Transforming integrations of an organization from point-to-point to iPaaS

Lappeenranta–Lahti University of Technology LUT

Master’s Programme in Software Product Management and Business, Master’s thesis

2023

Tomi Laakkonen

Examiner(s): Professor, Kari Smolander

ABSTRACT

Lappeenranta–Lahti University of Technology LUT
LUT School of Engineering Science
Software Engineering

Tomi Laakkonen

Transforming integrations of an organization from point-to-point to iPaaS

Master’s thesis
2023
56 pages, 3 figures, 5 tables, 1 appendix
Examiner(s): Professor Kari Smolander
Keywords: iPaas, API Management, integrations, API, interfaces, Business Finland

The focus of our research is on Business Finland, a governmental organization in Finland that has historically used a decentralized approach for its applications and system integrations. We conducted interviews with Business Finland's employees to understand the evolution of the integration domain over the years and by studying the literature. Our study also aimed to identify the pain points experienced with Enterprise Service Bus (ESB), such as a lack of strategy, leadership, ownership, and functionalities.

Through our literature review, we found that Integration Platform as a Service (iPaaS) is the most modern way to implement integrations. Therefore, we proposed an API management solution as the first centralized iPaaS component for Business Finland. This architecture can be replicated by other organizations to help them adapt iPaaS and overcome the pain points of ESB.

TIIVISTELMÄ

Lappeenrannan–Lahden teknillinen yliopisto LUT

LUT Teknis-luonnontieteellinen

Tietotekniikka

Tomi Laakkonen

Organisaation yhdestä-yhteen integraatioiden muuttaminen iPaaS:ksi

Tietotekniikan diplomityö

2023

56 sivua, 3 kuvaa, 5 taulukkoa, 1 liite

Tarkastaja(t): Professori Kari Smolander

Avainsanat: iPaaS, rajapinnat, API-hallinta, integraatiot, API, Business Finland

Tämä tutkimus keskittyy Business Finlandiin, suomalaiseen valtionhallinnon organisaatioon, joka on historiallisesti hyödyntänyt hajautettua lähestymistapaa sovellusten ja järjestelmien integrointiin. Tutkimuksessa tutkittiin kuinka integraatiot ovat kehittyneet vuosien aikana haastattelemalla Business Finlandin työntekijöitä, sekä tutkimalla kirjallisuutta. Tutkimuksessa tunnistettiin kipupisteitä, joita koettiin Enterprise Service Busin (ESB) kanssa. Näitä oli muun muassa strategian, johtajuuden, omistajuuden ja toimintojen puute.

Kirjallisuudesta opimme, että Integration Platform as a Service (iPaaS) on tällä hetkellä modernein tapa tuottaa integraatioita. Tämän takia ratkoimme ongelmaa tuomalla API:en hallinnan ensimmäisenä keskitettynä iPaaS-tuotteena Business Finlandille. Tätä arkkitehtuuria voi jatkossa hyödyntää muissa organisaatioissa ratkaisemaan ESB:n aiheuttamia ongelmia.

SYMBOLS AND ABBREVIATIONS

Abbreviations

API	Application Programming Interface
EAI	Enterprise Application Integration
ESB	Enterprise Service Bus
IAC	Infrastructure-as-Code
IPAAS	Integration Platform-as-a-Service
REST	Representational state transfer
SOAP	Simple Object Access Protocol

Table of contents

Abstract

Abbreviations

1 Introduction 7
2 Background 9
2.1 Point-to-point integrations 12
2.2 Enterprise Application Integration (EAI) 14
2.3 Enterprise Service Bus (ESB) 16
2.4 Integration Platform as a Service (iPaaS) 17
2.5 Application Programming Interface (API) 19
2.5.1 RESTful API 20
2.5.2 SOAP API 21
2.5.3 GraphQL 22
2.6 API Management 23
2.7 Infrastructure-as-Code 28
2.8 Cloud computing 30
3 Case study process 31
3.1 Case study presentation 31
3.2 Research process 32
4 Migration needs and implementation in the case organization 33
4.1 Business Finland’s establishment 33
4.2 Business Finland’s integrations 34
4.3 Improvement Needs and Actions 36
4.4 Migrating to iPaaS 37
4.5 Requirements for API management solution 38
4.6 Proposed solution 40
4.6.1 Project implementation 42
4.6.2 Solution architecture 44
4.7 Discussion 49

5 Conclusions 51
References 53

1 Introduction

Business Finland is a Finnish governmental organization that has traditionally employed a decentralized approach to application and system integrations. Existing integration practices have predominantly been ad-hoc, characterized by point-to-point connections. Additionally, some autonomous applications have independently implemented their own Enterprise Service Bus (ESB) platforms, but ESB has never been universally adopted across the organization. Building upon this context, the primary objective is to investigate the advantages of transitioning to an Integration Platform as a Service (iPaaS) environment, underpinned by API-based integrations. We aim to discern the strategies for API management and to chart a viable path toward iPaaS implementation within Business Finland's hybrid computing landscape.

The overarching goal of this research is to comprehend the benefits of iPaaS adoption within an organization that continues to operate in a hybrid mode, encompassing both on-premises and cloud-based infrastructure. Furthermore, we intend to scrutinize API-based integrations as a pivotal facet of iPaaS and elucidate the practical strategies for their incorporation within the organization.

To this end, we employ a multi-faceted approach. Firstly, we embark on an extensive exploration of Business Finland's organizational prerequisites and leverage existing literature to establish a clear rationale for the integration of iPaaS within its operational framework. This investigation is particularly focused on API management as an integral component of the broader API ecosystem, which becomes crucial as the organization transitions towards API-based integrations.

