7: Running Tyk API.

4.8.1 Kong vs Tyk

It's hard to compare these two API Management tools. Both of them have probed to be extremely useful when implementing security and monitoring of APIs. For my own experience, Kong appears to be extremely easy to use via requests and the number of plugins it offers makes it a strong and flexible tool. Tyk requests to configure and administrate APIs are more complex to perform but it solves this problem by adding a very intuitive and user-friendly dashboard with a bunch of tools to monitor and secure the APIs already implemented. In terms of plugins, Kong is not second to anything but Tyk has more features by default. At the very end, both of them are good enough to be considered and the main factor to take into account when deciding which one to use is its pricing.
For a more extend comparison between both of them **BBVA** provides one taking into account all the previous factors, including their billing. You can check it out on this web [93].

4.9 Integration Platform as a Service (iPaaS)

When organizations are looking for a greater depth of capability, iPaaS is a credible option. It includes tools for developing and testing applications, middleware tools, and tools for deployment and server/system management. But, although iPaaS is growing in ability and popularity, it still requires planning and evaluation before deciding to use it. It works well in simple scenarios, where speed to release is a key requirement, but for larger systems that are heavy with embedded legacy server systems and applications, iPaaS implementations are a struggle to complete.
That being said, as we saw, there are many other ways to use cloud services to integrate applications, and even if the solutions with simple API Gateways are more limited than performing a greater scale system, they tend to also be a bit cheaper to implement.

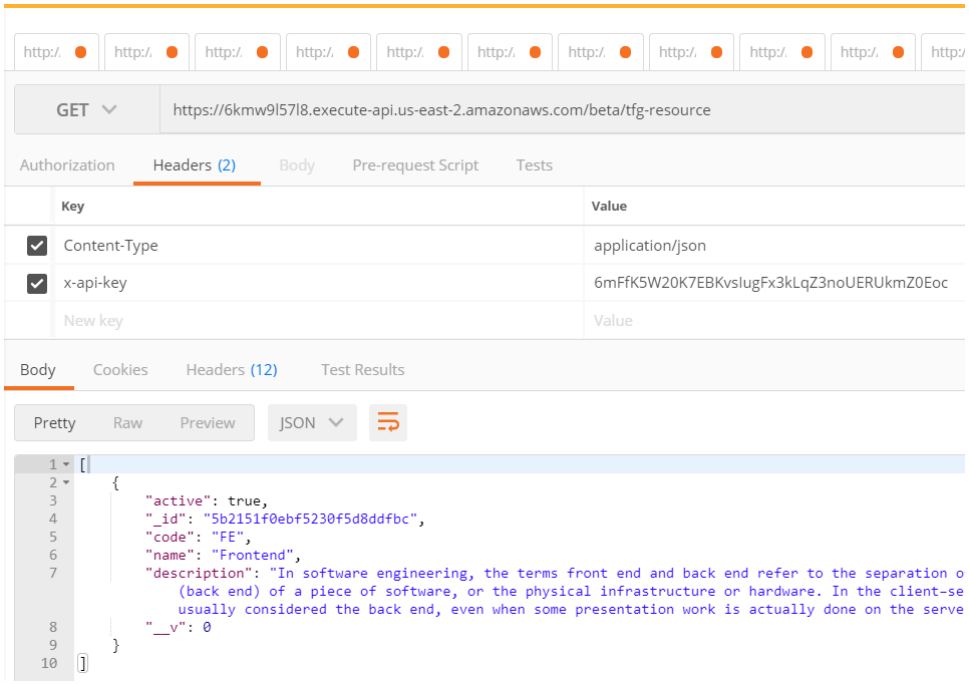


Figure 4.8: Result of GET request to public API Gateway.

Chapter 5

Budget

The proofs of concepts detailed on this document where all developed either free software or the free layer of services such as AWS. Thus, we will only take into account the salary of a junior engineer, around 23000€ per year with 40hour/week. This project had a dedication of around 50 hours/week and its duration was 18 weeks. Since there are 14 pays per year, if we consider that months have 4 weeks, we get a salary per hour of 10,26€/hour. This leaves us with the following table:

Concept	Cost
Junior engineer	9241,07 €

Chapter 6

Conclusions and future development

6.1 Conclusions

This project allowed us to see the large amount of options there are to allow data communication between our systems. We have gone from the use of methods to improve the performance of a connection between a frontend client to a backend server such as HTTP compression, Webhooks, Reactive Programming or GraphQL to systems that permits us to integrate a huge amount of separated services, databases and clients such as Kafka Streams, Api Management tools like Kong or Tyk, and Cloud Services like iPaaS or AWS API Gateway.
Each of them has its own benefits in their specific scenarios, and it is worth checking which ones to choose before starting to develop a project or trying to integrate two separate ones.
As a final statement, a combination that has proven to be pretty safe when integrating projects is using the combination Webhook - AWS Gateway - Reactive API, and make the desired modifications from there. HTTP compression is a good addition to this combination as long as it can be ensured that the clients won't have trouble to decompress the data. The addition of Kafka Streams would be recommended for projects based in Java that require an almost constant communication between several services for real-time communication. Regarding the other API Gateways, they are also good solutions, and would be an acceptable substitute from the AWS Gateway, though if the systems we want to integrate are hosted in AWS it is more recommended to use its own gateway. Finally, it could be good to try substituting the regular way of programming for GraphQL as an innovation project, though its learning curve makes it not as attractive yet, compared to other Reactive APIs.

