Designing and implementing a monitoring solution for Web APIs

Bachelor Thesis

Author: Dragos Fotescu

Director: Jordi Marco Gómez

Co-Director: Marc Oriol Hilari

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Abstract

The number of APIs is growing consistently as more and more businesses integrate them and use them in their core business. That means that any degradation or downtime in their API could be crucial as could impact their customers or revenues.

As the API ecosystem has been growing, it is still missing better tooling for API developers, maintainers and operators. One of the missing things that would increase the overall quality of APIs is monitoring and observability.

This project showcases how the market still needs better tools for monitoring APIs and a proposal to make a language-agnostic with minimal integration effort possible solution.

Table of contents

1. Introduction

This bachelor thesis consists of implementing a programming language-agnostic solution for monitoring exposed Web APIs. For example, understanding how the users interact with the APIs, measuring the Quality of Service or helping to mitigate failures or service disruptions.

1.1. Context

In this section we are going to explain the context of the project, the problem and why it is important.

1.1.1. GENESIS Project

This thesis is framed inside one of the tasks to complete in the GENESIS project. GENESIS (Generation and Evolution of Smart APIs) is a project formed by two research groups from the UPC: GESSI (Software and Service Engineering Group) and DTIM (Database Technologies and Information Management Group).

GENESIS is a project funded by the National Spanish Program for Research Aimed at the Challenges of Society 2016. The objective of this project is to provide techniques and tools to monitor, improve and evolutionate APIs. [1]

I'm going to be contributing to one of the main objectives of GENESIS, which is to collect information, metadata, and behavior of the functionality to expose provided by API providers. I will develop a solution to monitor exposed APIs. This data will also be used to generate complete and accurate documentation, including non-functional requirements that will facilitate the adoption of the APIs by third parties.

1.1.2. APIs

An application programming interface (API) is an interface or communication protocol between a client and a server intended to simplify the building of client-side software. It has been described as a “contract” between the client and the server, such that if the client makes a request in a specific format, it will always get a response in a specific format or initiate a defined action. APIs simplifies the development efforts by abstracting the underlying implementation and only exposing objects or methods the developer needs.

There are different types of APIs (e.g. operating system, computer hardware, software library, browser, etc.) but we are going to focus on HTTP server-side web APIs.

Today, with the rise of REST and web services over HTTP, the term is often assumed to refer to APIs of such services when given no other context (Web APIs). [2]

1.1.3. Web APIs

A server-side web API publicly exposes one or more endpoints to a defined request-response message system, typically expressed in JSON or XML, which is exposed via the web most commonly via an HTTP-based web server.

The number of available web APIs has grown consistently over the past years, as businesses realize the growth opportunities associated with running an open platform that any developer can interact with. There are numerous businesses that their main service is to provide APIs.

The API directory of the *ProgrammableWeb* directory, a massive searchable directory of web APIs, reached the 22,000-APIs mark in June 2019, up to 2,000-APIs from 2010 [3].

This number is only accounting for some part of Public APIs but it's not taking into consideration Partner APIs (exposed to business partners, customers, premium users, etc.) or Private APIs (exposed to internal systems, usually just for consumption inside the company).

1.1.4. Problem

As web APIs are becoming more critical to more business, it's also critical to make sure that the APIs you are exposing are running smoothly. API failures or disruptions can drastically affect customers or end-users. It's hard to quantify how much downtime affects, but Gartner released a study in 2014 which estimated that businesses lose an average of $300,000 per hour of downtime [4]. Hence it's more important than ever to have a reliable monitoring system for the provided APIs to help ensure the highest quality of service and minimize the risks of downtimes or failures.

Even though web APIs have experienced such a steep growth, solutions for monitoring exposed APIs lagged behind in the market compared to solutions for monitoring other kinds of systems (e.g. Google Analytics or Cloudflare).

2. Stakeholders

Now that the project was introduced. In this part are defined the stakeholders, which means all the people that are related to the project.

- **Director and co-director of the project:** The director, Jordi Marco Gómez is part of the UPC Computer Science Department, and the co-director Marc Oriol Hilari is part of the UPC Department of Service and Information System Engineering (ESSI). Both the director and co-director are also members of the Generation and Evolution of Smart APIs (GENESIS) project. They will be in charge of guiding and supervising the project and also propose changes or improvements to the project if they see it convenient.
- **Product and tech team:** This team will design and implement the solution for the project and assure its correct behavior. As this is a bachelor thesis, I will be the one doing all the tasks related to the product and tech team roles (Software Engineer, Product Manager, Designer, Infrastructure, Tester, etc.).
- **API providers:** Those will be the ones adopting the monitoring solution for APIs built in this project. Although it was proposed to be used with APIs exposed from the University, the solution should be adaptable and open-sourced so that everyone would be able to use it.
- **API users:** Those are the final users and consumers of the API provided by the API providers. They will be affected indirectly due to the consequences of monitoring the service they are somehow consuming or using.

3. State of the art

Nowadays there are already some solutions on the market that people are using to monitor their APIs. The main alternatives are commercial software that would get expensive to use and usually, they are not open-source. Here I'm going to list and analyze the main ones:

- **Apigee**. Apigee is an API management company that was acquired by Google. Besides other features, it offers API monitoring. The downsides of Apigee are that it's targeted at enterprise customers, which means that is cost-prohibitive for smaller companies, and it's not open source. That would make it impossible to use in the UPC context. [5]
- **Datadog**. It is a commercial monitoring service for applications, providing monitoring of servers, databases, and services. It's not specifically targeted at APIs, but it could be used for monitoring those as well. It works with small agents for different frameworks, programming languages or infrastructures hence that would not fulfill our requirements of being programming language agnostic. [6]
- **Elastic Uptime**. It is a commercial product where you can define your API endpoints, a request type, the expected response, and a time interval. Elastic Uptime will try the specified request in the time interval and monitor its results. It's a nice solution but you lose flexibility because you have to set up and maintain every endpoint. Furthermore, this solution is not monitoring all the requests/responses coming from the users, just the ones initiated by ElasticSearch Uptime. It has a cloud and a self-hosted version. [7]

- **Manual monitoring**. This is the most customizable solution but it’s also very expensive and time-consuming to implement. It consists of in-house software solutions tailored to its needs. Usually, only bigger companies are able to implement and deploy such solutions.

4. Objectives and requirements

The main objective of this project is to design and implement a system that allows API providers to monitor metrics and usage from the exposed APIs and a dashboard to visualize those results.

An important requirement for this project is that it must be language-agnostic, which means that the solution must work independently of the programming language the Web API was developed in and facilitate an easy integration with the monitored server.

Important metrics that we would like to monitor and visualize:

- **Response time**: the time it takes to the monitored server to respond to a request.
- **Availability**: it determines if the monitored server is running, serving the requests or returning a response.
- **HTTP Errors**: it will capture responses in which their status codes are greater than 399. Status codes greater than 399 represent failed requests or some other kind of error. Usually 4XX errors are reserved for errors caused by the client and 5XX are codes reserved for errors caused by the server.
- **Throughput**: number of requests served by the monitored system in a period of time. It is usually measured in requests per second.