The following questions will be answered in this research:

1) What compelling factors prompt organizations to shift from the traditional point-to-point integration paradigm to the adoption of iPaaS?
2) What transformative alterations occur within an organization's IT infrastructure and architectural landscape as it embraces iPaaS?

3) How do organizations navigate the selection of the most suitable iPaaS solution tailored to their specific needs, and what critical considerations inform this decision-making process?

Our research methodology commences with an in-depth examination of the current state of Business Finland's operational environment. This analysis not only brings to light the limitations and inefficiencies of the existing system but also forms the foundation for our exploration of how iPaaS can address these challenges. Subsequently, as our comprehension of iPaaS benefits matures, we delve into elucidating the prerequisites unique to Business Finland's use case, thereby narrowing down the selection of an iPaaS solution that aligns seamlessly with the organization's needs. Furthermore, we aim to devise a comprehensive API management strategy, integral to the successful implementation of iPaaS for API-based integrations.

2 Background

This chapter aims to provide a comprehensive literature review of terminologies and concepts associated with application and system integrations. The objective is to establish a solid foundation of understanding for the subsequent chapters, enabling a deeper exploration of the project work. The literature review delves into several integration terminologies, including iPaaS, APIs, EAI, ESB, point-to-point integrations, IaC, and cloud services. To ensure the credibility and reliability of the sources, renowned research databases such as ResearchGate and Google Scholar were utilized. A variety of keywords related to the aforementioned terminologies were employed, allowing for a comprehensive exploration of the literature.

The literature review commences by reviewing point-to-point integrations. According to Gulledge (2006) point-to-point integrations are direct connections between source and target systems where each integration significantly increases the number of connections. Scholarly literature is analyzed to understand the advantages, disadvantages, and challenges associated with point-to-point integrations. By exploring the literature, the review sheds light on the scalability, maintainability, and performance implications of this integration approach.

Next, the literature review delves into the concept of EAI. McKeen et al. (2002) delineated Enterprise Application Integrations (EAI) as centralized hub-based implementation that was developed after point-to-point and hardcoded integrations. By consulting relevant research papers and industry reports, the review offers insights into the principles, challenges, and best practices of EAI. Furthermore, the review explores the role of EAI in promoting efficient data exchange, process automation, and enhanced collaboration within organizations.

The subsequent section of the literature review centers around ESB. Menge (2007) described the Enterprise Service Bus (ESB) as a hub to connect applications and services. By examining scholarly articles and publications, the review elucidates the functionalities, benefits, and implementation considerations associated with ESB. The exploration of ESB-related literature provides a comprehensive understanding of its role in facilitating seamless communication, service orchestration, and message transformation.

The literature review continues by examining the concept of iPaaS. According to Marian (2012) integration Platform-as-a-Service (iPaaS) is a set of cloud services that are used to achieve application and data integration capabilities. iPaaS is used to integrate from cloud-to-cloud, cloud-to-on-premises, on-premises-to-on-premises and business-to-business applications and data. Relevant scholarly articles and publications were consulted to gain a comprehensive understanding of the functionalities, benefits, and challenges associated with iPaaS. By exploring the literature, key insights regarding iPaaS adoption, implementation strategies, and its role in enabling seamless integration were obtained.

The review then shifts its focus to APIs, which play a crucial role in enabling communication and data exchange between different software applications. Ranga and Soni (2019) described an Application Programming Interface (API) as a way for two different software entities to communicate with each other by mostly using Hypertext Transfer Protocol (HTTP) and these APIs follow commonly either Representational State Transfer (REST) or Simple Object Access Protocol (SOAP) architectural frameworks. A comprehensive exploration of API-related literature was conducted to uncover the diverse applications, standards, and protocols associated with APIs.

After APIs, the review centers on API management. As elucidated by Mathijssen et al. (2020), API management constitutes a comprehensive platform employed for the oversight, publication, monitoring, and regulation of APIs. Subsequently, our literature review endeavors to expound upon the operational intricacies of API management within organizational frameworks, delineating the requisite teams essential for service implementation. Moreover, an exploration into the lifecycle of APIs within the purview of API management services is undertaken to discern the nuanced processes involved.

Subsequently, the literature review explores the concept of IaC. Achar (2021) described Infrastructure-as-Code (IaC) as a process that treats the software infrastructure as code. It is used to track changes in infrastructure configuration, rollbacks, and prediction in a way that the outcome should be the same in each environment and consistency. By consulting relevant research papers and industry publications, the review uncovers the benefits of implementing IaC for application and system integrations. The exploration of IaC-related literature provides valuable insights into automating infrastructure deployment, configuration management, and facilitating continuous integration and delivery.

The final section of the literature review focuses on cloud services as they pertain to application and system integrations. Huth and Cebula (2011) defined cloud services as mostly subscription-based models where the consumer can buy computing resources as allocation when needed without upfront investment. By examining scholarly articles and industry reports, the review elucidates the role of cloud services in optimizing computing costs for the consumer organization. The exploration of cloud services provides insights into the advantages and challenges.

The chapter provides a comprehensive literature review of various terminologies and concepts associated with application and system integrations. By exploring scholarly literature from renowned research databases, a comprehensive understanding of iPaaS, APIs, API management, EAI, ESB, point-to-point integrations, IaC, and cloud services is obtained. The review serves as a solid foundation for comprehending the subsequent chapters, allowing for a more in-depth exploration of the project work.