6.2 Future development

This project opens a wide range of windows for future development. The possibilities are immense. The most direct thought after reading this document could be to choose one or several of the technologies learned to develop a completely new application. Other possibilities include the study of many other technologies that couldn't be included in it. Since the sector keeps evolving year by year, there would be always some new ones to test. An example of a technology that couldn't make it to enter this document is OData, which has a similar use case as GraphQl. As a final thought, it could be possible to offer the possibility to a company or university project to optimally integrate their systems using these technologies, including the addition of services provided for external projects such as Google Maps geolocation or Facebook's login page. Since the range of the ideas commented on this project are quite wide, its use cases are innumerable.

Bibliography

[1] Thoughtworks.com. (2018). Creative technology consultants | ThoughtWorks. [online] Available at: https://www.thoughtworks.com [Accessed 22 Mar. 2018].

[2] Docs.mongodb.com. (2018). Introduction to MongoDB − MongoDB Manual 3.6. [online] Available at: https://docs.mongodb.com/manual/introduction/ [Accessed 8 Mar. 2018].

[3] Docs.mongodb.com. (2018). Install MongoDB — MongoDB Manual 3.6. [online] Available at: https://docs.mongodb.com/manual/installation/ [Accessed 19 Mar. 2018].

[4] Robomongo.org. (2018). Robo 3T - formerly Robomongo — native MongoDB management tool (Admin UI). [online] Available at: https://robomongo.org [Accessed 20 Mar. 2018].

[5] Docs.mongodb.com. (2018). mongorestore — MongoDB Manual. [online] Available at: https://docs.mongodb.com/manual/reference/program/mongorestore/ [Accessed 13 Jun. 2018].

[6] Node Foundation. (2018). Node.js. [online] Node.js. Available at: https://nodejs.org/en/ [Accessed 21 Feb. 2018].

[7] Hope, C. (2018). How to set the path in Microsoft Windows. [online] Computerhope.com. Available at: https://www.computerhope.com/issues/ch000549.htm [Accessed 19 Mar. 2018].

[8] sigoa (2018). Express/Node introduction. [online] MDN Web Docs. Available at: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction [Accessed 20 Mar. 2018].

[9] DigitalCrafts. (2018). Student Blog: What is Postman, and why use it?. [online] Available at: https://www.digitalcrafts.com/blog/student-blog-what-postman-and-why-use-it [Accessed 29 May 2018].

[10] Angular.io. (2018). Angular Docs. [online] Available at: https://angular.io/docs [Accessed 19 Mar. 2018].

[11] sinedied (2017). The Missing Introduction to Angular and Modern Design Patterns. [online] Medium. Available at: https://medium.com/ngx-rocket/the-missing-introduction-to-angular-and-modern-design-patterns-43e8815c2801 [Accessed 19 Mar. 2018].

[12] Angular-2-training-book.rangle.io. (n.d.). Introduction · Rangle.io : Angular 2 Training. [online] Available at: https://angular-2-training-book.rangle.io/ [Accessed 21 Mar. 2018].

[13] Vega, J. (2017). Client-side vs. server-side rendering: why it's not all black and white. [online] freeCodeCamp. Available at: https://medium.freecodecamp.org/what-exactly-is-client-side-rendering-and-hows-it-different-from-server-side-rendering-bd5c786b340d [Accessed 21 Mar. 2018].

[14] Docker. (2018). What is Docker?. [online] Available at: https://www.docker.com/what-docker [Accessed 16 Mar. 2018].

[15] Docker Documentation. (2018). Get Started, Part 1: Orientation and setup. [online] Available at: https://docs.docker.com/get-started/ [Accessed 22 Mar. 2018].

[16] Sajid, N. (2017). Dockerising a Node.js and MongoDB App – Statuscode – Medium. [online] Medium. Available at: https://medium.com/statuscode/dockerising-a-node-js-and-mongodb-app-d22047e2806f [Accessed 23 May 2018].

[17] Store.docker.com. (2018). Docker Store. [online] Available at: https://store.docker.com/search?type=edition&offering=community [Accessed 23 May 2018].

