Tailored IoT & BigData Sandboxes and Testbeds for Smart, Autonomous and Personalized Services in the European Finance and Insurance Services Ecosystem

∞Infinitech

D5.10 – Data Management Workbench and Open APIs - I

Revision Number	3.0
Task Reference	T5.5
Lead Beneficiary	UBI
Responsible	Konstantinos Perakis
Partners	AGRO AKTIF BOI BOS CP GFT INNOV ISPRINT JSI LXS NBG PRIVE RB UBI WEA
Deliverable Type	Report (R)
Dissemination Level	Public (PU)
Due Date	2020-11-30
Delivered Date	2020-12-28
Internal Reviewers	CCA, RB, GFT
Quality Assurance	CCA
Acceptance	WP Leader Accepted and Coordinator Accepted
EC Project Officer	Pierre-Paul Sondag
Programme	HORIZON 2020 - ICT-11-2018
	This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no 856632

Contributing Partners

Partner Acronym	Role[1]	Author(s)[2]
UBI	Lead Beneficiary	Konstantinos Perakis, Dimitris Miltiadou, Stamatis Pitsios
LXS	Contributor	Alejandro Ramiro, Luis Miguel Garcia
CCA	Internal Reviewer	Paul Lefrere
RB	Internal Reviewer	Alex Kostopoulos, John Kaldis
GFT	Internal Reviewer	Ernesto Troiano
CCA	Quality Assurance	Paul Lefrere

Revision History

Version	Date	Partner(s)	Description
0.1	2020-10-01	UBI	ToC Version
0.2	2020-10-06	UBI	Initial contributions on Section 2 and 3
0.3	2020-10-19	UBI, LXS	Contributions on Section 2
0.35	2020-10-29	UBI	Updated contribution on Section 3
0.40	2020-11-03	UBI, LXS	Updated contributions on Section 2
0.45	2020-11-10	UBI	Updated contribution on Section 3
0.50	2020-11-18	UBI	Updated contribution on Section 3
0.55	2020-11-20	UBI	Contribution on Section 4
0.60	2020-11-24	UBI, LXS	Updated contributions on Section 2
0.65	2020-11-26	UBI	Updated contribution on Section 3
0.70	2020-11-26	UBI	Updated contribution on Section 3
0.80	2020-12-03	UBI	Finalisation of sections 2, 3 and 4
1.0	2020-12-08	UBI	First Version for Internal Review
2.0	2020-12-23	UBI	Version for Quality Assurance
3.0	2020-12-28	UBI	Version for Submission

---

[1] Lead Beneficiary, Contributor, Internal Reviewer, Quality Assurance

[2] Can be left void

Executive Summary

The goal of Task 5.5 "OpenAPI for Analytics and Integrated BigData/AI WorkBench" is to enable access to the added-value analytics functionalities of INFINITECH, that are offered via microservices implementations, and their respective Open APIs in an integrated way through a single entry-point. The document at hand, entitled "Data Management Workbench and Open APIs - I", constitutes the first report of the work performed and the produced outcomes of Task 5.5. The main purpose of this deliverable is to deliver the **detailed design specifications of the INFINITECH Open API Gateway** component which effectively addresses the accessibility and consumption challenges raised from the modular nature of the microservices implementations of added-value offerings in INFINITECH. Hence, the scope of the current report is threefold and includes:

- Performing a detailed and comprehensive **analysis of the challenges imposed by the microservices architecture**, in the consumption of the offered - by the underlying INFINITECH microservices - business functionalities from any client applications or services. In this context, the most commonly applied approaches are presented in detail, documenting their advantages and disadvantages and the reasoning for selecting the API Gateway pattern.
- Presenting the **INFINITECH Open API Gateway** component which constitutes the single entry point for the added-value analytics functionalities and other core offerings of INFINITECH, and a core ingredient of the INFINITECH Reference Architecture. In this context, the set of main functionalities of the component, the core design decisions and the design specifications that were driven by these decisions are presented. Furthermore, the high-level architecture of the component and the two distinct modules that compose this architecture are presented. Finally, the supported use cases from each module and their respective sequence diagrams are documented.
- Documenting the list of well-established **open-source technologies, libraries and frameworks** which will be leveraged during the implementation phase of the INFINITECH Open API Gateway component.

The current deliverable presents the **first version of the design specifications of the INFINITECH Open API Gateway** component. The outcomes of this deliverable will set the basis for the implementation activities performed under the scope of Task 5.5. It should be stressed that the curation of both the definition of the design specifications, as well as the implementation of the presented component is a living process that will last until M30 as per the INFINITECH Description of Action. Hence, the deliverable at hand constitutes a living document that will be continuously updated in order to depict the latest evolution and any introduced updates and refinements. These enhancements and optimisations will be based on the feedback that will be collected by the INFINITECH stakeholders, as well as new requirements that may arise as the project evolves, and will be documented in deliverables D5.11 and D5.12, which will be delivered on M22 and M30 respectively.

Table of Contents

List of Figures

List of Tables

Abbreviations/Acronyms

Abbreviation	Definition
AI	Artificial Intelligence
API	Application Programming Interface
DL	Deep Learning
IP	Internet Protocol
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JWT	JSON Web Token
KPI	Key Performance Indicator
ML	Machine Language
RA	Reference Architecture
REST	Representational State Transfer
URL	Uniform Resource Locator

1 Introduction

D5.10 is released in the scope of WP5 "Data Analytics Enablers for Financial and Insurance Services", which enumerates the associated activities and documents the preliminary outcomes of Task 5.5 "OpenAPI for Analytics and Integrated BigData/AI WorkBench". The deliverable is prepared in accordance with the INFINITECH Description of Action and reports the activities and results of the work performed as the first iteration within the context of Task 5.5. It provides the detailed documentation of the designed solution for the effective and efficient access to the added-value analytics functionalities and other core offerings of INFINITECH through a sophisticated, integrated and single entry-point manner.

The modular nature of the microservices architecture, together with the dynamic nature of the virtualised infrastructures where these microservices are usually deployed, dictates the need for a full solution that will address the problem of how the clients of a microservices-based platform can effectively and efficiently consume their offered business capabilities. In this context, a practical and well-defined pattern is employed to overcome the problem of clients directly accessing the underlying microservices once they have managed to discover their latest network location. As microservices constitute the main ingredient of the INFINITECH platform, and the various analytics and other main functionalities of INFINITECH will be based on microservices, it is imperative that a full solution is designed and implemented, aiming at eliminating the barriers of accessing these offered functionalities from different clients. In this context, the consortium has analysed the key requirements related to accessing microservices, in order to design a solution that will be easily integrated into the INFINITECH platform, and will constitute the single point of entry for these functionalities. The proposed solution will enable the INFINITECH project's pilots, as well as the stakeholders of the financial sector to consume the offered functionalities in a straight-forward and easy manner, and without requiring knowledge of the underlying microservices architecture.

1.1 Objective of the Deliverable

The purpose of this deliverable is to report the outcomes of the work performed within the context of Task 5.5 during the first period of the project till M14. During this first period, the activities mainly focused on the analysis of the key challenges of accessing the microservices implementations of functionalities such as the Machine Learning and Deep Learning algorithms, and the definition of the design specifications of the INFINITECH Open API Gateway component that effectively addresses these challenges.

In this context, the first objective of the deliverable is to present the results of the analysis performed towards addressing the challenges imposed in the microservices architecture for the clients that need to consume the business functionalities offered by the microservices. To this end, the two most common approaches, namely the direct client-to microservices and the API Gateway pattern, are presented in detail by documenting the benefits and drawbacks of each approach.

Following the analysis of the challenges raised by the microservices architecture, the second objective of the deliverable is to introduce the INFINITECH Open API Gateway component. The component is designed based on the API Gateway pattern and eliminates the barriers of accessing the underlying microservices implementations of the INFINITECH platform. First, the design overview of the component is presented, documenting the main functionalities that the component offers. Furthermore, the core design decisions that were taken during the design phase of the component are documented in detail. Following the core design decisions, the design specifications of the component are presented. The modular architecture of the component is described together with the core modules of this architecture, namely the Gateway and the Service Registry. For each module, the functionalities that each module offers, in accordance with the design decisions, are documented. Additionally, for each module the list of supported use cases are documented in detail. For each use case, the respective sequence diagram that depicts the interactions of the involved clients and modules is also documented.