During our investigation, it became apparent that the initial integration strategy was rooted in point-to-point and hardcoded integrations. Notably, point-to-point integration became a prevalent approach among various organizations. While these integrations were straightforward and easily implemented, a significant challenge emerged as the number of interfaces escalated considerably with the integration of multiple systems. (Risimic 2007)

To mitigate the challenges associated with point-to-point integrations, the integration strategy transitioned towards Enterprise Application Integration (EAI) and Enterprise Service Bus (ESB) (Hyrynsalmi 2022). In this paradigm, integrations are compelled to traverse through a centralized hub. Our research revealed that EAI was initially introduced to establish an integration strategy where all connections were routed through a centralized hub. However, the hub-and-spoke model faced criticism due to scalability issues as data volumes increased and vulnerabilities to network issues. (Soomro et al. 2012)

The era of EAI saw the development of ESB, where integrations operated through a service bus distributed across the IT infrastructure. ESB addressed scalability problems encountered by EAI. Nevertheless, both EAI and ESB faced criticism for scalability and reliability, prompting a shift in the integration strategy towards cloud-based Integration Platform as a Service (iPaaS). (Hyrynsalmi 2022)

iPaaS was identified as a comprehensive set of cloud-based integration capabilities, provided either by the cloud platform itself or as a turnkey service by iPaaS vendors (Marian 2012). These vendors offered scalable integration services akin to other cloud computing services, requiring no upfront investments. iPaaS solutions were commended for their cost-effectiveness and provision of low-code platforms for rapid integration development. Despite being recognized for enhanced security, iPaaS faced criticism for lack of control, particularly concerning security issues that could potentially arise from the cloud provider. (Hyrynsalmi et al. 2021)

The focus of this thesis centered on the API management capabilities offered by many iPaaS vendors. API management was deemed essential for organizations developing APIs, ensuring effective management, control, and monitoring. The research delved into the critical role API management plays in publishing external, internal, and partner APIs (Mathijssen et al. 2018). Additionally, insights were provided on how the API lifecycle should be navigated within the API management framework, detailing the necessary steps and the teams and resources integrated into the API management ecosystem. (Bondel et al. 2022)

Given the cloud-based nature of iPaaS, our investigation extended to explore the advantages of Infrastructure as Code (IaC) when applied to iPaaS. IaC emerged as a powerful capability for managing cloud infrastructure, treating it as code within a repository. Applauded for its reliability and predictability, IaC allows the cloning of the same infrastructure architecture for each environment. Version control facilitates easy rollback to prior versions, and detailed logs support issue investigation. Furthermore, embracing IaC aligned the development team with the DevOps process, where they actively engage in infrastructure maintenance (Achar 2021).

2.1 Point-to-point integrations

Point-to-point integrations are widely recognized as a conventional approach to enterprise integrations, with the integration itself being relatively inexpensive and fast to implement. However, problems arise when the integration needs to be expanded to provide connectivity to additional systems. The complexity of the integration increases when a new dependency

is added to an already complex topology, often referred to as spaghetti code, as highlighted by Risimic (2007).

Furthermore, Gulledge (2006) posits that point-to-point integrations are the most expensive form of integration. The integration consists of a data model of source and target systems that are connected by code, enabling the transfer of information between them. Each additional interfaced component significantly increases the number of maintained integrations, as noted by Gulledge. To illustrate the problem, Risimic provides a formula to calculate the number of necessary connections in the topology where n applications are given by n*(n-1)/2. For instance, if there are eight applications in total, they would require 28 individual integrations to work with one another.

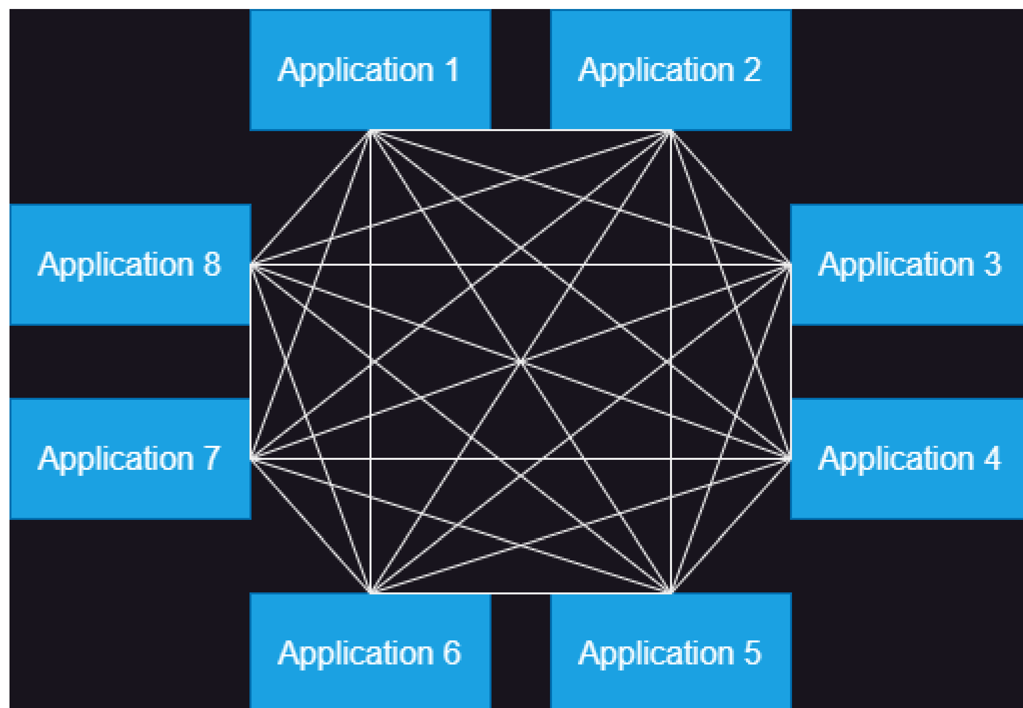

Figure 1. Point-to-point integration topology (Risimic, 2007)

According to Hyrynsalmi (2022), point-to-point integration is the simplest way to connect systems together which transformed to e.g., EAI to increase efficiency as the number of integrations kept increasing while the number of systems also increased. EAI is described as a process where applications and systems are integrated between enterprises. Hyrynsalmi also described that ESB was developed as a middleware model to address these challenges with point-to-point. However, also these models had issues with scalability and performance which then led to integration platforms to the cloud and iPaaS.