If we are well on time, it would be nice to have:

- An alerting system such that the maintainers of the monitored systems could be notified when certain events happen (e.g. no availability or latency >300ms).

- Mean Time To Repair metric, which is the mean transcurred time between the unavailability or disruption of a service and its normal behavior.

We decided that the final solution would be to have a proxy that intercepts and forwards all the requests and responses. That would allow us to analyze, measure and calculate the different metrics explained in this section.

Besides that, we have to consider several important non-functional requirements for the system:

- **Performance**: using this system should not compromise the response time of the API significantly.
- **Availability**: the system should be highly-available. It shouldn't impact the availability of the monitored APIs.
- **Security**: the solution should be secure. Compromising the monitoring system may lead to compromising the exposed API's security.
- **Usability**: the solution should be easy to set up and use.
- **Open source**: the solution should be open source so the community can benefit from it and help to improve it.

5. Working methodology

Since this will be a project where the tasks will be dynamically evolving as we move within the development phase, it would be great to follow an agile methodology. In this case, I will follow the Kanban methodology.

Kanban is a popular framework used to implement agile software development. It requires full transparency of work and real-time communication. Work items are represented visually on a kanban board, allowing team members to see the state of every piece of work at any time. [8]

Using Kanban, we will have the following columns on the board. The different development tasks will pass through the following workflow:

1. **Backlog.** Are all the identified tasks that need to be done within the project. We will be adding tasks to the backlog as soon as we identify them.
2. **Analysis.** This phase is used to analyze and design the identified tasks from a software engineering perspective.
3. **In progress.** The actual implementation of the tasks.
4. **Testing.** Validating and testing that the task matches the expected behavior and complies with its requirements. In case it doesn’t, the task will go back to the previous phase.
5. **Done.** The task is considered as finished.

I will also use unit testing, a software testing method that tests individual units of code, to validate the critical features of the project.

As the tasks are completed, we will run all the changes through the Continuous Integration pipelines to identify other possible defects in the software as we iterate it and that we don't break any piece from the previous work we've done. Continuous Integration is a development practice that requires developers to automatically integrate and test the shared code each time this is modified, in our case, we will be running our unit tests and building all the components to see that everything is still fine after each change.

5.1. Tools

To stick to the working methodology described earlier, I will be using the following tools:

- **Trello.com**. Trello is a simple service for creating virtual boards. I will use Trello to create and maintain the Kanban board previously described.
- **GitHub**. GitHub is a service that provides hosting for software development version control using Git. It will host all the code related to the project.
- **CircleCI**. It is a Continuous Integration platform. It will listen to all changes pushed to GitHub and will run all the checks needed to validate the new changes.

6.Time Planning

Next, it will describe the time planning. This project has an estimated duration of 5 months, from early September to early February. The equivalent workload is equivalent to around 500 hours, covering all the necessary tasks for its realization, from the final memory to the actual implementation and the oral defense.

This workload will be assigned to a sole person that will do all the required roles to accomplish the task. That means that it will not be possible to fully parallelize most of the tasks and a delay in a task will directly affect the planning of all the subsequent tasks.

It should be noted that the time planning presented below may be changed or not strictly followed because an Agile methodology is being used. New or changed tasks or dates may appear, even so, I will try to adjust to the planning as much as possible.

6.1. Tasks description

The project will have 4 parts, each one with its own tasks and it’s estimated duration in hours.

6.1.1. Initial Phase (Project Management)

The main objective of this section is to determine the basic aspects of this project: the objectives, the requirements, the planning, the working methodology, the definition and the viability of the project. This phase will take around a month and it will consist of the following tasks:

- PM1. Research about monitoring web-servers and APIs technologies. (10h)
- PM2. Meetings with stakeholders. (1h)
- PM3. Contextualization and scope definition. (10h)
- PM4. Temporal planning and the tasks needed for the project. (8h)
- PM5. Budget and sustainability management. (6h)

6.1.2. Design

In this phase, it will be mainly to design the software architecture from a high-level perspective. It will also be the stage to decide the specific components of the applications, which programming languages to use, which databases or other specifications. It will consist of the following tasks:

- D1. Design the software architecture of the monitoring system. (15h)
- D2. Define the environment, technologies, and infrastructure used for the project. (15h)
- D3. Document all the advancements from this phase. (10h)

6.1.3. Development

In the stage, I will implement and develop the multiple components of the application designed and planned from the previous stages. The tasks of the development phase will be:

- DEV1. Implement and set up the infrastructure and environments (development and production) to support the application. (15h)
- DEV2. Develop the proxy component allowed to intercept and forward network requests and responses. (25h)
- DEV3. Develop and integrate a persistence system to store all the events. (40h)
- DEV4. Create deployment scripts to enable an easy deploy of the application to a new environment. (10h)
- DEV5. Implement the software that should calculate metrics from the events. (25h)
- DEV6. Implement SSL certificate handling. The proxy should support the HTTPS protocol. (35h)
- DEV7. Create and expose an API to be able to access the persisted data. (30h)
- DEV8. Implement a dashboard to visualize the metrics. This task will take longer than usual because it would need to implement the whole user interface, implement all the front-end logic and connect it to the internal API. (50h)
- DEV9. Implement authentication and authorization to the application. (20h)
- DEV10. Add integration tests to ensure correct functionality and maintainability. (20h)
- DEV11. Extension of the application, depending on the needs and temporal availability. (20h)

- DEV12. Document all the advancements from this phase in the final memory. (20h)

6.1.4. Final Phase

Finally, we identify the final phase where we should have already finished the development of the platform. I will ensure the correct functionality and polish the final details of it. I should also complete and conclude the final documentation and presentation of the final defense.

- F1. Quality Assurance, testing, bugs search and validating the initial requirements and objectives. (10h)
- F2. Fixing bugs or implementation issues. (15h)
- F3. Complete and conclude the final memory. (10h)
- F4. Prepare the final presentation and defense of the project. (15h)