The third objective of the deliverable is to present the list of baseline technologies and tools that will be leveraged in the implementation phase of the INFINITECH Open API Gateway component. The list of

technologies, libraries and frameworks is presented, containing the solutions that are deemed more suitable for the defined design specifications.

It should be noted that according to the INFINITECH Description of Action, Task 5.5 lasts until M30. Hence, two more versions of the deliverable will be released on M22 and M30 with deliverables D5.11 and D5.12. Thus, the upcoming versions of the deliverable will provide the implementation details of the INFINITECH Open API Gateway component, as well as the optimisations and enhancements to the design specifications taking into consideration the evolvement of the project, as well as the feedback that will be collected by the pilots and the stakeholders of INFINITECH.

1.2 Insights from other Tasks and Deliverables

Deliverable D5.10 is released in the scope of WP5 "Data Analytics Enablers for Financial and Insurance Services" and documents the preliminary outcomes of the work performed within the context of Task 5.5 "OpenAPI for Analytics and Integrated BigData/AI WorkBench". The task is directly related to the rest of the Tasks of WP5, enabling access to the added-value analytics functionalities of INFINITECH which are formulated by the library of ML/DL algorithms for Financial and Insurance Services (as produced by T5.4), that incorporates the incremental and parallel data analytics (produced by T5.2) as well as the declarative real-time analytics (produced by T5.3) and leverages the data collection process for the algorithm's training and evaluation (produced by T5.1).

In addition, the task builds on top of the outcomes of WP2 "Vision and Specifications for Autonomous, Intelligent and Personalized Services" in which the overall requirements of the INFINITECH platform are defined. Specifically, Task 5.5 received as input the collected user stories of the pilots of the project and the extracted user requirements, as documented in deliverable D2.1 that reports the work performed in Task 2.1. Furthermore, the inputs of the task included the elicited technical requirements and the fundamental building blocks of the INFINITECH platform and their specifications, as reported in deliverable D2.5. Finally, the task received as input the outcomes of T2.7, in which the INFINITECH Reference Architecture (INFINITECH RA) was formulated during the design specifications definition of the INFINITECH Open API Gateway component that constitutes a part of the INFINITECH platform.

1.3 Structure

This document is structured as follows:

- Section 1 introduces the document, describing the context of the outcomes of the work performed within the task and highlights its relation to the rest of the tasks and deliverables of the project.
- Section 2 documents the motivation and challenges in the access of the deployed microservices of a microservices-based platform by the clients.
- Section 3 presents the design overview and design specifications of the INFINITECH Open API Gateway component, as well as the use cases addressed, along with the corresponding sequence diagrams.
- Section 4 presents the list of baseline technologies and tools that will be utilised in the implementation of the INFINITECH Open API Gateway component.
- Section 5 concludes the document.

2 Motivation and Challenges

In traditional monolithic applications, the application is designed and implemented with the aim of executing a specific, commonly domain-specific, business logic and interacts with the client-side or any third-party client (applications) via an exposed API. In this sense, the clients will usually retrieve the required data or invoke any specific operation by executing a single REST call, that is received and handled by the underlying server-side (backend) application. While the monolithic application approach has a number of benefits, such as simplifying development and deployment, as well as horizontal scalability by multiple running instances behind a load balancer, it also introduces a large number of drawbacks that are very critical and create severe barriers to application execution and evolution. The monolithic architecture has a limitation when it comes to application size and complexity. When the application grows in size and number of functionalities, complexity and sustainability issues arise. The source code becomes difficult to understand and maintain, while also changes in the functionalities offered (and/or the introduction of new functionalities) require great development effort, and add multiple dependencies that should be met, hence the complexity grows. Additionally, the scaling and continuous deployment becomes problematic, since a small update might require a redeployment of all the applications once extensive testing has been successfully performed.

The introduction of the microservices architecture approach solves the drawbacks of the traditional monolithic architecture approach by structuring the underlying application as a collection of microservices that are loosely-coupled, highly maintainable and testable, independently-deployable, organised by business capabilities and can be owned by different development teams [1]. Nevertheless, while the microservices architecture enables the effective and efficient development of large and complex applications in a rapid, frequent and reliable manner, it also introduces a number of challenges that need to be addressed.

In the microservices architecture, the application is partitioned in multiple microservices, where each one of them undertakes a specific business capability of the application and a set of responsibilities. However, this business capabilities decomposition imposes several requirements on the underlying microservices. Each microservice should provide an API that enables the needed intercommunication between the various services. As the context of each microservice is clearly defined, the provided API is usually fine-grained and restricted to the assigned business capability. Hence, the granularity of the APIs of the microservice is usually different than the one required by any client of the application, and there are cases where the client is required to invoke multiple microservices in order to obtain all the required data or execution results. Furthermore, the evolution of the application usually requires the introduction of new microservices or even the refinement of existing ones to support a new business capability. The dynamic nature of the virtualised infrastructures that are hosting the applications, and the microservices that compose these architectures, enables the deployment, upgrade, scaling and restart of each microservice independently of the rest of the microservices of the application. Hence, both the number of microservices, as well as their connection details, such as the hostname and their IP addresses might change and vary dynamically. Thus, the transition from a traditional monolithic application to a microservice application, imposes the problem of how the clients of this application can effectively and efficiently consume the offered business capabilities by either a specific microservice or a combination of microservices.

To this end, the problem can be narrowed down to the client-to-microservices communication. Two approaches can be followed to solve this problem. In the first approach, usually referenced as direct client-to-microservice communication [2], the client is able to directly make requests to the microservices of the application. In this case, each microservice provides an exposed API endpoint and can be reached through a specific URL that maps to a load balancer which in turn distributes the incoming requests to the requested microservice instances (see Figure 1).
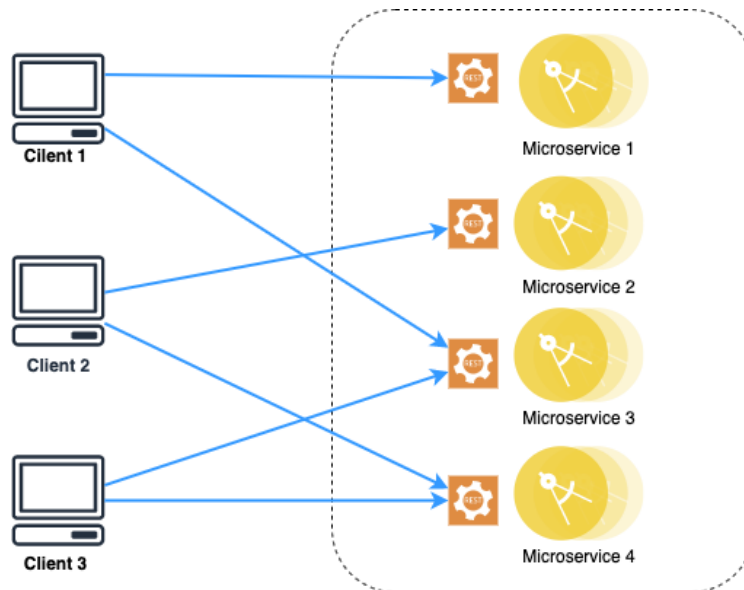
Figure 1: Direct Client-to-Microservices communication

However, this approach has several limitations and challenges that should be taken into consideration. The first challenge arises due to the fact that by design each microservice offers a fine-grained API with a specific context. Thus, there are cases where the client is forced to make multiple requests to different microservices, hence multiple server round trips, to collect the information needed to complete a single operation. This is inefficient and increases the latency in some cases, while it also increases the complexity in the source code of the client. An additional challenge is that microservices might use different and diverse communication protocols (e.g. HTTP, AMQP, Thrift) and as a consequence the client should be capable of interacting with all these diverse communication protocols. Moreover, common functionalities such as authorisation, authentication and logging, need to be taken care of at the microservices level, rather than at a common cross-cutting level on top of the microservices. Finally, as the implementation of the client is directly connected with the underlying microservices, any changes or updates in the existing microservices, such as the merging or splitting of an existing microservice, or the introduction of new ones, usually propagates changes on the clients that need to adapt to the evolution of the microservices.