2.2 Enterprise Application Integration (EAI)

As outlined by Soomro et al. (2012), the categorization of Enterprise Application Integration (EAI) revolves around its architectural design. The authors expounded on three principal architectural paradigms: point-to-point topology, hub-spoke topology, and bus topology. Point-to-point topology represents a conventional approach wherein distinct applications communicate via a direct channel.

In contrast, the hub-spoke topology employs a centralized intermediary (the hub), with individual applications connected to it (the spokes). This architecture fosters a flexible coupling mechanism, enabling asynchronous communication among applications through the central hub. However, the hub-spoke model drew criticism due to its limited scalability as message volume escalates and its susceptibility to network failures. A central hub failure could disrupt the entire integration network.

Addressing these shortcomings, the bus topology was introduced as a refinement. Similar to hub-spoke, it relies on centralized routing, yet it diverges by employing a distributed bus structure to manage communication tasks. The components constituting the bus can be distributed across diverse locations and duplicated for enhanced scalability. This bus-based arrangement serves as the foundation for the Enterprise Service Bus (ESB).

One salient observation offered by the authors pertains to the issues surrounding EAI's terminology. They noted a prevalent lack of awareness that EAI is fundamentally an architectural principle rather than a singular product. Moreover, instances of EAI implementation without a coherent integration strategy were highlighted as problematic.

In summation, Soomro et al. (2012) delineated the architectural classifications within EAI, emphasizing the interplay of point-to-point, hub-spoke, and bus topologies. They underscored the nuanced nature of EAI as an architectural guideline and underscored the importance of strategic implementation in achieving successful integration outcomes.

McKeen et al. (2002) expounded upon the historical context of enterprise application integrations (EAI), elucidating the transition from formerly hardcoded and point-to-point integrations to a centralized hub-based approach upon the implementation of EAI. This paradigm shift facilitated the relocation of integration logic from individual applications to the centralized hub, thereby fostering the development of decoupled applications.

Nonetheless, a prescient concern emerged regarding the long-term viability of this hub, termed middleware, which was anticipated to eventually evolve into legacy software. Notably, the respondent focus group of McKeen's research acknowledged the inevitability of software obsolescence, yet underscored that the advantages conferred by middleware transcended this eventuality. Central to this perspective was the empowerment bestowed by middleware in facilitating seamless integrations, thereby amplifying return on investment. Furthermore, middleware was deemed instrumental in enforcing uniform development standards across the organization.

McKeen et al. divided EAI into four distinct realms: data integration, process integration, application integration, and inter-organizational integration. Data integration materialized as a strategic solution to address incongruities stemming from disparate databases, such as divergent product codes. This was ameliorated through the establishment of a singular database source harmonizing data across applications, thus simplifying the integration of heterogeneous systems. Integrative endeavors encompassed data transformation and normalization, culminating in the efficient dissemination of data to various consumer nodes.

Application integration, a cornerstone of EAI, entailed the amalgamation of disparate applications into a seemingly cohesive user experience. This fusion was achieved through asynchronous messaging, albeit the authors noted that messaging-oriented middleware (MOM) had already begun to exhibit limitations in adhering to emerging standards such as zero-latency and straight-through processing.

Process integration, synonymous with event-oriented or transaction-oriented integration, was underscored as a mechanism to synchronize events or transactions originating in one application with pertinent business processes in another. This orchestration facilitated the harmonization of diverse systems in support of comprehensive business workflows.

Lastly, inter-organizational integration, akin to process integration, underscored the synchronization of processes across two or more distinct organizations, predominantly in the context of business-to-business partnerships. In essence, this facet extended the EAI framework beyond individual organizational boundaries, nurturing collaborative processes across interconnected entities.

Erasala et al. (2003) expounded upon EAI as a form of integration facilitating the exchange of information and orchestration of business processes. Additionally, they provided an

intricate elucidation of the EAI architectural framework, comprising four fundamental layers: process management, transformation services, distribution services, and communication services.

In this architectural schema, the distribution layer assumes the pivotal role of routing messages, functioning either through request and response paradigms or publish and subscribe mechanisms. The transformation layer, on the other hand, undertakes the task of data conversion to a format compatible with the destination application's requirements. Concurrently, the process management layer is vested with the responsibility of delineating and enforcing business rules integral to the integration and associated business processes. Moreover, it exercises authority over the transformation layer, enabling the realization of application integrations within the context of business processes.

Erasala et al. articulated the communication services layer as typically comprising a message queue mechanism, adept at facilitating the synchronous or asynchronous transmission of messages, bolstered by a store-and-forward attribute. Their insights underscored that IT components commonly interface with this communication layer through connectors or adapters, enabling seamless integration with the broader EAI infrastructure.

2.3 Enterprise Service Bus (ESB)

The Enterprise Service Bus (ESB) is a critical component of Service-Oriented Architecture (SOA) that facilitates communication between various services and applications within an organization. According to Schmidt et al. (2005), the ESB is essential in making SOA a reality because it provides a flexible and scalable platform for integrating and managing different services.

Menge (2007) describes the ESB as a central hub for managing communication between different services and applications, providing a set of services and tools for handling various aspects of SOA. These include message routing, data transformation, and security, among others. The ESB also provides a framework for managing different protocols and interfaces, enabling organizations to integrate different services and applications regardless of their underlying technology.