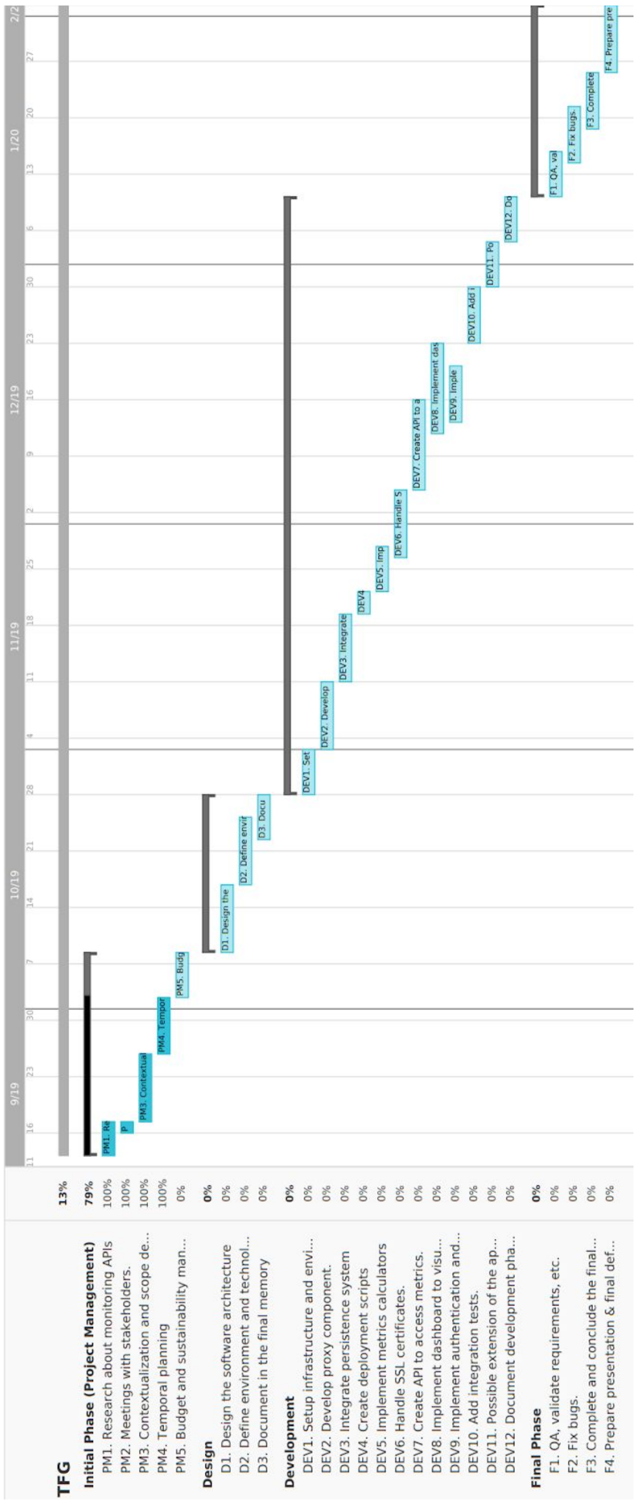
6.2. Risks and possible obstacles

As in all projects, there are risks and possible obstacles that we should be aware of. I have reserved some hours dedicated to the main risks so in case it appears an obstacle, I could have some time to react. The main possible obstacles to keep in mind in this project are:

- **Bad decisions:** it can arise problems from bad choices or decisions that may have a big negative impact on the development phase of the project. To mitigate that risk we must make thorough research and analyze different options before making important decisions. I have reserved 40 hours from the total workload for mitigating this risk.

- **Technical challenges:** this monitoring solution can be a big technical challenge to solve with many different parts and technologies to implement. It is important to leverage self-learning. I have reserved 40 hours dedicated to self-learning and researching.
- **Limited time:** the bachelor thesis has a fixed delivery date scheduled having a tight timeline to develop and deliver the project. That means that we must plan accordingly and that any unexpected problems that may arise have to be solved as flexible and quickly as possible.

6.3. Gantt Diagram



TFG	**13%**
Initial Phase (Project Management)	**79%**
PM1. Research about monitoring APIs	100%
PM2. Meetings with stakeholders.	100%
PM3. Contextualization and scope de...	100%
PM4. Temporal planning	100%
PM5. Budget and sustainability man...	0%
Design	**0%**
D1. Design the software architecture	0%
D2. Define environment and technol...	0%
D3. Document in the final memory	0%
Development	**0%**
DEV1. Setup infrastructure and envi...	0%
DEV2. Develop proxy component.	0%
DEV3. Integrate persistence system	0%
DEV4. Create deployment scripts	0%
DEV5. Implement metrics calculators	0%
DEV6. Handle SSL certificates.	0%
DEV7. Create API to access metrics.	0%
DEV8. Implement dashboard to visu...	0%
DEV9. Implement authentication and...	0%
DEV10. Add integration tests.	0%
DEV11. Possible extension of the ap...	0%
DEV12. Document development pha...	0%
Final Phase	**0%**
F1. QA, validate requirements, etc.	0%
F2. Fix bugs.	0%
F3. Complete and conclude the final...	0%
F4. Prepare presentation & final def...	0%

Gantt diagram [9]

7. Resources

7.1. Human Resources

Although there are different roles needed for completing the project and its tasks, I will only dispose of a sole Software Engineer, myself, for accomplishing all the tasks. I will also perform different roles like Product Manager, Designer, Infrastructure or Tester engineer.

7.2. Material Resources

There will be used material resources mainly for the development equipment and environment and the running infrastructure or software used.

7.2.1. Hardware Resources

- Development laptop. I will be using a MacBook Pro, 15-inch model from Mid 2015.
- External monitor. To facilitate the development I will use an external monitor of 24” with 1080p resolution. The monitor model is a Samsung S24D330H.
- Laptop support. I will be using a laptop support stand with the dual-screen setup to have better ergonomics as the laptop screen will be aligned with the natural vision.
- Keyboard and mouse. I will be using a generic keyboard and mouse because the laptop will be lifted in the laptop support.

- Standing desk. I will be using an ergonomic table in my development environment. It's ideal for development because it can be changed to a standing position so I can be working both sitting and standing, ideal for ergonomics. I will be using the Ikea Skarsta 160x80cm model.
- Office chair. I will need a comfortable adjustable and ergonomic chair for working. I will be using the Ikea Markus model.
- Servers. I will need servers for deploying the application and its infrastructure.

7.2.2. Software Resources

- IDE. IDE stands for Integrated Development Environment, it consists of an enhanced source code editor, debugger and other build and automation tools. It will be used for writing and maintaining the source code of the application. I'm going to be using WebStorm and Idea IntelliJ IDEA editors from JetBrains company.
- Version Control System. It is where the source code is stored, and it tracks the changes in it. I will be using git VCS hosted on GitHub.
- Development Frameworks. It is a software designed to support and facilitate the development of applications, services or APIs.
- Google Suite. Composed by a diverse set of productivity and collaboration tools. In particular for this project it has been used GMail, Calendar, Meet, Docs and Slides.

8. Budget estimation

Once we have done the time planning and the task definitions, we have enough information to make a budget estimation. Budget estimation is an important tool to analyze the project's economic viability.

The costs will be divided into three categories: direct costs, indirect costs, and unexpected costs.

8.1. Direct costs

In this section, there will be the costs associated and directly derived from all the activities related to the completion of the project.

8.1.1. Human resources

The main roles needed for developing and completing this project are Software Engineer, UI/UX Designer, and Product Manager.

To estimate the human resources cost first it's necessary to estimate how many hours will be needed from each role, and multiply the estimated hours by their respective hourly market rate. Finally, we need to multiply each gross salary by a factor of 1.35 in the concept of the cost it would take a company to maintain this employee, for example, Social Security or training.