The second approach that addresses all the above-mentioned challenges is the API Gateway pattern [3]. The API Gateway is introduced as an intermediate layer between the clients and the underlying microservices, and acts as a single entry-point for all clients that consume these microservices. All incoming requests are generated towards the API Gateway which serves them in two core ways. The API Gateway usually routes the request to the appropriate microservice acting as a reverse proxy, however in some cases it handles the request by invoking multiple microservices and aggregating the results, which are provided back to the requestor (see Figure 2). In addition to this, the API Gateway undertakes several cross-cutting functionalities such as the authorisation, authentication, monitoring and load balancing.

The API Gateway encapsulates the underlying system architecture, hiding the details of how the application is partitioned into microservices, providing different APIs for diverse clients if needed. It eliminates the need for clients to discover or keep track of the network locations of the microservices and their exposed APIs that can change dynamically. Furthermore, it reduces the problem of multiple server round trips, as it handles the invocation of multiple microservices and aggregation of the results. The API Gateway is able to handle the diverse communication protocols problem, since it exposes a well-defined and web-friendly API, undertaking the protocol conversion internally. Thus, the API Gateway is able to simplify both the communication with the clients, as well as the complexity of the client's source code. The client is only interacting with the API Gateway, hence the update or evolution of the application's microservice is hidden from the client. Furthermore, it handles cross-cutting functionalities, while the development of the underlying microservices is also simplified.
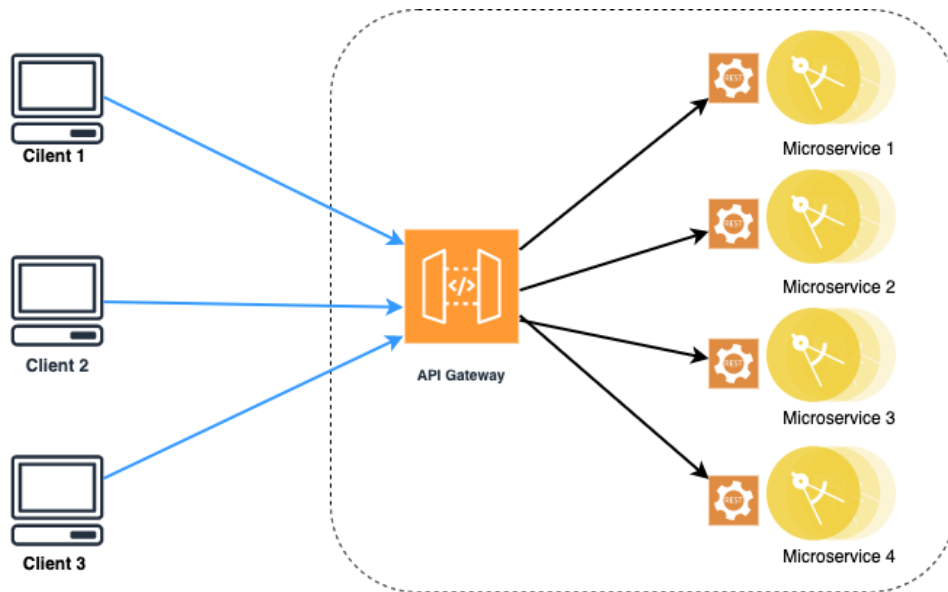
Figure 2: API Gateway pattern

However, the API Gateway pattern also has some drawbacks. As it constitutes a component with high availability, it should be designed, developed, deployed and maintained in a proper way in order to avoid becoming a bottleneck for the application. Hence, it is crucial that several aspects such as performance, adaptability and fault tolerance are taken into consideration to benefit from the API Gateway pattern solution. Nevertheless, despite these drawbacks, the API Gateway pattern is considered to be the proper approach in the case of INFINITECH, as it effectively solves all the fundamental issues and challenges described in the direct client-to-microservice communication.

3 The INFINITECH Open API Gateway

3.1 Design overview

The **INFINITECH Open API Gateway** is a sophisticated API Gateway that encompasses the Open API specification in order to provide a single point of entry for the added-value functionalities of INFINITECH which are based on microservices. The work that was performed during this phase was mainly focused on the Machine Learning (ML) / Deep Learning (DL) analytics functionalities of INFINITECH, which are implemented as microservices. Following the API Gateway pattern, as presented in Section 2, it is capable of effectively handling and determining the network location of dynamically-deployed microservice instances, while at the same time hiding the internal application's architecture from the clients of the application.

Hence, the main functionalities of the INFINITECH Open API Gateway are as follows:

- To act as a single point of entry for the ML/DL analytics functionalities of INFINITECH, handling the incoming requests towards the dynamically-deployed microservice instances;
- To facilitate the communication between the underlying microservice instances;
- To enable the self-registration of the microservice instances;
- To facilitate the discovery of the microservice instances;
- To enable the discovery of the microservice instance Open APIs;
- To effectively handle the unavailability or unreachability of the underlying microservice instances;
- To provide authentication, monitoring and logging functionalities to the microservice instances.

During the design phase of the INFINITECH Open API Gateway, several design decisions were taken based on the specifications set by the INFINITECH Reference Architecture, as documented in deliverable D2.13, the elicited technical requirements as reported in deliverable D2.5, as well as the requirements of the pilots and stakeholders of INFINITECH, as documented in deliverable D2.1. Within the INFINITECH Reference Architecture, the INFINITECH Open API Gateway is positioned in the Interface layer of the architecture undertaking the task of providing the required information from the Analytics layer to the Presentation layer where the clients reside.

With regards to the *Request Handling* processes of the INFINITECH Open API Gateway, it was decided that it should support both core ways of the API Gateway pattern. Hence, the INFINITECH Open API Gateway should be able to handle one to one (1-to-1) microservices invocation, meaning it can handle requests that are simply translated by routing the incoming request to the appropriate microservice. On the other hand, the INFINITECH Open API Gateway should handle one to many (1-to-many) microservices invocation, thus a request could be translated into the invocation of multiple microservices, whose results are aggregated and returned to the requestor. In the latter case, a sophisticated logic will be employed to effectively handle the multiple invocations, dependencies between the invocations, and failure handling through modern approaches, such as reactive programming with the promises pattern instead of the traditional asynchronous call-back pattern.

As the underlying microservices are operating in a distributed manner, it is required to establish a set of *Communication Mechanisms* across the different processes that enable the invocation of the microservices in both an asynchronous and synchronous manner. To this end, both approaches will be supported with the proper mechanisms for asynchronous message-based communication and synchronous communication.

One the core aspects of the INFINITECH Open API Gateway is the effective and dynamic *Service Registry* and *Service Discovery* processes. The INFINITECH Open API Gateway is designed to be able to find and keep track of the network location of each microservice that it handles the requests for. As explained in Section 2, microservices are usually operating in a cloud infrastructure, within virtual machines or containers, that change dynamically in terms of IP addresses and ports. Hence, it is imperative that the INFINITECH Open API Gateway incorporates the appropriate Service Registry and Service Discovery processes that effectively handle the changes of the microservice instances in a dynamic manner.

Service Registry is a database that maintains the updated network locations of the microservices. Hence, the microservice instances are registered with the Service Registry on start-up and deregistered on shutdown. In the meanwhile, the Service Registry performs periodic health checks on the registered microservices to ensure their availability. For the service registration, there are two approaches, the self-registration pattern and the third-party registration pattern. In the Self-Registration pattern [4], the microservice instance is responsible for the registration and deregistration of itself with the service registry, while in the third-party registration pattern [5] the registration and deregistration is handled by a third party application called registrar, by performing polling operations or event subscriptions. In the INFINITECH Open API Gateway, the self-registration pattern is followed as it keeps the complexity of the component at the lower levels, while the coupling of the registration process with the microservices requires less effort, as there is a large number of available libraries performing these operations with out-of-the-box integration.

With regards to Service Discovery there are two approaches, namely the client-side service discovery and the server-side discovery. In the client-side service discovery [6], the client retrieves the latest network location of a microservice by querying the Service Registry. While this approach is considered more efficient in terms of network requests, it bounds the client implementation with the Service Registry. On the other hand, in the server-side discovery [6] the client makes requests to the API Gateway and the API Gateway queries the Service Registry before it finally routes the request to the latest network location of the requested microservice. In the INFINITECH Open API Gateway, the server-side discovery is followed as it eliminates the need of coupling the client's implementation with the Service Registry, and additionally it hides the implementation details of API Gateway and the registered microservices from the client.