[18] Docker Documentation. (2018). Install Docker Toolbox on Windows. [online] Available at: https://docs.docker.com/toolbox/toolbox_install_windows/ [Accessed 23 May 2018].

[19] Docker Forums. (2018). Pre-create check failed when first time launch Docker Quickstart Terminal. [online] Available at: https://forums.docker.com/t/pre-create-check-failed-when-first-time-launch-docker-quickstart-terminal/9977 [Accessed 23 May 2018].

[20] GitHub. (2018). docker/kitematic. [online] Available at: https://github.com/docker/kitematic/wiki/Common-Proxy-Issues-&-Fixes [Accessed 23 May 2018].

[21] Medium. (2018). Angular in Docker with Nginx, supporting environments, using Docker multi-stage builds. [online] Available at: https://medium.com/@tiangolo/angular-in-docker-with-nginx-supporting-environments-built-with-multi-stage-docker-builds-bb9f1724e984 [Accessed 25 May 2018].

[22] Foundation, N. (2018). Dockerizing a Node.js web app | Node.js. [online] Node.js. Available at: https://nodejs.org/en/docs/guides/nodejs-docker-webapp/ [Accessed 25 May 2018].

[23] NGINX. (2018). What is NGINX? - NGINX. [online] Available at: https://www.nginx.com/resources/glossary/nginx/ [Accessed 25 May 2018].

[24] Keough, K. (2015). Setting up MongoDB Instance with Docker Toolbox. [online] Code Hangar - Orlando Custom Full-Stack Web App Development. Available at: https://codehangar.io/mongodb-image-instance-with-docker-toolbox-tutorial/ [Accessed 27 May 2018].

[25] Ezequiel, P. (2017). Creating a docker image with MongoDB – Pablo Ezequiel – Medium. [online] Medium. Available at: https://medium.com/@pablo_ezequiel/creating-a-docker-image-with-mongodb-4c8aa3f828f2 [Accessed 28 May 2018].

[26] Hoffman, C. (2012). How to Forward Ports to a Virtual Machine and Use It as a Server. [online] Howtogeek.com. Available at: https://www.howtogeek.com/122641/how-to-forward-ports-to-a-virtual-machine-and-use-it-as-a-server/ [Accessed 27 May 2018].

[27] Amazon Web Services, Inc. (2018). What is AWS? - Amazon Web Services. [online] Available at: https://aws.amazon.com/what-is-aws/?nc1=h_ls [Accessed 20 Mar. 2018].

[28] TechPrimers (2018). What is AWS? | Amazon Web Services. [online] YouTube. Available at: https://www.youtube.com/watch?v=8xm8uraSxLY [Accessed 22 Mar. 2018].

[29] www.tutorialspoint.com. (2018). Amazon Web Services Cloud Computing. [online] Available at: https://www.tutorialspoint.com/amazon_web_services/amazon_web_services_cloud_computing.htm [Accessed 11 May 2018].

[30] www.tutorialspoint.com. (2018). Amazon Web Services Elastic Compute Cloud. [online] Available at: https://www.tutorialspoint.com/amazon_web_services/amazon_web_services_elastic_compute_cloud.htm [Accessed 11 May 2018].

[31] Okulewicz, M. (2017). Deploy Angular 2 + node.js website using AWS – codeburst. [online] codeburst. Available at: https://codeburst.io/deploy-angular-2-node-js-website-using-aws-1ac169d6bbf [Accessed 11 Jun. 2018].

[32] Buonincontri, S. (2017). angular 5 aws api gateway tutorial. [online] YouTube. Available at: https://www.youtube.com/watch?v=bbRA8LtshkU [Accessed 11 Jun. 2018].

[33] Docs.aws.amazon.com. (2018). Tutorial: Setting Up Node.js on an Amazon EC2 Instance - AWS SDK for JavaScript. [online] Available at: https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/setting-up-node-on-ec2-instance.html [Accessed 12 Jun. 2018].

[34] Scotch. (2018). Deploying a MEAN App to Amazon EC2 (Part 1). [online] Available at: https://scotch.io/tutorials/deploying-a-mean-app-to-amazon-ec2-part-1 [Accessed 12 Jun. 2018].

[35] Docs.mongodb.com. (2018). Install MongoDB Community Edition on Amazon Linux — MongoDB Manual. [online] Available at: https://docs.mongodb.com/manual/tutorial/install-mongodb-on-amazon/ [Accessed 12 Jun. 2018].

[36] Mobaxterm.mobatek.net. (2018). MobaXterm free Xserver and tabbed SSH client for Windows. [online] Available at: https://mobaxterm.mobatek.net/ [Accessed 12 Jun. 2018].

[37] Rodi, A. (2017). Deploy Angular2 on AmazonS3 – Alessandro Rodi – Medium. [online] Medium. Available at: https://medium.com/@coorasse/deploy-angular2-on-amazons3-5bbc040ab64a [Accessed 12 Jun. 2018].