		Dedication per role (h)		
Task	**Total Hours**	**Product Manager**	**Software Engineer**	**UI/UX Designer**

Initial Phase	35h	27h	6h	2h
Design Phase	40h	6h	34h	0h
Development Phase	310h	20h	250h	40h
Final Phase	50h	25h	15h	10h

Role	**Hours**	**€/Hour**	**Total**	**Total with company costs (x1.35)**
Product Manager	78h	25€/h	1,950€	2,632.50€
Software Engineer	305h	35€/h	10,675€	14,411.25€
UI/UX Designer	52h	30€/h	1,560€	2,106€
Total	435h		14,185€	19,149.75€

8.1.2. Material resources

Here it includes material resources used for the development of the project. Includes hardware and software used. Basically it consists of the laptop used for development, the IDE and the cloud instances for deploying the project.

Product	**Unit Cost**	**Total Cost**	**Units**	**Lifecycle**	**Amortization**
MacBook PRO 15’, Mid 2015	1,900€	1,900€	1	8 years	1,900€
IntelliJ IDEA Ultimate (IDE)	80€/year	80€	1	1 year	0€
Google Cloud Compute Engine standard instance	22€/month	66€	2 instances for 3 months	Until the project is finished	0€
Total		2046€			1,900€

8.2. Indirect costs

On the other hand, indirect costs are indirect expenses from the activity of completing the project. It consists mainly of the Coworking space, which already includes electricity, internet, and the workplace necessary to develop the project.

Product	**Price**	**Units**	**Estimated price**
Coworking space	250€/month	4 months	1,000€

8.3. Unexpected costs

According to the risks and possible obstacles previously defined in the time planning and the contextualization of the project, we may need more time to deal with unexpected tasks. Mainly that could happen because of some estimation deviation, wrong decisions or a more challenging task than expected.

Taking the 35€/hour rate of a Software Engineer, I will allocate 60 hours as a worst-case scenario, which means 2,100€ for unexpected costs.

8.4. Total costs

Finally, with the previous sections from the budget estimation, we can calculate the total costs. We will add 10% of the costs in the concept of contingency, which is the typical percentage used in technological projects.

	Price
Direct Costs	21,049.75€
Indirect Costs	1,000€
Unexpected Costs	2,100€
Subtotal	24,149.75€
Contingency (10%)	2,414.97€
Total	**26,564.73€**

8.5. Budget management control

To control the budget it will be needed to monitor the time used to develop the project. Therefore, after finishing each task it will be logged the actual time spent in the task so we will be able to calculate the real cost. This way, we can calculate 2 types of deviations: consumption deviation and cost deviation. To get the consumption deviation, we would need to calculate the difference between the estimated hours and actual hours for completing the project multiplied by the estimated cost. For the cost deviation, we need to figure out the difference between the actual and the estimated cost, multiplied by the actual number of hours spent in the project.

*Consumption deviation = (estimated hours - actual hours) * estimated cost*

*Cost deviation = (estimated cost - actual cost) * real hours*

This would allow us to know the situation of the project regarding the cost consumption and its possible deviations, hence we could figure out if the project would be able to handle the deviations using the contingency.

9. Sustainability and social commitment

I have attempted to categorize the project's sustainability by first making sure we understand all the concepts involved in making a sustainable project. As such, this analysis is based on a Survey provided by GEP.

I believe I have always had a firm understanding of the economic impact of my actions, the carbon footprint they incur, along with making sure the project is socially sensitive, but doing the survey helped me fill in the blanks in my knowledge by looking at the information I was missing.

In order to categorize the sustainability of the project, we will be looking at three different categories, the Environmental, Economical and Social aspects of our project.

9.1. Environmental dimension

This project uses very few resources for a programming project, as we only employ one laptop, with the environmental impact that carries. We will be using a laptop that would be employed for tasks regardless, so the most notable impact will be that of the used electricity.

Furthermore, I will also be running the project on Google Cloud's servers which totally run on renewable energy and maintains a commitment to carbon neutrality (net zero carbon footprint). [10] As the architecture was designed such that it's possible to upscale or downscale depending on the resources that are needed, we can minimize the servers and computing power running and we can increase the compute power or number of instances only when needed.

9.2. Economical dimension

We have already gone over the budget of the project extensively in the previous section.

This solution is reasonably efficient for a project of this scope, and the main expense is in the part of the human resources, so our footprint won't be large. The architecture was designed such that it's possible to upscale or downscale depending on the resources needed, hence minimizing underspending and overspending.

As for the project itself, as stated on the introduction of this thesis, businesses incur heavy costs for every hour of downtime or degraded performance of their APIs or web services. Our solution will help businesses grow and save money because of the increased quality of their exposed APIs by having an

9.3. Social dimension

It would be an open-source solution that would benefit the community, will offer other people to learn from the code and will allow the possibility to expand it to their needs. Everyone would be allowed to use or modify the code for free and that will help to improve the overall quality of the different APIs from the Internet.

9.4. Legal dimension

As we wouldn't be storing any personal identifiable information, not even the IP, we wouldn't need to take any special measures about data protection and privacy regulation laws like GDPR. That may change if we serve URLs with sensitive information in it because we do store the requested URLs. That being said, serving sensitive information in the URLs is an awful security practice that must be avoided. [9]

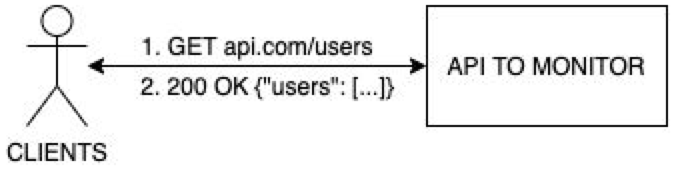
10. Proposed solution

In this section, it will be explained the solution and architecture to solve our objectives and requirements. First it is going to be an overview of the chosen solution and later we're going to see a deeper explanation of it.