A crucial aspect of the INFINITECH Open API Gateway is the embracement of **Open APIs** to enhance the accessibility of the underlying ML / DL microservices. The Open API specification is proposed by the Open API initiative, which is an open-source collaboration project of the Linux foundation, and defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service, without access to the source code, documentation, or through network traffic inspection [7]. The INFINITECH Open API Gateway will embrace the Open API specification which all underlying ML / DL microservices are required to follow in order to help the clients understand and consume their exposed services without the need for knowledge of the implementation details or access to the source code of the microservices. Hence, the INFINITECH Open API Gateway will facilitate publishing the offered Open APIs of the microservices in order to enable their easy and effortless discovery by the clients of the INFINITECH Open API Gateway.

Another aspect of the INFINITECH Open API Gateway is *Fault Tolerance*. For either 1-to-1 or 1-to-many microservices invocation, a mechanism performing efficient failure handling should be employed. Towards this end, the Circuit Breaker pattern [8] will be followed in the INFINITECH Open API Gateway. The specific pattern is utilised to handle the case where one of the invoked microservices is unavailable or unreachable and the whole process is not stalled, while the allocated resources are properly managed. In this pattern, if a microservice reaches a threshold of consecutive failures, then the circuit breaker that acts as a proxy will immediately reject all upcoming requests to this microservice for a predefined testing period. When this testing period ends, a limited number of test requests are made; if they succeed, the circuit breaker allows the invocation of the microservice again, else a new testing period is started again.

One final aspect of the INFINITECH Open API Gateway is the offering of cross-cutting functionalities such as authentication, monitoring and logging. With regards to the authentication, the INFINITECH Open API Gateway will embrace the Access Token pattern and specifically the JSON Web Token (JWT). The Open API Gateway that acts as the single point of entry for the ML/DL functionalities of INFINITECH will authenticate the incoming requests from the clients and will provide a JWT that securely authenticates the requestor back to the microservices. Furthermore, the INFINITECH Open API Gateway performs continuous monitoring and detailed logging of all the received requests and executed operations for security, malicious activity and performance analysis purposes.

The following table summarizes the main design decisions that were taken during the design phase of the INFINITECH Open API Gateway.

Table 1: INFINITECH Open API Gateway design aspects

Design Aspect	INFINITECH Open API Gateway
Request Handling	Both 1-to-1 and 1-to-many microservices invocation
Communication Mechanisms	Both asynchronous message-based and synchronous communication
Service Registry	Self-registration pattern
Service Discovery	Server-side discovery pattern
API Specifications	Open API specification
Fault Tolerance	Circuit Breaker pattern
Cross-cutting functionalities	Authentication, Monitoring, Logging

It should be noted that the presented design aspects of the INFINITECH Open API Gateway do not limit the support for additional added-value functionalities which are based on microservices. On the contrary, the described approach facilitates the expansion of the list of supported microservices implementations in an effortless and effective manner upon the project's needs

3.2 Design Specifications

During the analysis of the client-to-microservices communication problem, it was clear that the API Gateway pattern is considered as the most dominant solution for this problem as it effectively and efficiently resolves the problem in a solid and robust manner. Hence, the consortium decided to design and implement the INFINITECH Open API Gateway that is based on the API Gateway pattern, aiming at providing the required access to the underlying ML/DL microservices. However, as presented in the overview of the INFINITECH Open API Gateway in the previous section, there are several aspects that were carefully addressed in the design phase towards the implementation of the optimal (most effective and efficient) solution that will address the requirements of INFINITECH stakeholders.

The INFINITECH Open API Gateway has a modular architecture composed of two core modules, namely the *Gateway* and the *Service Registry*. Each module has a clear scope and a distinct context undertaking a specific set of responsibilities. Additionally, the modules are interacting through well-defined REST APIs in order to perform the main operations of the INFINITECH Open API Gateway. Their role in the INFINITECH Open API Gateway is as follows:

- The Gateway provides the single entry point for all incoming requests from the pilots and undertakes the responsibility of invoking the appropriate microservices while also ensuring the required inter-communication between the microservices where needed.
- The Service Registry provides the database that maintains the updated network locations of the microservices. Furthermore, it provides the mechanism for self-registration and deregistration, while also performing the health check operations.

Figure 3 depicts the high-level architecture of the INFINITECH Open API Gateway. As illustrated, the Gateway is capable of receiving new requests from the clients for the invocation of the underlying ML/DL microservices. Upon receiving a new request, the Gateway consults the Service Registry in order to discover the updated network location of the requested microservice. Once the information is retrieved by the Service Registry, the Gateway, acting as a reverse proxy, routes the request to the appropriate microservice by invoking the respective endpoint of the microservice. The selected microservice handles the request and provides the results back to the Gateway. At this final step, the Gateway replies to the client with the results as provided by the respective ML/DL microservice.
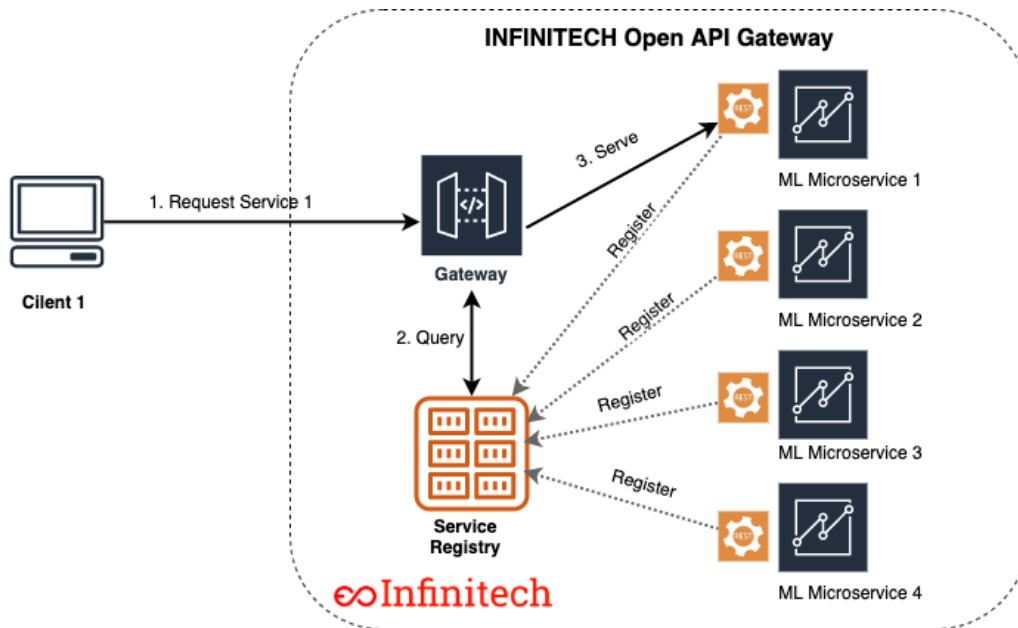
Figure 3: INFINITECH Open API Gateway high-level architecture

The execution is slightly differentiated in the case where a request requires the invocation of multiple microservices. In this case, the Gateway upon receiving the request, consults the service registry and retrieves the list of microservices. Then, multiple requests are initiated to the respective microservices. Each microservice provides the results back to the Gateway and when all results are received the Gateway aggregates the results and sends them back to the client.

The scope of the Gateway is to facilitate the invocation of ML/DL microservices, hiding the details of the deployed microservices from the client. It undertakes all the core functionalities of the INFINITECH Open API Gateway utilising the services of the Service Registry to fulfil its purpose. Furthermore, the Gateway provides cross-cutting operations to the respective ML/DL microservices decoupling their implementation from these operations.

In accordance with the INFINITECH Open API Gateway overview presented in section 3.1, the main functionalities of the Gateway module are as follows:

- It performs the handling of incoming requests, acting as a reverse proxy and routing the incoming requests to the respective ML/DL microservices either for 1-to-1 microservices invocation or 1-to-many microservices invocation.
- It provides the communication mechanism for the microservices invocation in both asynchronous and synchronous ways.
- It performs the service discovery operations by interacting with the Service Registry during the request handling process.
- It enables the publishing of the Open APIs of the ML/DL microservices.
- It implements the fault tolerance mechanism for the effective handling of failures from the microservices side implementing the Circuit Breaker pattern.
- It undertakes the authentication of the requestor operations utilising JWT which are passed to the respective microservice.
- It performs the continuous monitoring and logging of all operations performed in the INFINITECH Open API Gateway

The scope of the Service Registry is to support the operations performed by the Gateway for the service discovery. In particular, the Service Registry is the complementary module that is capable of maintaining the updated network location of each microservice, and of providing the appropriate information to the Gateway. Additionally, it decouples the service registration and deregistration from the INFINITECH Open API Gateway.