[38] En.wikipedia.org. (2018). HTTP compression. [online] Available at: https://en.wikipedia.org/wiki/HTTP_compression [Accessed 14 Feb. 2018].

[39] Kanjilal, J. (2017). Compressing Web API responses to reduce payload. [online] InfoWorld. Available at: https://www.infoworld.com/article/3174597/application-development/compressing-web-api-responses-to-reduce-payload.html [Accessed 14 Feb. 2018].

[40] McLaughlin, T. (2017). The Benefits and Drawbacks of HTTP Compression. 1st ed. [ebook] Bethlehem: Lehigh University, p.1. Available at: https://pdfs.semanticscholar.org/84ea/367d4a3f4cd5eaf4c77536165bf3bae842ff.pdf [Accessed 21 Jun. 2018].

[41] Markheath.net. (2017). A Basic Introduction to Webhooks. [online] Available at: http://markheath.net/post/basic-introduction-webhooks [Accessed 14 Feb. 2018].

[42] Ajmera, S. (2017). Creating a NodeJS based Webhook for Intelligent Bots. [online] Chatbot's Life. Available at: https://chatbotslife.com/creating-a-nodejs-based-webhook-for-intelligent-bots-a91ecbe33402 [Accessed 16 Mar. 2018].

[43] MDN Web Docs. (2018). Cross-Origin Resource Sharing (CORS). [online] Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS [Accessed 6 Apr. 2018].

[44] Postman. (2018). Postman. [online] Available at: https://www.getpostman.com/ [Accessed 6 Apr. 2018].

[45] Lobo, J. (2017). What is a Webhook and how can I use them at my company?. [online] Inbenta. Available at: https://www.inbenta.com/en/blog/webhook-api-chatbot-nlp/ [Accessed 21 Jun. 2018].

[46] Staltz, A. (2014). The introduction to Reactive Programming you've been missing. [online] Gist. Available at: https://gist.github.com/staltz/868e7e9bc2a7b8c1f754 [Accessed 14 Feb. 2018].

[47] En.wikipedia.org. (2018). Observer pattern. [online] Available at: https://en.wikipedia.org/wiki/Observer_pattern [Accessed 14 Feb. 2018].

[48] Anon, (n.d.). Angular Docs. [online] Available at: https://angular.io/guide/dependency-injection.

[49] (2018). angular: HttpClient map response. [online] Stackoverflow.com. Available at: https://stackoverflow.com/questions/46422118/angular-httpclient-map-response [Accessed 16 Apr. 2018].

[50] Rxmarbles.com. (2018). RxMarbles: Interactive diagrams of Rx Observables. [online] Available at: http://rxmarbles.com/ [Accessed 22 Jun. 2018].

[51] Suteparuk, P. (2017). Understanding Reactive Programming Through an Example with RxJS (Part I). [online] Medium. Available at: https://medium.com/@psuteparuk/understanding-reactive-programming-through-an-example-with-rxjs-part-i-a69c6aa93a09 [Accessed 22 Jun. 2018].

[52] Brenkoweb.com. (2016). Introduction to server-sent events: - BrenkoWeb. [online] Available at: http://www.brenkoweb.com/tutorials/html5/html5-advanced/introduction-to-server-sent-events [Accessed 14 Feb. 2018].

[53] Streamdata.io. (2017). Push: SSE vs Websockets - Streamdata.io. [online] Available at: https://streamdata.io/blog/push-sse-vs-websockets/ [Accessed 4 May 2018].

[54] Recarey, A. (2017). WebSockets vs. Server-Sent events/EventSource. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/5195452/websockets-vs-server-sent-events-eventsource [Accessed 4 May 2018].

[55] Fidanov, S. (2015). How to Add Simple Real Time Streaming to Your Express App with Server Sent Events. [online] Terlici. Available at: https://www.terlici.com/2015/12/04/realtime-node-expressjs-with-sse.html [Accessed 4 May 2018].

[56] www.voorhoede.nl. (2018). Real-time communication with Server Sent Events | De Voorhoede. [online] Available at: https://www.voorhoede.nl/en/blog/real-time-communication-with-server-sent-events/ [Accessed 7 May 2018].

[57] Streamdata.io. (2017). Server-Sent Events explained with usecases. [online] Available at: https://streamdata.io/blog/server-sent-events/ [Accessed 8 May 2018].

[58] Almazán Cabo, A. (2017). Introducción a GraphQL | adictosaltrabajo. [online] Adictosaltrabajo.com. Available at: https://www.adictosaltrabajo.com/tutoriales/introduccion-a-graphql [Accessed 13 Feb. 2018].