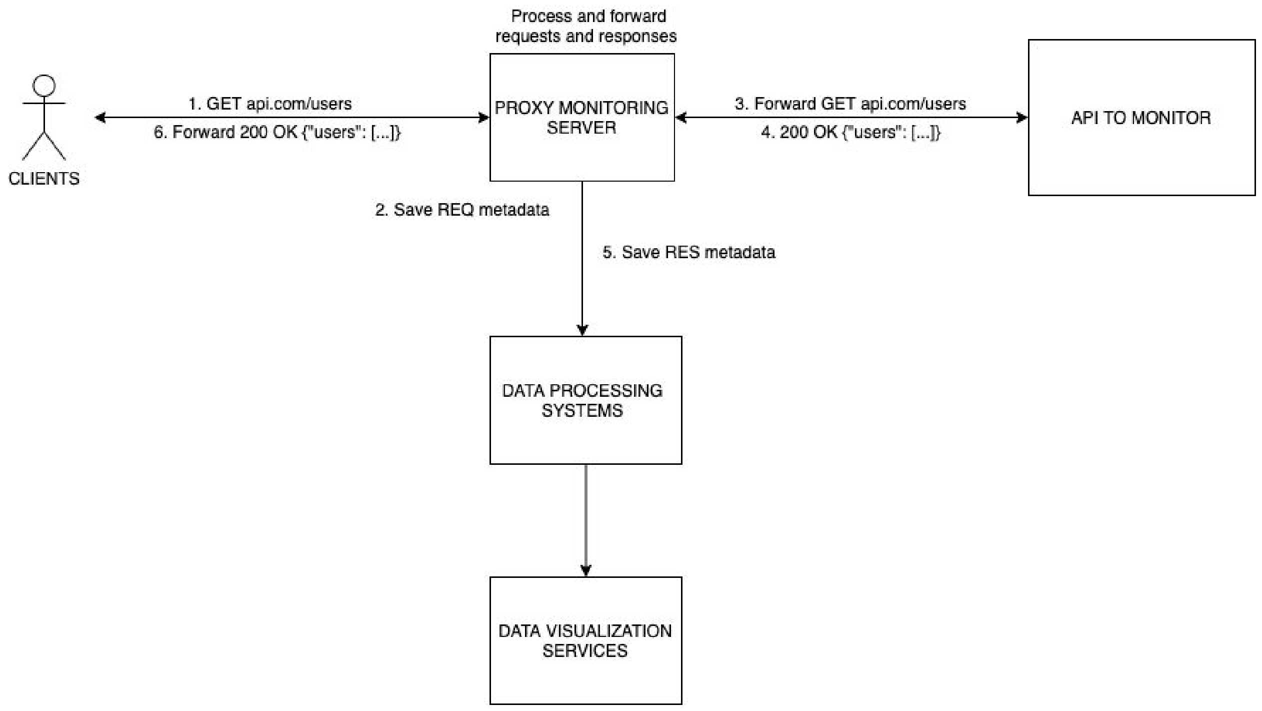
10.1. Architecture overview

After analyzing all the project requirements and its objectives, it's time to analyze and propose a suitable solution.

Normally, the clients would just make a request directly to the API or Web service, as we can see in the following images:



According to the specifications, the solution from the following picture has been designed:


Process and forward requests and responses
CLIENTS
1. GET api.com/users
6. Forward 200 OK {"users": [...]}
PROXY MONITORING SERVER
3. Forward GET api.com/users
4. 200 OK {"users": [...]}
API TO MONITOR
2. Save REQ metadata
5. Save RES metadata
DATA PROCESSING SYSTEMS
DATA VISUALIZATION SERVICES

As we can see in this diagram, the client doesn't interact directly with the API server but it communicates with a Proxy service that monitors the request and response metadata. As an overview:

1. The client starts an HTTP request for the API to monitor but it actually points to the Proxy monitoring server. This change could be made for example via a DNS change or a Load-Balancer on top of the API to monitor such that the Client doesn't need to change anything.

2. The Proxy server intercepts and saves the necessary HTTP request metadata like host, path, method, headers, timestamp, etc.

3. The Proxy server forwards the request to the monitored API while keeping alive the initial request initiated by the Client. Actually the Proxy server starts a new request with the same parameters pointing to the monitored API.
4. The monitored API replies to the HTTP request that the Proxy requested. The Proxy server saves or calculates the necessary HTTP response metadata like status code, headers, time taken to reply, etc.
5. The Proxy server sends all the necessary data to a service that will be consuming, processing and aggregating all the monitored requests.
6. The Proxy sends the response received by the monitored server to the Client.

Then, the data processing systems will process, compute and aggregate the data from the requests published by the proxy. We can later have diverse visualization services connected to the results of these data processing services such that we can easily visualize and take action on the different metrics.

This solution allows us to easily integrate with the monitored server without a need to change its codebase and hence it's language-agnostic.

10.2. Detailed architecture

Now that we've seen the big picture of our solution, let's dig deeper into the components of this solution.

10.2.1. Kafka

Apache Kafka is an open-source distributed stream processing platform developed by LinkedIn and donated to the Apache Software Foundation, written in Scala and Java.

The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds.

Kafka has three main capabilities: publish and subscribe to streams of records (similar to a message queue or to a messaging system), store streams of records in a fault-tolerant durable way and process streams of records as they occur. That makes it perfect for building real-time streaming applications that transform or react to the streams of data. [12]

We’re going to mainly use 3 of the core Kafka APIs in this project:

1. The Producer API allows an application to publish a stream of records to one or more Kafka topics. We will use this API for publishing the new requests that the proxy is receiving.
2. The Consumer API allows an application to subscribe to one or more topics and process the stream of records produced to them. We will be using this API for ingesting records to process or getting the output from the processed records.
3. The Streams API, that allows an application to act as a stream processor, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to new output streams.

We’re going to be using Kafka for receiving all the requests intercepted by the proxy component and we will also use it for processing and aggregating these requests into the metrics we want to track. Kafka would be a central and critical piece of our architecture as all the events are going to be passing through Kafka.

10.2.2. Proxy component

This component is the front-facing component to the user. The final user will directly communicate with this server instead of the monitored server. This component is an HTTP server that proxies all the requests that receive to another server. The host that it's going to be proxied is decided from the `PROXY_MONITORING_TARGET` environment variable. An environment variable is a dynamic-named value that can affect the way running processes will behave on a computer [13].

Besides proxying the HTTP requests, this server creates an event with the following properties: target host (the server being monitored), HTTP method, URL path, query parameters, HTTP status code response, if the request timeout or the server is offline, and the response time (how long it took the target server to respond). This event is published on a Kafka topic.

This component is written and running in Node.js environment mainly because of its simplicity and its asynchronous nature which allows handling thousands of concurrent connections with ease.

10.2.3. Stream processor component

This component will be consuming the requests published by the proxy component and it will calculate and aggregate the different metrics we'll need. This service is written in Java because it's using the Kafka Streams API which is the best way to do stream-processing with Kafka as it already has a lot of useful operations for processing and aggregating data.

Besides doing stateless transformations like mapping or filtering records, Kafka Streams API also allows to easily perform stateful operations like aggregations (computing counts, sums or grouping) or windowing (which lets you group records on time-based windows). As these later operations need state, Kafka Streams allows to store and query data for the required state for processing.