In accordance with the INFINITECH Open API Gateway overview presented in section 3.1, the main functionalities of the Service Registry module are as follows:

- It provides the mechanism for the self-registration and self-deregistration of the ML/DL microservices by employing the mechanism that handles these requests, and the registration client that should be integrated in the ML/DL microservices.
- It supports the service discovery operations by providing the required information to the Gateway.
- It performs periodic health checks upon the registered ML/DL microservices to ensure the availability of the registered microservices.

3.3 Use Cases and Sequence Diagrams

As explained in Section 3.2, the INFINITECH Open API Gateway is composed of two core modules, namely the Gateway and the Service Registry. In this section, the detailed use cases that each specific module addresses are documented, presenting in detail the relevant information. Additionally, for each use case, the corresponding sequence diagram that depicts the interactions of the modules and the involved clients is presented.

3.3.1 Gateway

3.3.1.1 Open APIs discovery

The specific functionality enables the discovery of the Open APIs of the registered microservices by the client. To achieve this, the Gateway maintains and displays a dynamic list where all registered microservices are listed. For each microservice, the list of Open APIs can be retrieved by the user. The list of Open APIs is made available to the Gateway during the self-registration process of each microservice.

Table 2: Gateway – Open APIs discovery

| Stakeholders involved:	Client's User
Pre-conditions:	1. The existence of at least one registered microservice with Open APIs
Post-conditions:	1. The client's user is able to retrieve the list of Open APIs of a registered microservice
Data Attributes	None
Normal Flow	1. The client's user navigates to the page where the dynamic list of registered microservices is displayed by the Gateway 2. The client's user selects one of the microservices from the list 3. The list of Open APIs is displayed to the user following the Open API specification
Pass Metrics	1. The client's user is able to retrieve the list of Open APIs for the selected microservice
Fail Metrics	1. The client's user cannot retrieve the list of Open APIs for the selected microservice

Open APIs discovery
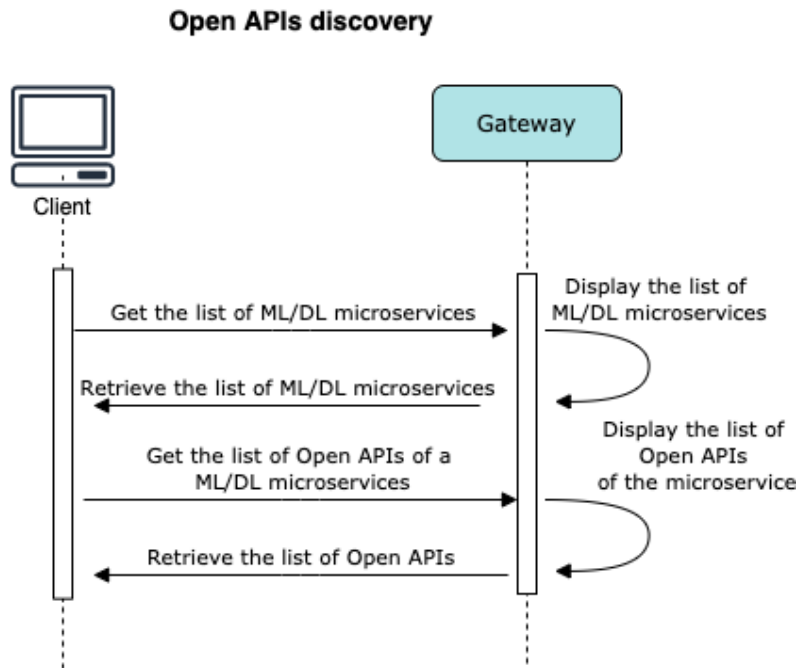
Figure 4: Open APIs discovery sequence diagram

3.3.1.2 Request Handling (1-to-1)

The specific functionality handles the incoming request for the invocation of the ML/DL microservice. The Gateway receives the request from the client and consults the Service Registry to retrieve the updated network location of the requested microservice. Upon retrieving the network location, the Gateway routes the request to the requested ML/DL microservice. The microservice receives and handles the request. The results are provided back to the Gateway which in turn returns them to the client. The specific use case also handles the case where the requested ML/DL microservice is unreachable or irresponsive.

Table 3: Gateway – Request Handling (1-to-1)

Stakeholders involved:	Client, Gateway
Pre-conditions:	1. The existence of at least one registered microservice with Open APIs
Post-conditions:	1. The client retrieves the results of the executed ML/DL microservice
Data Attributes	1. The endpoint of the desired microservice, prefixed with the name that the corresponding microservice used to register with the registry. 2. The data that needs to be forwarded to the desired microservice 3. The JWT that will be used to authenticate the client
Normal Flow	1. The client initiates a request for a specific ML/DL microservice to the Gateway 2. The Gateway consults the Service Registry to retrieve the updated network location of the ML/DL microservice 3. Upon the retrieval of the updated network location, the Gateway routes the request to the specific ML/DL microservice

|	
	In the case where the ML/DL microservice is available and operational:
	4. The ML/DL microservice receives and handles the request. The results are provided to the Gateway
	5. The Gateway propagates the results to the client
	In the case where the ML/DL microservice is unavailable and irresponsive:
	4. The request is not properly received by the requested ML/DL microservice
	5. The Gateway is informed and reports the failure to the requestor
Pass Metrics	1. The requested ML/DL microservice is invoked successfully and the results of the execution are returned back to the client
	2. In the case where the requested ML/DL microservice is unavailable and irresponsive, the client is informed of the failure of the request
Fail Metrics	1. The requested ML/DL microservice cannot be invoked and the request fails
	2. The client is not informed when a failure occurs
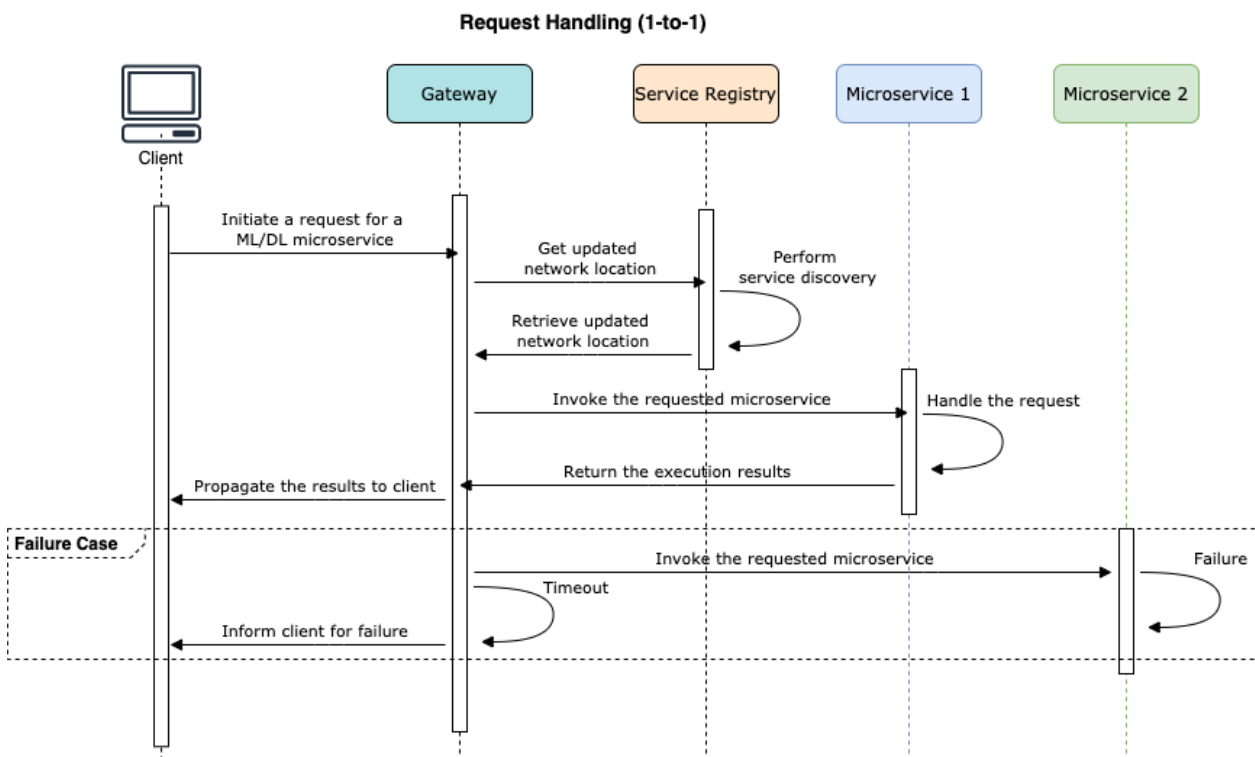
Figure 5: Request Handling (1-to-1)