[59] Byron, L. (2016). GraphQL: A query language for APIs. [online] Graphql.org. Available at: http://graphql.org/learn/best-practices/ [Accessed 15 Feb. 2018].

[60] Helfer, J. (2018). Tutorial: How to build a GraphQL server – Apollo GraphQL. [online] Apollo GraphQL. Available at: https://dev-blog.apollodata.com/tutorial-building-a-graphql-server-cddaa023c035?_ga=2.117504845.51125209.1519720607-1721446032.1519287669 [Accessed 15 Feb. 2018].

[61] Everitt, P. (2015). ES6 Imports with Babel — Paul Everitt documentation. [online] Pauleveritt.org. Available at: http://www.pauleveritt.org/articles/pylyglot/es6_imports/ [Accessed 26 Feb. 2018].

[62] Byron, L. (2016). GraphQL: A query language for APIs.. [online] Available at: http://graphql.org/graphql-js/ [Accessed 16 Feb. 2018].

[63] Byron, L. (2016). Graphql.org. (2018). GraphQL: A query language for APIs.. [online] Available at: http://graphql.org/graphql-js/type/ [Accessed 8 Mar. 2018].

[64] Afroza, Y. (2017). A CRUD app with Apollo, GraphQL, NodeJs, Express, MongoDB, Angular (v5). [online] CloudBoost. Available at: https://blog.cloudboost.io/a-crud-app-with-apollo-graphql-nodejs-express-mongodb-angular5-2874111cd6a5 [Accessed 8 Mar. 2018].

[65] Boubacar, B. (2018). Getting Started with GraphQL, Angular & Apollo Tutorial. [online] Howtographql.com. Available at: https://www.howtographql.com/angular-apollo/1-getting-started/ [Accessed 12 Mar. 2018].

[66] Apollographql.com. (2018). Apollo GraphQL. [online] Available at: https://www.apollographql.com/ [Accessed 12 Mar. 2018].

[67] Times Open. (2017). The New York Times — Now On Apollo – Times Open. [online] Available at: https://open.nytimes.com/the-new-york-times-now-on-apollo-b9a78a5038c [Accessed 15 Mar. 2018].

[68] Gayed, J. (2017). A Look at Apollo, From a Relay Perspective – On Frontend Engineering. [online] On Frontend Engineering. Available at: https://jeremygayed.com/a-look-at-apollo-from-a-relay-perspective-1c2d43b0a37a [Accessed 15 Mar. 2018].

[69] Neary, A. (2017). Rearchitecting Airbnb's Frontend – Airbnb Engineering & Data Science – Medium. [online] Medium. Available at: https://medium.com/airbnb-engineering/rearchitecting-airbnbs-frontend-5e213efc24d2 [Accessed 15 Mar. 2018].

[70] Odata.org. (2018). OData - the Best Way to REST. [online] Available at: http://www.odata.org/ [Accessed 23 Jun. 2018].

[71] Knoldus. (2017). Introducing Kafka Streams: Processing made easy. [online] Available at: https://blog.knoldus.com/2017/06/12/introducing-kafka-streams-processing-made-easy/ [Accessed 15 Feb. 2018].

[72] Docs.confluent.io. (2018). Kafka Streams Quick Start — Confluent Platform. [online] Available at: https://docs.confluent.io/current/streams/quickstart.html [Accessed 11 May 2018].

[73] Apache Kafka. (2018). Apache Kafka. [online] Available at: http://kafka.apache.org/documentation.html#introduction [Accessed 14 May 2018].

[74] Tiwari, G. (2018). Setting Up and Running Apache Kafka on Windows OS - DZone Big Data. [online] dzone.com. Available at: https://dzone.com/articles/running-apache-kafka-on-windows-os [Accessed 14 May 2018].

[75] Oracle.com. (2018). Java SE Runtime Environment 8 - Downloads. [online] Available at: http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html [Accessed 14 May 2018].

[76] Zookeeper.apache.org. (2018). Apache ZooKeeper - Releases. [online] Available at: http://zookeeper.apache.org/releases.html [Accessed 14 May 2018].

[77] Apache Kafka. (2018). Apache Kafka. [online] Available at: http://kafka.apache.org/downloads.html [Accessed 14 May 2018].

[78] Willig, D. (2017). Apache Kafka with Node.js. [online] Blog.mimacom.com. Available at: https://blog.mimacom.com/apache-kafka-with-node-js/ [Accessed 15 May 2018].

[79] Mitra, T. (2018). Getting started with NodeJS and Kafka. [online] Node Web Apps. Available at: https://nodewebapps.com/2017/11/04/getting-started-with-nodejs-and-kafka/ [Accessed 15 May 2018].

[80] Anon, (2018). [online] Available at: https://www.linkedin.com/pulse/introduction-kafka-using-nodejs-pankaj-panigrahi [Accessed 15 May 2018].