The ESB architecture consists of three primary layers: the messaging layer, the mediation layer, and the service management layer (Schmidt et al., 2005). The messaging layer provides a platform for exchanging messages between different services and applications. The mediation layer provides a set of tools for handling message routing, data transformation, and other tasks necessary for communication between different services. The service management layer provides a set of tools for managing different services and applications, including monitoring, discovery, and service-level agreements.

2.4 Integration Platform as a Service (iPaaS)

Marian (2012) described Integration Platform as a service as cloud service that is used for cloud, business-to-business, and on-premises integrations. Normal use case scenarios are cloud-to-on-premises integration, cloud-to-cloud integration, on-premises-to-on-premises integration, and B2B (business-to-business) integration. Usually, the iPaaS offers a package of cloud services for certain integration actions such as development tools, repositories, orchestration, transformation, service virtualization, etc. On iPaaS the integrations are developed in a way where the integration itself is described as an integration flow. The integration flow is developed by using the iPaaS client that is connected to the iPaaS environment. Another way to offer iPaaS is embedded iPaaS where only the iPaaS provider develops the integrations.

Cestari et al. (2020) introduced four iPaaS providers that were the most used and reviewed services in 2019. According to their research, these providers were Dell Boomi, IBM Cloud Integration, Microsoft Azure Integration Services, and Informatica Intelligent Cloud Services. Dell Boomi has several features such as core application or data integration, API management, B2B management, and workflow automation. Boomi had over 1500 unique connectors at that time but was said to lack customer support and did not have features like event stream analytics or data virtualization. IBM Cloud Integration's features are called IBM API Connect, App Connect, MQ, Event Streams, IBM Aspera, Secure Gateway Service, and IBM BPM. IBM Cloud Integration had over 100 connectors in 2019, but the consumer can also develop custom connectors to support their needs. IBM Cloud Integration was said to be the best for integrations between IBM's other products. However, the

customer service was criticized, and the solution was described as difficult to implement. Microsoft Azure Integration Services uses Azure services such as Azure Logic Apps, Azure API Management, and Azure Service Bus. Separately data integration and workflow automation services are also offered. In 2019, Azure offered over 360 connectors. Azure's solution was criticized for causing vendor locking issues but also praised for offering good support between other Microsoft products. Lastly, Informatica Intelligent Cloud Servies (IICS) offers features such as API management, data integration, B2B integration, and in 2019 had over 400 connectors and the possibility to develop custom connectors as well. IICS was criticized for its price being more expensive than the competitors and lack of features like event processing.

Hyrynsalmi et al. (2021) elucidated the phenomenon of cloud-based integrations as a consequential outcome of the transformation of the software industry into a service-centric paradigm. This transition led to the emergence of cloud-based integrations, denoted as Integration Platform as a Service (iPaaS) and Integration Software as a Service (iSaaS), which were noted for their augmentation of development and operational (DevOps) efficiencies, as well as the enhancement of integration governance. Within the scope of their investigation, the authors conducted comprehensive interviews with numerous organizations, aiming to capture their perspectives on integration practices.

The interviewees conveyed divergent viewpoints concerning the efficacies of on-premises integrations vis-à-vis cloud-based counterparts. Specifically, some organizations expressed a preference for on-premises integrations due to perceived cost-effectiveness and swifter implementation, stemming from the absence of intermediaries between the organization and the infrastructure vendor, thereby eliminating dependencies on external servers. However, these on-premises solutions were critiqued for their intricate design and suboptimal code quality. Furthermore, the architectural landscape often appeared disorganized, attributed to the tendency of developers to focus solely on integration tasks, neglecting holistic architectural design, consequently yielding untidy outcomes.

Conversely, interviewees with experience in utilizing iPaaS platforms attested to the scalability and structural clarity offered by cloud-based integration solutions. Notably, the utilization of low-code options within these platforms emerged as a cost-efficient alternative, obviating the need for external consultants to troubleshoot operational challenges. This phenomenon was ascribed to the user-friendly graphical interface, facilitating

comprehension and ease of use among employees. From a security perspective, iPaaS platforms were generally regarded as superior to on-premises environments, albeit with the caveat that potential security breaches might emanate from the cloud provider's domain.

The discourse also encompassed the intricate pricing structures associated with iPaaS solutions, often deemed inscrutable and capable of yielding unexpected financial implications. Despite this complexity, interviewees uniformly advocated for the strategic adoption of cloud-based integration platforms, should such opportunities be viable within the organizational context.

Furthermore, Hyrynsalmi et al. (2021) delineated the demographic composition of companies engaging in iPaaS solutions during their research. Forefront enterprises were found to predominantly embrace iPaaS solutions, whereas others exhibited a propensity for on-premises integration strategies. A degree of skepticism towards contemporary iPaaS solutions was also detected among certain entities. The authors illuminated that Finnish software companies, especially those categorized as small or medium-sized, exhibited a predilection for on-premises integration approaches. This preference was attributed to either deliberate strategic decisions or the prevailing belief that integration activities should be conducted on-premises.

2.5 Application Programming Interface (API)

The Application Programming Interface (API) is identified as “a set of well-specified routines” by Andersson (2002). Furthermore, Andersson describes API as a way for a developer to develop programs independently from the hardware or the software.

Biehl (2016) elucidates that a conventional solution framework encompasses an application, APIs, and a backend component. Within this structure, end-users interact with the application, which in turn utilizes data furnished by the API. It is imperative to note that the API functions as an intermediary, retrieving requisite data from the backend without direct end-user engagement. The application is solely responsible for initiating requests to the API to procure the necessary data.