3.3.1.3 Request Handling (1-to-many)

The specific functionality handles the incoming request for the invocation of a set of ML/DL microservices. The Gateway receives the request from the client and consults the Service Registry to retrieve the updated network location of the requested microservices. Upon retrieving the network location of the microservices, the Gateway initiates one request to each of the ML/DL microservices involved in the request. The respective microservices receive and handle the requests. The results from each microservice execution are provided back to the Gateway. Upon receiving all the results, the Gateway aggregates the results and returns them to the client. The specific use case also handles the case where one of the requested ML/DL microservices is unreachable or irresponsive.

Table 4: Gateway – Request Handling (1-to-many)

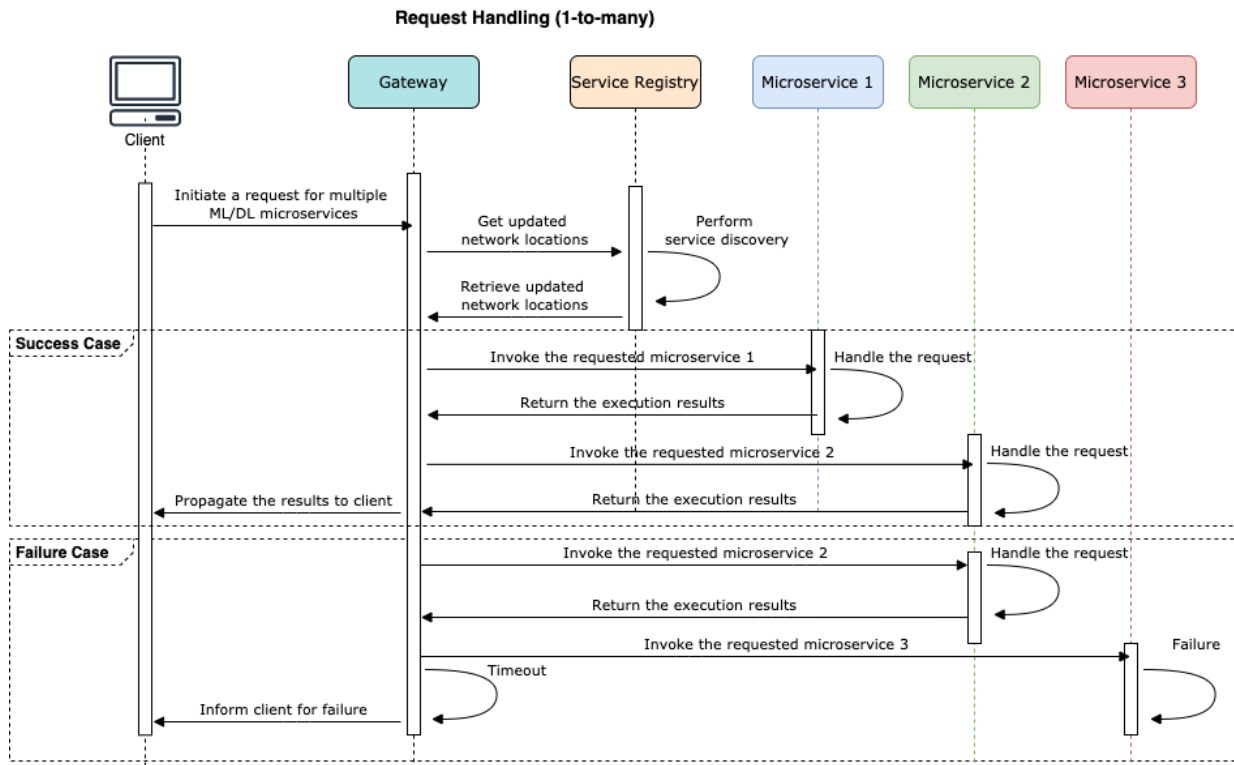| Stakeholders involved:	Client, Gateway
Pre-conditions:	1. The existence of at least two registered microservices with Open APIs
Post-conditions:	1. The client retrieves the results of the executed ML/DL microservices
Data Attributes	1. A list of the endpoints of the desired microservices, prefixed with the names that the corresponding microservices used to register with the registry. 2. The data that needs to be forwarded to each of the desired microservices 3. The JWT that will be used to authenticate the client
Normal Flow	1. The client initiates a request to the Gateway. 2. The Gateway retrieves the request which involves the execution of the multiple microservices. It consults the Service Registry to retrieve the updated network location of the ML/DL microservices 3. Upon the retrieval of the updated network locations of the microservices, the Gateway initiates a request to each specific ML/DL microservice *In the case where all ML/DL microservices are available and operational:* 4. Each ML/DL microservice receives and handles the request. The results are provided to the Gateway 5. Upon receiving all the results from all invoked microservices, the Gateway aggregates the results 6. The Gateway propagates the aggregated results to the client *In the case where one of the ML/DL microservices is unavailable and irresponsive:* 4. One of the requests is not properly received by the requested ML/DL microservice 5. The Gateway is informed, stops the processing and reports the failure to the requestor
Pass Metrics	1. The requested ML/DL microservices are invoked successfully and the aggregate results are returned back to the client 2. In the case where one of the requested ML/DL microservice is unavailable and irresponsive, the client is informed of the failure of the request
Fail Metrics	1. The requested ML/DL microservices cannot be invoked and the request fails 2. The client is not informed when a failure occurs

Figure 6: Request Handling (1-to-many)

3.3.2 Service Registry

3.3.2.1 Self-registration

The specific functionality handles the self-registration of the ML/DL microservice in the Service Registry of the INFINITECH Open API Gateway. The prerequisite for the registration is the integration of the registry client in the microservice implementation. During the microservice start-up, the microservice is self-registered to the Service Registry.

Table 5: Service Registry – Self-registration

Stakeholders involved:	ML/DL microservice, Service Registry
Pre-conditions:	1. The existence of a ML/DL microservice that has integrated the registry client and its implementation
Post-conditions:	1. The ML/DL microservice is registered in the Service Registry
Data Attributes	Upon self-registration, the microservice needs to send the following information to the service registry: 1. The name that the microservice will use as a unique identifier to register to registry 2. The host of the microservice 3. The port of the microservice 4. The health-check endpoint

|	
	5. The health-check interval
Normal Flow	1. The ML/DL microservice is started. The integrated registry client is invoked. 2. The registry client communicates with the Service Registry and self-registers the microservice instance.
Pass Metrics	1. The ML/DL microservice is successfully registered in the Service Registry and it is ready to receive new requests
Fail Metrics	1. The ML/DL microservice failed to register in the Service Registry
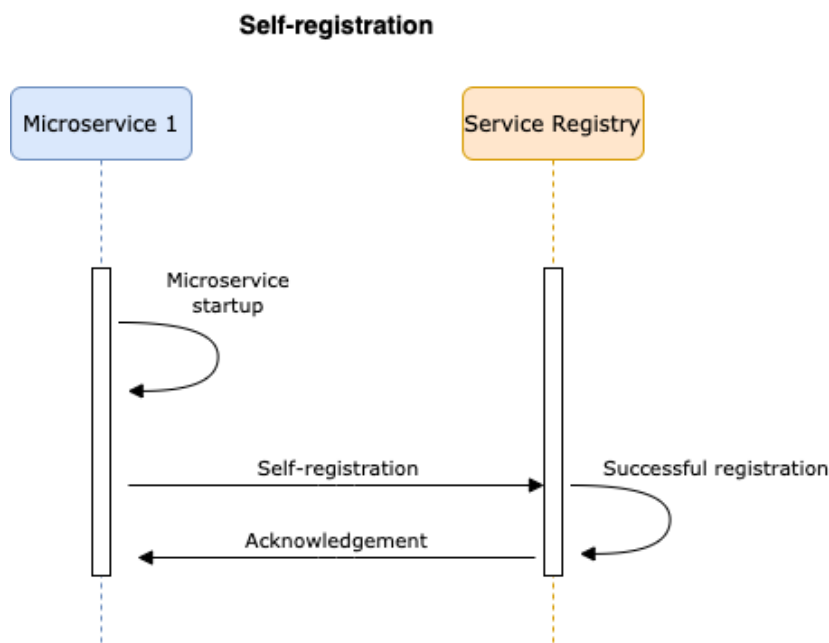
Figure 7: Self-registration sequence diagram

3.3.2.2 Self-deregistration

The specific functionality handles the self-deregistration of the ML/DL microservice in the Service Registry of the INFINITECH Open API Gateway. As a prerequisite, the registry client should be integrated with the microservice implementation. During the microservice shutdown, the microservice is self-deregistered from the Service Registry.