[81] Baeldung. (2018). An Introduction to Kong | Baeldung. [online] Available at: http://www.baeldung.com/kong [Accessed 13 Feb. 2018].

[82] Saxena, T. (2016). Getting Started With Application Authentication Via Kong API Gateway | TO THE NEW Blog. [online] TO THE NEW BLOG. Available at: http://www.tothenew.com/blog/getting-started-with-application-authentication-via-kong-api-gateway/ [Accessed 16 Mar. 2018].

[83] Kong. (2018). Open-Source API Management and Microservice Management. [online] Available at: https://getkong.org/docs/0.5.x/getting-started/introduction/ [Accessed 16 Mar. 2018].

[84] KongHQ. (2018). Installations - KongHQ. [online] Available at: https://konghq.com/install/ [Accessed 28 May 2018].

[85] Hub.docker.com. (2018). [online] Available at: https://hub.docker.com/_/kong/ [Accessed 28 May 2018].

[86] Irani, R. (2015). Docker Toolbox Setup — Windows – Romin Irani's Blog. [online] Romin Irani's Blog. Available at: https://rominirani.com/docker-toolbox-setup-windows-4d65c3f691eb [Accessed 30 May 2018].

[87] GitHub. (2017). feat(router) new API router with multiple hosts/paths and HTTP methods support by thibaultcha · Pull Request #1970 · Kong/kong. [online] Available at: https://github.com/Kong/kong/pull/1970 [Accessed 31 May 2018].

[88] Chrome.google.com. (2018). Postman Interceptor. [online] Available at: https://chrome.google.com/webstore/detail/postman-interceptor/aicmkgpgakddgnaphhhpliifpcfhicfo?hl=es [Accessed 1 Jun. 2018].

[89] KongHQ. (2018). Plugins - KongHQ. [online] Available at: https://konghq.com/plugins/ [Accessed 1 Jun. 2018].

[90] Bridge, T. (2018). Introduction - Tyk API Gateway and API Management. [online] Tyk API Gateway and API Management. Available at: https://tyk.io/blog/docs/tyk-multi-datacenter-bridge/introduction/ [Accessed 14 Feb. 2018].

[91] Tyk.io. (2018). With Docker. [online] Available at: https://www.tyk.io/docs/get-started/with-tyk-on-premise/installation/docker/ [Accessed 4 Jun. 2018].

[92] Tyk API Gateway and API Management. (2018). Set up your first API - Tyk API Gateway and API Management. [online] Available at: https://tyk.io/blog/docs/tyk-api-gateway-v1-9/tutorials/set-up-your-first-api/ [Accessed 6 Jun. 2018].

[93] Perez, A. and Evgeniev, M. (2018). API Gateways: Kong vs. Tyk | BBVA. [online] NEWS BBVA. Available at: https://www.bbva.com/en/api-gateways-kong-vs-tyk/ [Accessed 7 Jun. 2018].

[94] buger (2018). TykTechnologies/tyk-dashboard-docker. [online] GitHub. Available at: https://github.com/TykTechnologies/tyk-dashboard-docker [Accessed 8 Jun. 2018].

[95] Nair, P. (2017). An Introduction To iPaaS - What Is Integration Platform-As-A-Service?. [online] Built.io Blog. Available at: https://www.built.io/blog/an-introduction-to-ipaas [Accessed 14 Feb. 2018].

[96] Rouse, M. (n.d.). What is iPaaS (integration platform as a service)? - Definition from WhatIs.com. [online] SearchCloudApplications. Available at: http://searchcloudapplications.techtarget.com/definition/iPaaS-Integration-platform-as-a-service [Accessed 19 Mar. 2018].

[97] Rouse, M. (n.d.). What is B2B (business-to-business)? - Definition from WhatIs.com. [online] SearchCIO. Available at: http://searchcio.techtarget.com/definition/B2B [Accessed 19 Mar. 2018].

[98] Rouse, M. (n.d.). What is vendor lock-in? - Definition from WhatIs.com. [online] SearchConvergedInfrastructure. Available at: http://searchconvergedinfrastructure.techtarget.com/definition/vendor-lock-in [Accessed 19 Mar. 2018].

[99] Anon, (2018). [online] Available at: https://www.quora.com/What-is-the-best-way-to-implement-iPaaS-on-AWS [Accessed 20 Mar. 2018].

[100] Reichert, A. (2014). IPaaS integration: When, how and why to use iPaaS. [online] SearchCloudApplications. Available at: http://searchcloudapplications.techtarget.com/tip/IPaaS-integration-When-how-and-why-to-use-iPa S [Accessed 22 Mar. 2018].