After ingesting the requests published by the proxy service, this component will compute and will publish to the following Kafka topics:

- `REQS_COUNT_PER_MIN_BY_PATH`: first it will group all the requests by the same URL path, host and HTTP method. Then, it will count them aggregated in a 1 minute window, that means that every minute it will publish how many requests per path have been done per route and method. This is going to be used for calculating the throughput handled by the system. With these events we can also understand which are the more popular requests.
- `AVG_RES_TIME_PER_MIN_BY_PATH`: first it will group all the requests by URL path, host and HTTP method. Then, it will compute the average response time for every path. This metric is also going to be aggregated in a 1 minute window. We’re going to use this for visualizing the evolution of the response time in the system. We could also understand which requests are slower or faster.
- `REQ_WITH_ERROR`: in this topic will be published the requests that had an HTTP error status code (4XX or 5XX codes) or that the server didn’t respond to (because it was offline or timeouted). This is a simpler operation as we will only need to filter from the monitored requests by HTTP status code or to the `notAvailable` field. The events from this topic are going to be used for calculating the error rate of any particular path or the one from the global

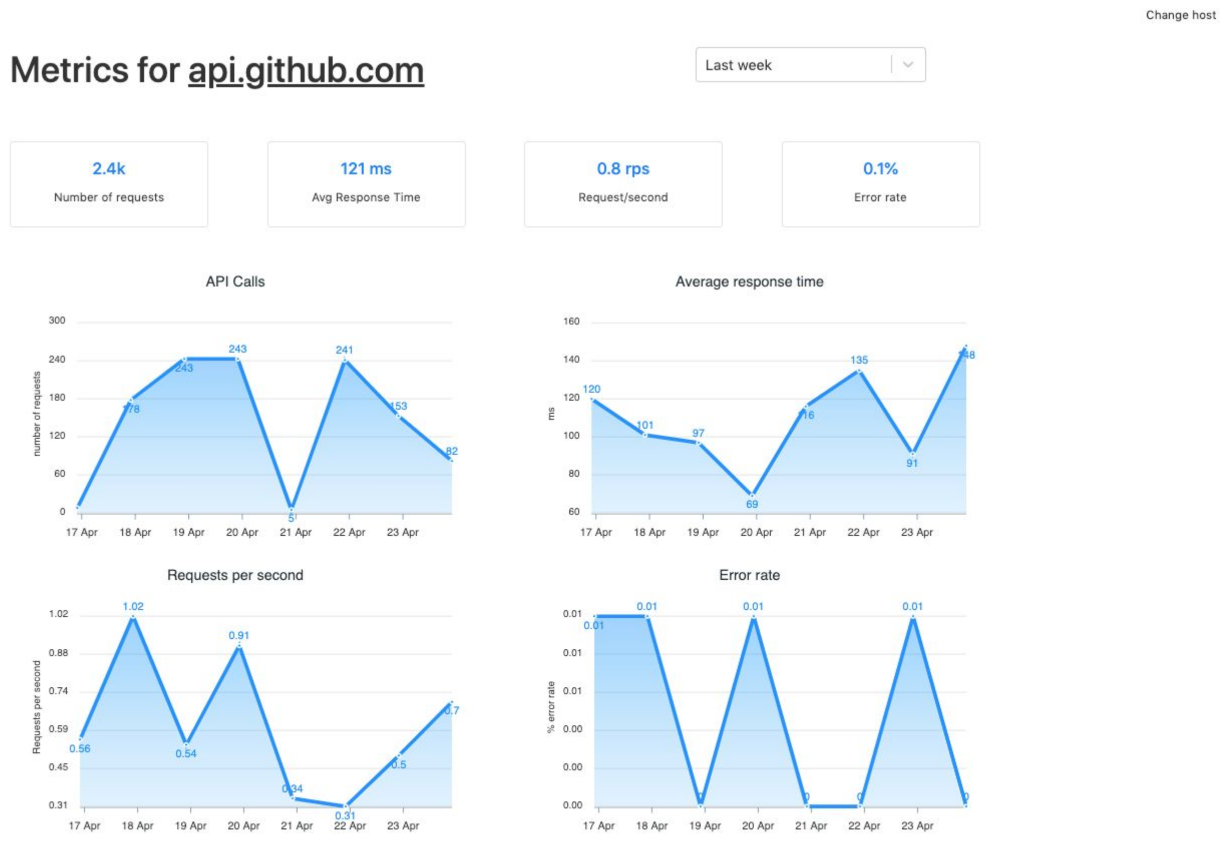
application. It will also be used for displaying which errors happened on the application.

10.2.4. Visualization component

This service is in charge of ingesting and displaying the tracked metrics published in the different Kafka topics. This component is composed by a Node.js application that listens to different Kafka topics and stores the events that it is interested in its own database.

This service also exposes a HTTP REST API for querying the events on a time based manner. We're exposing the average response time on the system and per path, the number of requests in the system and per path, the throughput of the system and per path and the errors on the system and per path including the availability of the system. All of those metrics can be queried by periods of time (last 10 minutes, last day, last week, last month or any custom period you want).

Besides the ingestion of events and the REST API, this service also serves a dashboard to be able to easily visualize all the exposed metrics by the REST API. Next we can see some picture of the main screens of the dashboard.

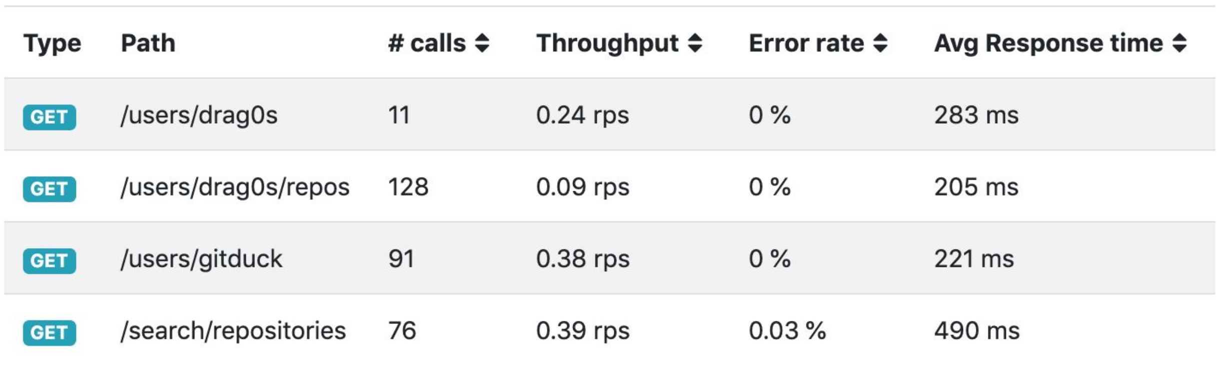


Change host

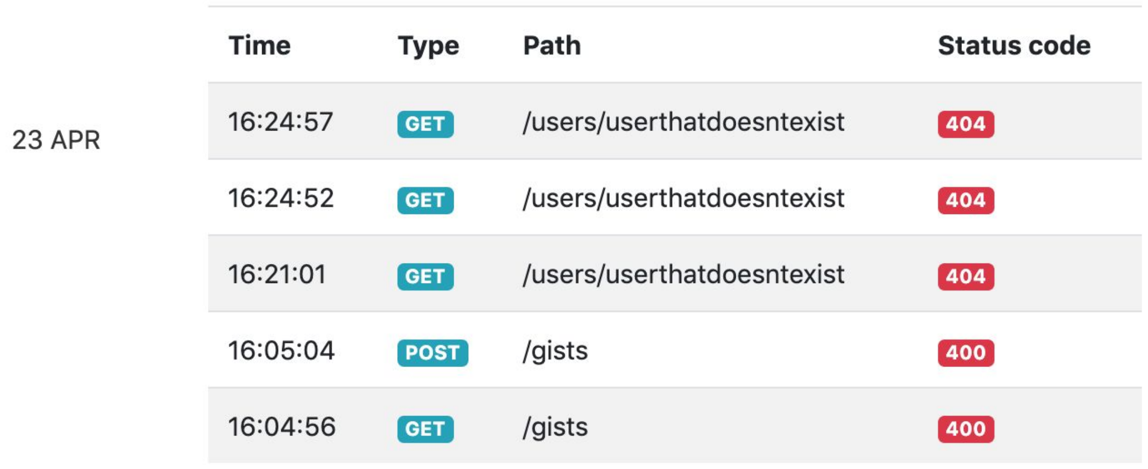
Metrics for api.github.com

Last week

2.4k	121 ms	0.8 rps	0.1%
Number of requests	Avg Response Time	Request/second	Error rate