Furthermore, Biehl (2016) delineates the imperative architectural decisions pertinent to APIs, including the pivotal choice between RESTful and SOAP protocols. Subsequent to

their development, APIs are meticulously documented and communicated through specialized description languages, notably OpenAPI and RAML, thereby providing a robust framework for their implementation and utilization in various applications.

Ranga and Soni (2019) elucidated the role of Application Programming Interfaces (APIs) as pivotal conduits facilitating communication between disparate software entities. These interfaces employ the Hypertext Transfer Protocol (HTTP) as a foundational mechanism to instantiate their designated functionalities. Moreover, APIs inherently adhere to one of two predominant architectural frameworks: Representational State Transfer (REST) or Simple Object Access Protocol (SOAP).

2.5.1 RESTful API

Richardson and Ruby (2008) elucidated the concept of RESTful architecture, characterizing it as inherently resource-oriented in the context of web services. The term "resource-oriented" in this instance implies that the HTTP request Uniform Resource Identifier (URI) should unambiguously convey the client's intended action. The authors further delineated that a service cannot be deemed RESTful if there is a mismatch between the HTTP method employed and the information used, or if the URI does not encapsulate the scope of the relevant information.

Similarly, Biehl (2016) expounded on the REST architectural design framework, highlighting its emphasis on fostering clear, concise, user-friendly, and straightforward Application Programming Interfaces (APIs) for consumer utilization.

Ranga and Soni (2019) provide a historical perspective, noting that the Representational State Transfer (REST) architectural style was conceived by Roy Fielding in the year 2000. They enumerate seven cardinal constraints intrinsic to REST. Firstly, there should be a distinct separation between the client and the server, facilitating their independent development. The inception of the design process should be characterized by a Null Style, signifying a clean slate approach. Each request must be inherently stateless, encapsulating all the requisite information to facilitate the execution of corresponding actions. The endpoint should optimally leverage caching mechanisms to enhance efficiency, scalability, and performance metrics. Adherence to a uniform interface is imperative, whereby a single

logical URI is allocated, granting developers the capability to retrieve associated data as necessitated. The architecture should be layered, with each component strategically positioned on distinct layers, thereby obfuscating visibility beyond their respective layers. Although this layered approach engenders processing delays, it is crucial for the systematic organization of the architecture. Lastly, REST provides the flexibility to download and execute code snippets as required, thereby minimizing the volume of code necessitated on the client side.

2.5.2 SOAP API

According to Ranga and Soni (2019), Simple Object Access Protocol (SOAP) uses XML format to communicate on the network. SOAP uses HTTP to transfer the data over the network and each SOAP message has an envelope. Inside the envelope is the data that includes a header which is used to provide e.g., authorization data. After the header the body is given which includes the XML data, either request or response. Lastly, it may contain fault which is a block that contains optional error data if the server failed to process the request.

SOAP (Simple Object Access Protocol) is a messaging protocol used in web services to facilitate communication between different applications. It relies on the XML format to encode messages and can be transmitted over various transport protocols, such as HTTP and SMTP. SOAP is a widely used standard and has been implemented by many vendors, making it a reliable choice for enterprise-level systems (Belqasmi et al., 2012).

SOAP's reliability is partly due to its support for different types of security mechanisms, such as digital signatures and encryption. This feature makes it a secure protocol for exchanging sensitive data across different platforms. Additionally, SOAP supports different styles of messaging, such as request-response and one-way messaging, making it a versatile protocol for different types of applications (Katayama et al., 2010).

However, SOAP has some disadvantages when compared to RESTful web services. For instance, SOAP messages are typically larger than their RESTful counterparts due to their reliance on XML. This makes them less efficient for mobile and low-bandwidth environments. Additionally, SOAP relies heavily on XML schema for validation, which can make it difficult to modify or extend the schema in the future (Mulligan & Gračanin, 2009).

Belqasmi et al. (2012) conducted a case study comparing SOAP-based and RESTful web services for multimedia conferencing. They found that SOAP-based web services were more suitable for handling complex multimedia sessions due to their support for advanced features like security, reliability, and transaction management. However, they noted that RESTful web services were more suitable for simple tasks and provided better performance and scalability.

Katayama et al. (2010) developed TogoWS, an integrated platform that provides both SOAP and RESTful APIs for interoperable bioinformatics web services. They found that the combination of both SOAP and RESTful APIs provided greater flexibility in terms of interoperability with other platforms and systems.

Mulligan and Gračanin (2009) conducted a comparison between SOAP and RESTful implementations of a service-based interaction independence middleware framework. They found that while both SOAP and RESTful APIs provided similar functionality, SOAP-based web services required more effort to implement due to their complex messaging structure and support for advanced features.

2.5.3 GraphQL

According to Hartig, and Pérez (2018), GraphQL was initially released by Facebook in 2016 after developing and using it internally for three years. It is an alternative to RESTful interfaces and its main feature is the ability to precisely define the data the consumer wants. The advantage over the RESTful interface is the capability to replace multiple RESTful calls with one GraphQL query.

Hartig and Pérez further elucidated that GraphQL presents a data repository accessible via an interface, allowing for targeted querying to specify the requisite dataset nodes for the end consumer. It has gained prominence as a prevalent alternative to traditional RESTful APIs.

According to Stünkel et. al (2020), web services typically employ one of two protocols, namely SOAP or REST. Furthermore, GraphQL is designed to provide the advantages of both protocols within a unified protocol.

2.6 API Management

Mathijssen et al. (2018) introduced API management as a platform that helps organization to publish their APIs to external, internal, and partner consumers. Furthermore, whenever API is developed it needs to be managed, the documentation must be published, the API has to be monitored and also the utilization has to be analysed.