Table 6: Service Registry – Self-deregistration

Stakeholders involved:	ML/DL microservice, Service Registry
Pre-conditions:	1. The existence of a ML/DL microservice that has integrated the registry client on its implementation and has been successfully registered in the Service Registry.
Post-conditions:	1. The ML/DL microservice is successfully deregistered from the Service Registry
Data Attributes	1. The identifier of the microservice that needs to be deregistered

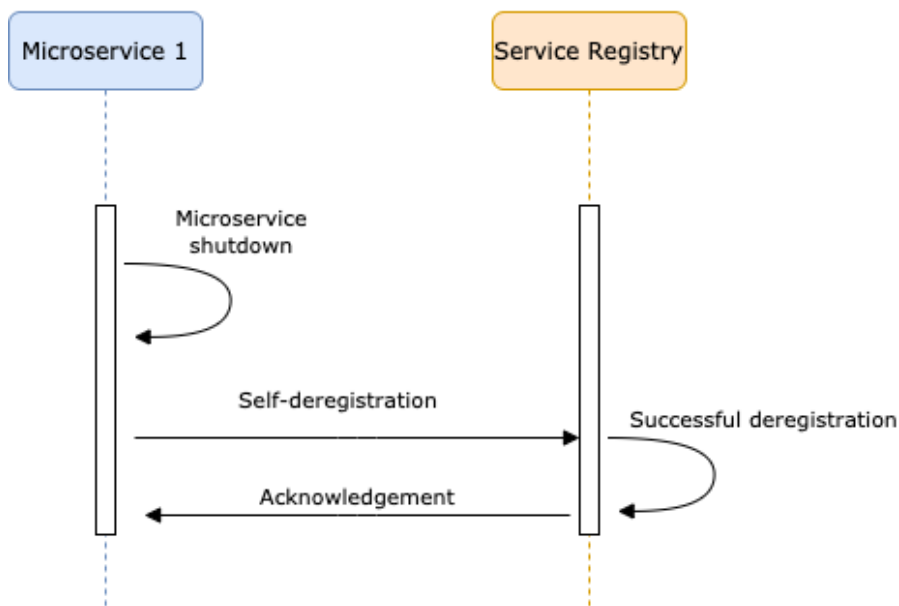|	
Normal Flow	1. The ML/DL microservice initiates shutdown. The integrated registry client is invoked. 2. The registry client communicates with the Service Registry and self-deregisters.
Pass Metrics	1. The ML/DL microservice is successfully deregistered from the Service Registry
Fail Metrics	1. The ML/DL microservice failed to deregister from the Service Registry

Figure 8: Self-deregistration sequence diagram

3.3.2.3 Periodic Health Check

The specific functionality handles the need for periodic health check execution on the registered microservices to ensure their availability. In this context, the Service Registry in every period checks if each registered service is reachable and operating as expected. In the case where a microservice is not responding, then the microservice is marked as in an "Unhealthy" state. At this point, the microservice can properly self-deregister when it comes back into service or it can be removed by the administrator of the INFINITECH Open API Gateway at any time.

Table 7: Service Registry – Periodic Health Check

Stakeholders involved:	ML/DL microservice, Service Registry
Pre-conditions:	1. The existence of a ML/DL microservice that has been successfully registered in the Service Registry and is operating as expected. 2. The existence of a ML/DL microservice that has been successfully registered in the Service Registry and is not responding or is not reachable any more.

| Post-conditions:	1. The operational ML/DL microservice remains in the Service Registry in a "Healthy" state and the irresponsive ML/DL microservice is flagged as "Unhealthy"
Data Attributes	N/A
Normal Flow	1. The Service Registry initiates the health check operation for a specific microservice that is not reachable or irresponsive. 2. The specific microservice is flagged as "Unhealthy" 3. The Service Registry initiates the health check operation for a specific microservice that is operational. 4. The specific microservice remains in the Service Registry in a "Healthy" state
Pass Metrics	1. The operational microservice remains in the Service Registry in a "Healthy" state, while if irresponsive is in an "Unhealthy" state.
Fail Metrics	1. The operational microservice is flagged as "Unhealthy" and / or the irresponsive is flagged as "Healthy"
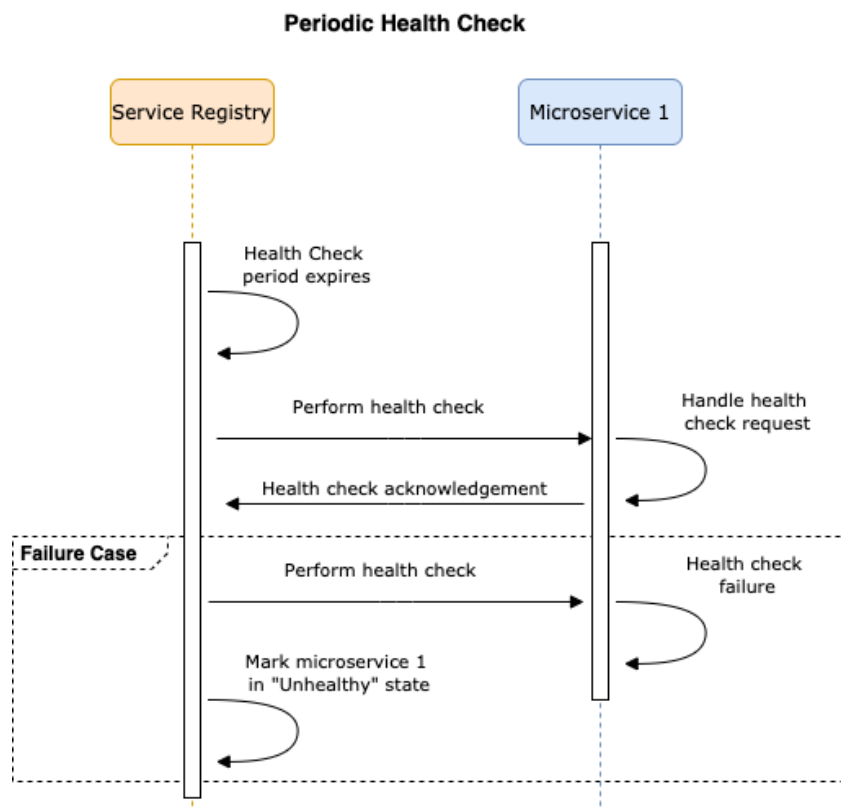
Figure 9: Periodic Health Check sequence diagram

4 Baseline technologies and tools

During the design phase of the INFINITECH Open API Gateway, several high-potential technologies and tools were investigated and an evaluation of multiple aspects of each candidate technology and tool was conducted. The basic criteria during the selection process were as follows:

- The offered functionalities of each technology and tool, as well as their relevance to the aspired functionalities of the INFINITECH Open API Gateway.
- The applicability of the offered functionalities to the designed use cases for the INFINITECH Open API Gateway.
- The level of maturity of each technology and tool.
- The effort for the implementation of the various components with the specific technologies and tools.
- The integration options and the level of compatibility between them

For each of the two modules incorporated in the INFINITECH Open API Gateway, different technologies, tools and frameworks were selected as a result of this selection process. In detail, the Gateway module that constitutes the core module of the component's architecture is based on the Java programming language and specifically Java version 11. To facilitate the implementation of the module, the dominant open-source Java-based Spring Boot Framework is utilised [9]. Spring Boot is a powerful lightweight application framework that facilitates the implementation of Java-based enterprise applications in an efficient and modular manner. Spring Boot offers out-of-the-box functionalities capable of supporting the application development such as the embedded HTTP server and easy integration with a large variety of Java-based libraries and technologies. The already embedded and well-established open-source Tomcat servlet container [10] is leveraged in the implementation of the module. With regard to the Circuit Breaker pattern implementation, the Spring Cloud Circuit Breaker [11] is exploited in order to provide the required abstraction layer and out-of-the-box integration with multiple open-source implementations such as Netflix Hystrix, Resilience4j, Sentinel and Spring Retry.