[101] Graham, R., Liddick, W., Weil, D. and Newhart, J. (2018). Tying It All Together: Integration PaaS in the Next-Gen Enterprise. [online] Available at: https://er.educause.edu/articles/2018/2/tying-it-all-together-integration-paas-in-the-next-gen-enterprise [Accessed 11 May 2018].

[102] MuleSoft. (2018). What is iPaaS? Gartner Provides a Reference Model. [online] Available at: https://www.mulesoft.com/resources/cloudhub/what-is-ipaas-gartner-provides-reference-model [Accessed 31 May 2018].

[103] MSV, J., Staff, T. and Semenov, S. (2018). Five Reasons to Consider Amazon API Gateway for Your Next Microservices Project - The New Stack. [online] The New Stack. Available at: https://thenewstack.io/five-reasons-to-consider-amazon-api-gateway-for-your-next-microservices-project/ [Accessed 14 Jun. 2018].

[104] Letsencrypt.org. (2018). Let's Encrypt - Free SSL/TLS Certificates. [online] Available at: https://letsencrypt.org/ [Accessed 15 Jun. 2018].

[105] Aws.amazon.com. (2018). AWS Marketplace: Cloudreach Connect. [online] Available at: https://aws.amazon.com/marketplace/pp/B01FSYVIWG [Accessed 16 Jun. 2018].

[106] Google Docs. (2018). Copy of CR_CONNECTFlyers_Final_NoCrops.pdf. [online] Available at: https://drive.google.com/file/d/0B-VWd4nBbkVHMGhocHM4bWRnUWM/view [Accessed 17 Jun. 2018].

Appendix A

Workplan

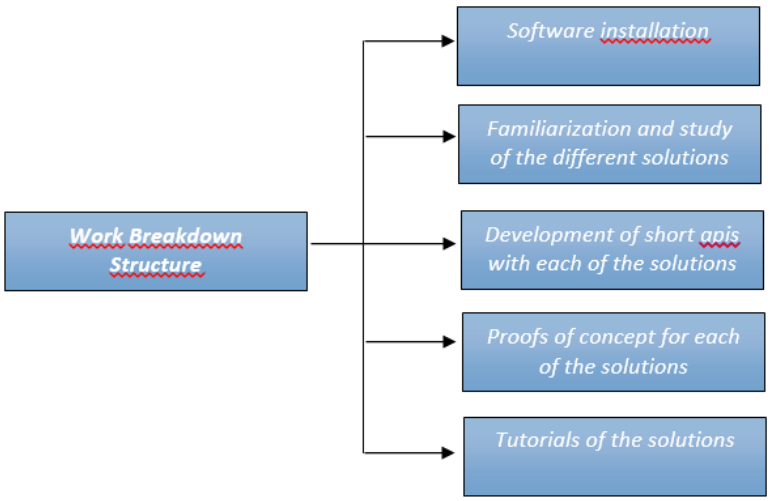


Figure A.1: Work Breakdown Structure

Work Packages

Project: Integration of business systems optimized with Cloud	WP ref: WP1	
Major constituent: Learning		
Short description: Installation of needed software	Planned start date:15/02/18 Planned end date:28/02/18	
	Start event:-- End event:--	
Internal task T1: Installation of uninstalled main software. NodeJs, Angular 4. Internal task T2: Installation of specific software for each solution. Mainly node-modules dependencies.	Deliverables:	Dates:

Project: Integration of business systems optimized with Cloud	WP ref: WP2	
Major constituent: Software		
Short description: Familiarization and study of the different solutions	Planned start date:20/02/18 Planned end date:15/04/18	
	Start event:-- End event:--	
Internal task T1: Study of GraphQl. Internal task T2: Study of Kong. Internal task T3: Study of TyK. Internal task T4: Study of Webhooks. Internal task T5: Study of Request Compression. Internal task T6: Study of Reactive APIs. Internal task T7: Study of iPaaS. Internal task T8: Study of Kafka Streams	Deliverables:	Dates:

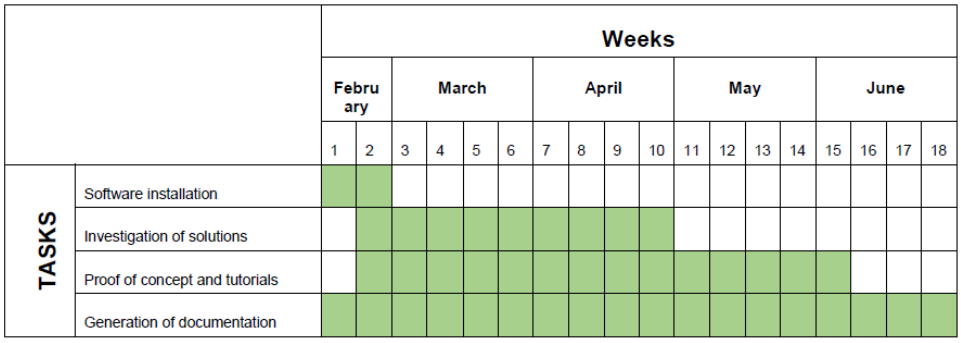
Project: Integration of business systems optimized with Cloud	WP ref: WP3	
Major constituent: Software		
Short description: Proof of concept and tutorials of each of the solutions.	Planned start date:20/02/18 Planned end date:30/05/18	
	Start event:-- End event:--	
Internal task T1: Creation of several angular-node-mongo generic projects. Internal task T2: Modification of the projects applying the different solutions. Internal task T3: Development of tutorials and practices	Deliverables:	Dates:

Project: Integration of business systems optimized with Cloud	WP ref: WP6	
Major constituent: Software		
Short description: Generation of documents	Planned start date:15/02/18 Planned end date:31/06/18	
	Start event:-- End event:--	
Internal task T1: Realization of project proposal and workplan. Internal task T2: Realization of critical review. Internal task T3: Realization of final report. Internal task T4: Realization of TFG memory. Internal task T5: Preparation of TFG presentation	Deliverables:	Dates:

Milestones

WP#	Task#	Short title	Milestone / deliverable	Date (week)
1	T1	Software installation	Install all the software needed to develop the project	2
2	T2	Study of solutions	Knowledge about all the technologies that are going to be used	10
3	T2	Tutorials and practices of solutions	Realization of tutorials and apis for each of the studied solutions	15
4	T1	Project proposal and workplan	Document del Project proposal and workplan	2
4	T3	Project critical review	Document del Project critical review	11
4	T4	Final Report	Document del Final Report	18
4	T4	Memòria TFG	Document final del TFG.	18
4	T4	Presentació final TFG	Document per a la presentació del TFG	18

Gantt Diagram



The technical project with all its sections is included in a separate document.

Glossary

Global:

- **API:** Applications Programming Interface.
- **REST:** Representation State Transfer.
- **JSON:** Javascript Object Notation.
- **JWT:** JSON Web Token.
- **CRUD:** Create, Retrieve, Update, Delete.

Amazon Web Services:

- **EC2:** Elastic Compute Cloud.
- **S3:** Simple Cloud Storage Service.

Digital Architectures:

- **Architecture Team:** Group of people in an organization that work with Digital Architectures.
- **Agile IT:** Agile software development describes an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer(s)/end user(s).
- **Open Innovation:** term used to promote an information age mindset toward innovation that runs counter to the secrecy and silo mentality of traditional corporate research labs.
- **Frontend:** in software engineering, the terms front end and back end refer to the separation of concerns between the presentation layer (front end), and the data access layer (back end) of a piece of software, or the physical infrastructure or hardware. In the client–server model, the client is usually considered the front end and the server is usually considered the back end, even when some presentation work is actually done on the server.
- **Mobile:** technology used for cellular communication.
- **Internet of Things (IoT):** network of physical devices, vehicles, home appliances, and other items embedded with electronics, software, sensors, actuators, and connectivity which enables these things to connect and exchange data, creating opportunities for more direct integration of the physical world into computer-based systems, resulting in efficiency improvements, economic benefits, and reduced human exertions.
- **Bots:** software application that runs automated tasks (scripts) over the Internet. Typically, bots perform tasks that are both simple and structurally repetitive, at a much higher rate than would be possible for a human alone.
- **Robot Process Automation (RPA):** emerging form of business process automation technology based on the notion of software robots or artificial intelligence (AI) workers.

- **Big Data:** data sets that are so voluminous and complex that traditional data-processing application software are inadequate to deal with them. Big data challenges include capturing data, data storage, data analysis, search, sharing, transfer, visualization, querying, updating, information privacy and data source.
- **API Management:** process of creating and publishing web APIs, enforcing their usage policies, controlling access, nurturing the subscriber community, collecting and analyzing usage statistics, and reporting on performance.
- **Microservices:** software development technique, a variant of the service-oriented architecture (SOA), architectural style that structures an application as a collection of loosely coupled services. In a microservices architecture, services are fine-grained and the protocols are lightweight.
- **Blockchain:** continuously growing list of records, called blocks, which are linked and secured using cryptography. Each block typically contains a cryptographic hash of the previous block, a timestamp, and transaction data. By design, a blockchain is resistant to modification of the data. It is an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way.
- **Big Content:** big files such as media and other types of documentation.
- **Business Services:** approach used to manage business IT services.
- **Integration Platforms:** computer software which integrates different applications and services.
- **DevOps:** software engineering culture and practice that aims at unifying software development (Dev) and software operation (Ops).
- **Cloud Platforms:** suite of cloud computing services that runs on the same infrastructure that the provider uses internally for its end-user products.
- **IT Governance:** process of governing a network.
- **Security Platforms:** computer software which protects applications and services.