Mathijssen et al. (2018) conducted a systematic literature review to identify the practices and capabilities of API management. They found that API management involves 114 practices and 39 capabilities in total. From these practices, the most frequently mentioned practices were caching, OAuth authentication, load balancing, rate/quota limiting, usage throttling, activity logging/monitoring, access control, and billing. The most frequently mentioned capabilities were authentication, monitoring, security, analytics, catalog/documentation, API publication/deployment, monetization, and version management.

Bondel et al. (2022) proposed API management patterns for public, partner, and group web API initiatives with a focus on collaboration. The authors identified common challenges and solutions for each type of API initiative, such as versioning, documentation, and governance. According to Bondel et al. (2022), the basic concept of API management consists of a backend, an API gateway, a web API, an API developer portal, and an API-consuming application. According to the researchers, a backend can be any component that provides data or functionality that is made accessible to other stakeholders. Web API makes the functionality or data accessible over the HTTP protocol. The application then integrates with the web API to consume the functionality or the data. API management itself is then supported by two software platforms, the API gateway, and the API developer portal. The API gateway is part of the infrastructure and manages the API provider and consumer at runtime as a reverse proxy. The API developer portal is a way to allow the API provider to communicate information about the APIs to the API consumers.

The API lifecycle consists of several key steps that guide the development, management, and eventual deprecation of an API. Each step plays a crucial role in ensuring the successful implementation and utilization of the API. Bondel et al. defined the API management lifecycle as illustrated in Table 1.:

Table 1. API management lifecycle according to Bondel et al. (2022)

Lifecycle step	Description
Plan and design.	In this step, the API is either newly created or evolved from an existing API. The API provider defines the business goals, target market, and use cases for the API. Additionally, an API specification is created to outline the structure and behavior of the API endpoints.
Implement and deploy.	The implementation of the API takes place in this step, where existing backend functionalities may be reused in an agile manner. Integration frameworks can support the implementation process. The API is then deployed using Continuous Integration/Continuous Deployment (CI/CD) pipelines to ensure efficient and automated deployment.
Test and secure.	The API undergoes testing to validate both functional and non-functional requirements. This includes ensuring that the API functions as expected and meets performance, scalability, and reliability standards. Security measures are also enforced to protect the API from potential exploits and vulnerabilities.
Manage and configure.	In this step, any defects or issues identified in the API are fixed through change requests and versioning. The API management platform is configured to control authentication methods, access rights, and other security measures.
Discovery and on-boarding.	API consumers are provided access to documentation and resources that allow them to discover and understand the functionality of the API. This step may involve self-service capabilities or sandbox environments where consumers can test the API before integrating it into their applications.
Integration and consumption.	API consumers are supported with additional resources such as detailed documentation and Software Development Kits (SDKs) to facilitate seamless integration of the API into their applications. Ensuring the reliability and performance of the API is crucial to keep consumers satisfied, and a support channel is established for reporting defects or issues.
Monitoring and monetization.	The API is continuously monitored to track its usage, performance, and adherence to defined metrics. Monitoring helps identify potential issues and provides insights for optimizing the API. Monetization strategies may also be implemented to generate revenue from the API, such as usage-based billing or subscription models.

Deprecate.	Over time, an API may become outdated or no longer aligned with business objectives. In the deprecation phase, the API provider informs current consumers about the deprecation and discontinuation of the API. Support and updates for the API cease, and eventually, the API is shut off.

By following these lifecycle steps, organizations can effectively manage their APIs, ensure their security and reliability, and provide a seamless experience for API consumers. The lifecycle approach allows for structured and controlled development, maintenance, and retirement of APIs, ensuring their continued value and relevance.

The API ecosystem involves various stakeholders who play distinct roles in the management and utilization of APIs. Each stakeholder contributes to different aspects of the API lifecycle and has specific responsibilities. Bondel et al. presented the list of API management stakeholders as illustrated in Table 2.

Table 2. API management stakeholders according to Bondel et al. (2022)

Stakeholder	Description
API provider	The API provider is responsible for managing the API management platform, the web API itself, and the API developer portal. The API provider team consists of individuals from both the business and technical sides. The business roles focus on defining and achieving business goals, while the technical roles ensure the fulfillment of technical key performance indicators (KPIs). The API provider team oversees the entire API management lifecycle, from design and development to deployment and maintenance. Backend developers often have a role in implementing the web API and may take on multiple responsibilities within the API provider team.
API consumer	API consumers are responsible for integrating the published web API into their own applications or systems. They can belong to different organizations than the API provider. Based on the relationship between the consumer and provider, APIs are categorized as private, partner, or public. Private APIs are limited to use within the same organization, partner APIs are accessible to selected partners, and public APIs can be accessed by anyone interested in utilizing them.
Backend provider	The backend provider is responsible for designing and maintaining the backend

	infrastructure that supports the web API. This stakeholder collaborates closely with the API provider, as they need to address any reported bugs, incorporate change requests, and ensure the smooth functioning of the API. The backend provider may be part of the same organization as the API provider or a different organization. In some cases, the API provider also operates a marketplace where other API and backend providers can publish their APIs. Alternatively, there are instances where the API provider and backend providers are subsidiaries of the same organization, with their interactions guided by the goals and objectives of each subsidiary.
Upper management	Upper management plays a crucial role in promoting API management within the organization. They provide strategic direction, support, and resources to ensure the successful implementation and adoption of API practices. Upper management's involvement is instrumental in driving organizational alignment, establishing governance frameworks, and fostering a culture of API-centric development and innovation.
Legal department	The legal department is responsible for ensuring the privacy conformance of the APIs. They play a crucial role in reviewing and drafting contracts with partners for the usage of partner APIs. Additionally, the legal department may be involved in addressing legal and compliance-related aspects of API management, such as data protection, intellectual property rights, and regulatory requirements.
Sales and marketing	The sales and marketing teams are responsible for promoting the APIs and driving their adoption among potential consumers. They engage in marketing activities to raise awareness of the APIs, showcase their features and benefits, and enforce compliance with corporate identity guidelines. By effectively positioning and marketing the APIs, these teams contribute to their successful integration and utilization within the market.
Customer support	The customer support team serves as the primary point of contact for API consumers. They handle inquiries, address issues or concerns raised by consumers, and facilitate communication between the consumers and the API provider. The customer support team plays a critical role in ensuring a positive experience for API consumers and helps in

	maintaining strong relationships between the provider and the consumers.