Type	Path	# calls ⬍	Throughput ⬍	Error rate ⬍	Avg Response time ⬍
GET	/users/drag0s	11	0.24 rps	0 %	283 ms
GET	/users/drag0s/repos	128	0.09 rps	0 %	205 ms
GET	/users/gitduck	91	0.38 rps	0 %	221 ms
GET	/search/repositories	76	0.39 rps	0.03 %	490 ms



	Time	Type	Path	Status code
23 APR	16:24:57	GET	/users/userthatdoesntexist	404
	16:24:52	GET	/users/userthatdoesntexist	404
	16:21:01	GET	/users/userthatdoesntexist	404
	16:05:04	POST	/gists	400
	16:04:56	GET	/gists	400

This service is written in Node.js because of its simplicity and its asynchronous manner.

For the dashboard frontend it uses the React framework with Next.js. The React framework is a declarative component-based JavaScript library for easily building user interfaces. Next.js is a React framework that, besides other things, it allows you to easily render React pages in your Node.js server.

For building the charts, we're going to use an open-source JavaScript library called apexcharts.js, that offers data visualization components for multiple libraries or frameworks, including the React library.

This server is also going to have a WebSocket interface for notifying that new events are published and that the front-end dashboard will use for updating its interface accordingly.

The database that we're going to use for ingesting and storing the Kafka events is going to be TimescaleDB. TimescaleDB is an open-source time-series database powered by

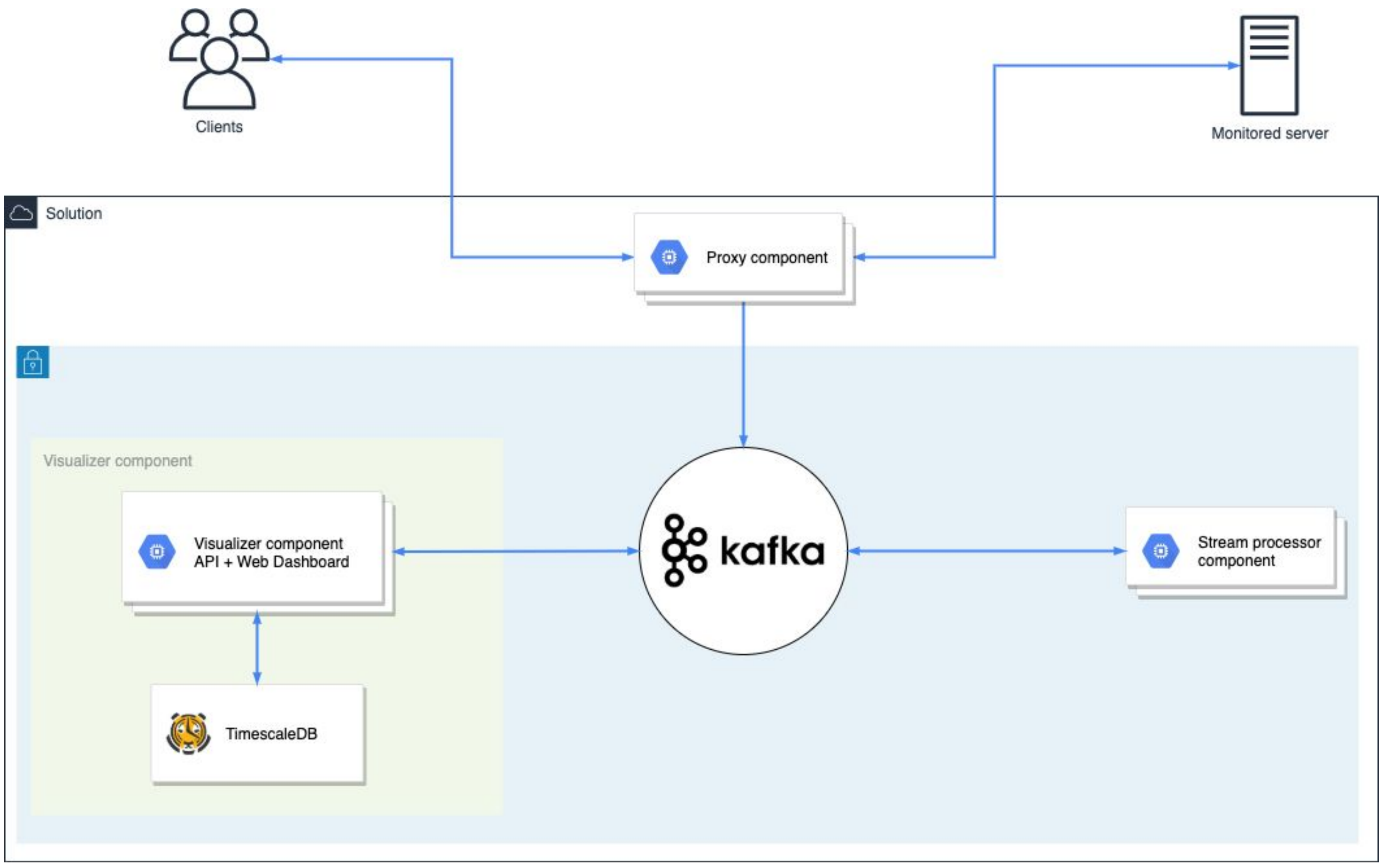
PostgreSQL. It leverages the reliability and maturity of PostgreSQL (it allows you to use all PostgreSQL features) while offering powerful time-oriented features and important optimizations for insertion and querying time. As our data and queries are time-based this is an ideal fit for our needs.

The schema it is in TimescaleDB right now is:

- Table *errors_per_route* with fields id, method, path, query_string, host, timestamp, response_status_code, response_time, is_unavailable.
- Table *req_per_minute_per_route* with fields id, method, path, query_stirng, host, count, timestamp, resolution.
- Table *windows_avg_res_time_per_route* with fields id, method, path, query_string, host, avg_response_time, timestamp, resolution.