The Service Registry module is based on Consul which is the well-established open-source solution for service discovery and health checking [12]. Consul provides a sophisticated service registry that enables the self-registration and self-deregistration of microservices, maintaining their dynamic network location and providing simplified service discovery via an open and extensible API and an HTTP interface. Furthermore, it provides real-time health monitoring in order to track the availability and operational status of the registered microservices. Additionally, Consul provides the registration client in multiple programming languages that is integrated within the implementation of each microservice. The registration client undertakes the responsibility of self-registering the respective microservice during the microservice start-up process and the self-deregistration of the microservice during the microservice shutdown process.

In addition to this, the integration of the API Gateway with the Service Registry is enabled with the Spring Cloud Consul that covers the integration aspects of Spring Boot applications with Consul.

The following table summarises the main technologies and tools which will be used in the implementation of the INFINITECH Open API Gateway:

Table 8: INFINITECH Open API Gateway list of technologies

Module Name	Software Artefact Name
Gateway	- Java version 11 - Spring Boot Framework - Tomcat Servlet Container - Spring Cloud Circuit Breaker - Spring Cloud Consul
Service Registry	Consul

5 Conclusions

The purpose of the deliverable at hand, entitled "D5.10 - "Data Management Workbench and Open APIs - I", was to report the preliminary outcomes of the work performed within the context of Task 5.5 "OpenAPI for Analytics and Integrated BigData/AI WorkBench" of WP5. To this end, the deliverable documented the challenges of accessing the microservices implementations of added-value offerings in INFINITECH and the detailed design specifications of the INFINITECH Open API Gateway component that eliminates these challenges with a sophisticated (high-TRL) solution that has potential for novel extensibility.

At first, a thorough analysis of the two most common approaches that can be used to solve the problem of accessing underlying microservice-based added-value functionalities from the clients were presented in detail, namely the direct client-to-microservices pattern and the API Gateway pattern. For each approach, the list of advantages and disadvantages were presented. Furthermore, the reasons for selecting the API Gateway pattern to build the appropriate solution within the context of INFINITECH were also highlighted.

Following the analysis of the two approaches, the initial design decisions and design specifications of the INFINITECH Open API Gateway component were presented. The deliverable presented the core design decisions that were taken during the design phase in order to formulate the core aspects of the INFINITECH Open API Gateway. The design decisions had driven the design specifications and the main functionalities that are offered by the INFINITECH Open API Gateway, as well as the modular architecture of the designed solution. The INFINITECH Open API Gateway component is composed of two core modules, namely the Gateway and the Service Registry. Each component was designed with a solid role and operates under a clear context undertaking a specific set of functionalities, and the two modules are combined in order to formulate the designed solution. Besides the design specifications of the modules, the list of supported use cases was documented describing in detail the execution steps by each involved client, accompanied by the detailed sequence diagrams that depict the interactions between the modules and the clients.

Finally, the deliverable presented the list of baseline technologies and tools which will be exploited and combined for the implementation of the INFINITECH Open API Gateway component. The list is composed of mature and well-established technologies and tools which will be leveraged and smoothly integrated in the proposed solution, effectively addressing the requirements elicited by the defined design specifications.

It should be noted at this point that the current deliverable constitutes the first report of the work performed within the context of Task 5.5. In accordance with the INFINITECH Description of Action, two more iterations will be delivered on M22 and M30, with deliverable D5.11 and D5.12 respectively. The outcomes of this deliverable will drive the implementation activities of the INFINITECH Open API Gateway component. Nevertheless, as the project evolves and the implementation of the specific component is a living process that will last until M30, updates and refinements will be introduced based on the feedback that will be collected, as well as any requirements that may arise. The updated documentation of both the design specifications as well as the implementation details will be documented in the upcoming versions of this deliverable.

Table 9 - Conclusions (TASK Objectives with Deliverable achievements)

Objectives	Comment
Exploit the API Gateway pattern offerings.	The proposed solution successfully exploits the offerings of the API Gateway pattern related to the effective and efficient service discovery and straight-forward access to the dynamically deployed microservices, the elimination of multiple server round trips and the hiding of the microservice's architecture details and complexity from the clients.
Provide a single point entry for the added-value offerings functionalities of INFINITECH.	The INFINITECH Open API Gateway delivers the required single entry-point for all the ML/DL microservices and their exposed Open APIs that will be available in INFINITECH. Furthermore, it provides the basis for the support of additional microservice-based added-value functionalities in INFINITECH.

Design and deliver the required solution that enables the discovery and consumption of the ML/DL microservices of INFINITECH and their exposed Open APIs.	The detailed design specifications of the INFINITECH Open API Gateway are delivered with the current deliverable; its offered functionalities successfully cover both the discovery and the consumption of the underlying ML/DL microservices of INFINITECH as well as of their exposed Open APIs. The implementation of the INFINITECH Open API Gateway is an ongoing process that will last until M30.

Table 10: – (map TASK KPI with Deliverable achievements)

KPI	Comment
API Gateways available for enabling the access to INFINITECH ML/DL microservices.	*Target Value = 1* The specific KPI is successfully achieved with the presented solution, namely the INFINITECH Open API Gateway, whose detailed design specifications were documented in the current deliverable.
Coverage of ML/DL microservices exposing Open APIs.	*Target Value = 100%* Per the design specifications documented in the current deliverable, the INFINITECH Open API Gateway is capable of covering all ML/DL microservices that are exposing Open APIs within the context of INFINITECH, hence the coverage is 100%.
Number of services exposed to the clients of the API Gateway.	*Target Value >= 4* The INFINITECH Open API Gateway exposes five main services: a) the request handling service, b) the discovery service of the Open APIs of the ML/DL microservices, c) the ML/DL microservices service registry, d) the self-registration service, and e) the self-deregistration service.

Appendix A: Literature

[1]   "Microservices.io," [Online]. Available: https://microservices.io/. [Accessed 20 October 2020].

[2]   M. Dudjak and G. Martinović, "An API-first methodology for designing a microservice-based Backend as a Service platform.," *Information Technology and Control,* vol. 49, no. 2, pp. 206-223, 2020.

[3]   F. Montesi and J. Weber, "Circuit breakers, discovery, and API gateways in microservices," 2016.

[4]   "Self-Registration," 2020. [Online]. Available: https://microservices.io/patterns/self-registration.html. [Accessed 10 October 2020].

[5]   "3rd-Party Registration," 2020. [Online]. Available: https://microservices.io/patterns/3rd-party-registration.html. [Accessed 15 October 2020].

[6]   D. Taibi, V. Lenarduzzi and C. Pahl, "Architectural patterns for microservices: a systematic mapping study," 2018.

[7]   "Open API specification," 2020. [Online]. Available: https://swagger.io/specification/#:~:text=The%20OpenAPI%20Specification%20(OAS)%20defines,or%20through%20network%20traffic%20inspection.. [Accessed 20 October 2020].

[8]   "Circuit Breaker Pattern," 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/patterns/circuit-breaker. [Accessed 10 October 2020].

[9]   "Spring Boot Framework," [Online]. Available: https://spring.io/projects/spring-boot. [Accessed 20 October 2020].

[10]   "Apache Tomcat," [Online]. Available: http://tomcat.apache.org/. [Accessed 15 October 2020].

[11]   "Spring Cloud Circuit Breaker," [Online]. Available: https://spring.io/projects/spring-cloud-circuitbreaker. [Accessed 21 October 2020].

[12]   "Consul," [Online]. Available: https://www.consul.io/. [Accessed 10 October 2020].