10.2.5. Detailed architecture chart

Now that we better defined the architecture and its components in a more detailed and specific way, we can visualize the solution with a more technical and accurate chart.



10.2.6. Kubernetes

As we have very different components and this is a service oriented architecture, I've opted to build and deploy this solution with Kubernetes. Kubernetes is an open-source system for automating deployment, scaling, load-balancing, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery. [14]

Kubernetes makes it very easy to deploy new versions of any service from our application without worrying about downtimes. It also handles the health of our services so if any service crashes Kubernetes will start a new instance of it. If we need more resources or compute power, we can very easily add more replica instances of any service. To more extent, there are Kubernetes packages that roll out a deployment of Kafka or TimescaleDB with a lot of things out of the box like replication, security or backups.

It was also for the development environment a tool called Telepresence. This tool allows you to develop and debug microservices locally while running the rest of the services in the Kubernetes cluster. That way I don’t need to have running the whole infrastructure in my computer but only the parts that I’m working with.

The Kubernetes cluster is going to be deployed in Google Cloud Platform using Google Kubernetes Engine which is a managed version of Kubernetes. Using a managed Kubernetes cluster allows us to quickly spin scalable clusters managed clusters without the worry of fully administering them.

11. Obstacles

After having defined the project with its requirements and its possible risks and obstacles, there are going to be presented the different obstacles faced in this project.

11.1. Time

Time is a risk that was evaluated from the beginning of the project. Some tasks ended up taking more time than expected but the main objectives were successfully achieved. That being said, there was no time to complete the extra objectives we defined like alert notifications and the “Mean Time To Repair” metric.

Apart from that, I had to do an unexpected work-related trip to another country for more than one and a half months which made it impossible to continue working on the project and presenting it on the dates planned initially. That was not that big of a problem because I was able to continue and finish the work in the next thesis announcement in the following quarter.

11.2. Technical challenges

Initially there were also expected risks and obstacles related with technical challenges. In this case, it was true that it was a challenging and demanding project and I worked with new technologies that I never worked before. In particular, Kafka had a big impact here as it was a very different and complex technology. It was challenging to properly set up and make Kafka work with a full stream-processing pipeline. I was trying to make the stream-processing service in Node.js which I realized is much more difficult because of the lack of stream-processing libraries in this language. I ended up writing this service in Java using the Kafka Streams API which was very powerful but at the same time challenging and with an extended documentation.

I also faced that my computer didn’t had the sufficient resources to smoothly run all the services at the same time, I solved this challenge deploying all the services into a development Kubernetes cluster in Google Cloud Platform and using Telepresence tool such that in my development laptop I only need to have running the service I’m writing right now while seamlessly connecting to the other services from the Kubernetes cluster.

11. Conclusion

After explaining the important details about the design and development of this project, in the conclusions we will analyze if we completed the defined objectives or the technical skills defined at the start of the project. Finally, we will analyze the future of the project and reflect on the personal learnings while working on this project. Finally, it is going to be explained the future vision for this project.

11.1. Objectives completion

Regarding the objectives defined In the earlier sections of this thesis, we can say that we achieved them successfully.

The main objective of this project was to build a solution for monitoring external Web APIs in a language-agnostic way and with minimal integration difficulty. After the work of this project we can say that this was successfully achieved.

There were also defined extra objectives to be done in case there was extra time in the project: an alerting system and the “mean time to repair” metric. In this case, there was no extra time so these extra objectives were not achieved. Other things that suffered because of time were automated tests, even though most of the critical functions were tested I would definitely like to have more automated tests in place. That being said, the architecture was designed to be open for extension such that the implementation of these objectives should be relatively straightforward.

11.2. Technical skills

I can say that I achieved the technical skills defined at the start of the project. Given the requirements of this project, I've been able to solve them developing, maintaining and evaluating a complex distributed services architecture. I had evaluated, designed and implemented different database architectures both with relational technologies like TimescaleDB or other approaches like Kafka. I've leveraged the Kafka architecture to create a real-time system. In the development of this project, there were performed different tests, prototypes and configurations for ensuring a good quality following the defined requirements and objectives while mitigating potential risks that could emerge.

11.3. Future work

I really believe the quality, reliability and observability of the APIs is an important problem to solve and that there is still not an ideal solution on the market. That being said I think that this project still needs a lot of work for being a full-fledged production solution that is able to satisfy and serve the market.

Some feature I believe would be extremely valuable for this project:

- Alerting system: be able to configure notification alerts (SMS, email, Slack, etc) when a monitored metric drops a certain threshold or it has an unexpected behaviour.
- More metrics and more advanced visualizations. For example, from which countries does the most of the traffic come from? Ability to correlate anomalies.

- Authentication and authorization. A way to protect your metrics visualizations with an accounts system.
- To be able to spin-up a new monitoring solution for an API automatically without having to set-up any server.

11.4. Personal conclusions

This project was a gratifying experience to work on. Besides being able to achieve the ambitious objectives initially stated, I've been able to work on a system that I really enjoyed and was being able to extend my knowledge in fields that I was really interested in like web APIs, networks, stream-processing, database and cloud computing.

I also loved learning and working with systems like Kafka and being able to learn more about different types of architectures that Kafka enables.

12. Bibliography

[1] GENESIS UPC. Generation and Evolution of Smart APIs. https://genesis.upc.edu/en [Accessed Apr. 1st 2020].

[2] Application programming interface https://en.wikipedia.org/wiki/Application_programming_interface [Accessed Jan. 5th 2020]

[3] Santos, W. (2019). *APIs show Faster Growth Rate in 2019 than Previous Years*. ProgrammableWeb. https://www.programmableweb.com/news/apis-show-faster-growth-rate-2019-previous-years/research/2019/07/17 [Accessed Jan. 5th 2020].

[4] Gartner Blog Network. (2014). The Cost of Downtime. https://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime/ [Accessed Jan. 8th 2020].

[5] API Management, Apigee. https://cloud.google.com/apigee [Accessed Jan. 9th 2020]

[6] Datadog APM. https://www.datadoghq.com/apm/ [Accessed Jan. 9th 2020]

[7] Elastic Uptime. https://www.elastic.co/uptime-monitoring [Accessed Jan. 9th 2020]

[8] Kanban - A brief introduction. https://www.atlassian.com/agile/kanban [Accessed Feb. 1st 2020]

[9] Gantt diagram. Made with https://www.teamgantt.com/

[10] Google. (2019). Google Cloud Sustainability. https://cloud.google.com/sustainability/ [Accessed Jan. 20th 2020]

[11] CWE-598: Use of GET Request Method With Sensitive Query Strings (4.0). https://cwe.mitre.org/data/definitions/598.html [Accessed Apr. 22th 2020]

[12] Apache Kafka introduction https://kafka.apache.org/intro [Accessed Jan. 25th 2020]

[13] Environment variable. https://en.wikipedia.org/wiki/Environment_variable [Accessed Jan. 24th 2020]

[14] Kubernetes. https://kubernetes.io/ [Accessed Apr. 3rd 2020]