Understanding the roles and responsibilities of each stakeholder within the API ecosystem is essential for effective collaboration, efficient API management, and successful integration of APIs into various applications and systems. By leveraging the expertise and contributions of each stakeholder, organizations can foster an ecosystem that enables seamless API usage, promotes innovation, and drives business growth.

CITO Research (2015) conducted an insightful examination of cloud-based API management, presenting a white paper that delved into the advantages and challenges associated with implementing API management in the cloud. The research placed significant emphasis on the critical factors of security, scalability, and performance in the context of cloud-based API management. Notably, the paper highlighted several key benefits attributed to API management platforms:

- Secured and protected channel between the API gateway and backend systems, ensuring the confidentiality and integrity of data transmission.
- Robust request authentication and authorization mechanisms, guaranteeing that only authorized consumers can access and interact with the APIs.
- Business and operational insights facilitated through comprehensive reports and dashboards, enabling organizations to monitor and analyze API usage, performance, and other relevant metrics.
- Interactive API documentation, providing comprehensive documentation and resources for developers to effectively utilize and integrate with the APIs.
- A façade layer that decouples the internal implementation of APIs, extending the lifespan of older APIs and facilitating seamless transitions during updates or changes.
- Self-service onboarding and efficient off-boarding processes, enabling streamlined and user-friendly experiences for developers and API consumers.

The research paper suggested that hosting API management systems in the cloud offers significant advantages for the vast majority of IT organizations. Cloud hosting provides an array of benefits, including enhanced security, elasticity, scalability, and cost-effectiveness. Cloud platforms offer a level of security that often surpasses that of on-premise data centers,

as cloud providers typically employ superior resources and implement robust processes to ensure the protection of data and infrastructure. Furthermore, the research identified several reasons for deploying API management in the cloud:

- Rapid deployment, enabling organizations to quickly establish and start utilizing the API management platform.
- Multi-region deployment capabilities, leveraging a global content delivery network to make APIs accessible and responsive to consumers across various regions, leading to improved application response times and local caching.
- Centralized integration, allowing all APIs to be accessed from a single location, regardless of the location of the backend systems. This facilitates seamless integration and connectivity between disparate systems and services.
- Elasticity and scalability, enabling organizations to easily scale their API management infrastructure based on demand, ensuring optimal performance during peak usage periods.
- Superior security standards provided by cloud providers, who are held to higher certifications and industry standards compared to corporate data centers. This ensures a robust security posture for the API management platform and associated data.
- Turnkey solutions available out of the box, simplifying the deployment and configuration process, and providing organizations with a ready-to-use API management environment.

The research from CITO provides valuable insights into the advantages of cloud-based API management, reinforcing the notion that hosting API management in the cloud is a compelling choice for most IT organizations.

2.7 Infrastructure-as-Code

According to Achar (2021), Infrastructure-as-Code (IaC) is a process to improve continuously e.g., scalability, fault tolerance, maintainability, and performance. IaC gives a possibility to treat the software infrastructure as code instead of traditional manual

deployment that causes delays and decreased agility in development. IaC is often seen as part of development and operations (DevOps) practices to enhance the collaboration of the developers and operations staff.

In addition, Achar defined that IaC includes principles such as version control, predictability, consistency, repeatability, and composability. Version control is a concept where the source code is maintained as a versioned artifact in the version control environment and the maintained version always corresponds to the current release. In the IaC context, version control is used to manage the infrastructure configuration and changes in the source code repository. In this way, every change can be tracked in the version control environment to see change history that elaborates e.g., who has made a change and what has changed compared to previous commits. Version control helps version rollbacks if an issue occurs on the current version. Predictability is closely related to version control as it ensures that the environment will always be the same as it is defined in the configuration in the code repository. Consistency is defined to ensure that every instance of the same baseline code provides the same environment. It ensures that there are not any inconsistencies as to what may happen when complex infrastructure is built manually. Repeatability refers to the previous principles in a way that the solution can be repeated as many times as needed and it will always provide the same results with the same input. Composability means that the infrastructure is handled in a modular and abstract format that helps organizations to build complex systems. The focus can be switched to the target application as the infrastructure does not need to be understood by the other stakeholders as IaC can perform the complex provisioning.

Achar continues by defining IaC concept. According to Achar, every infrastructure component is declared as a code. This includes directories, packages, utilities, user accounts, and configurations. The IaC’s source code is referred to by a term, e.g., manifests and templates. IaC should have aspects of repeatability, increased reliability, and speed improvements. IaC should offer consistency in the build, such as in different environments like development, QA, staging, and production should be built from the same codebase.

2.8 Cloud computing

Huth and Cebula (2011) expounded upon the concept of cloud computing, delineating it as a service typically offered through subscription-based models, affording customers the capacity to procure a variety of computational resources. Notably, cloud computing transcends geographical constraints inherent in traditional computing setups, imparting a distinct financial advantage. Unlike conventional setups necessitating substantial upfront investments in computing infrastructure, cloud computing allows for judicious allocation of resources, enabling users to selectively procure storage and other services on a just-in-time basis, thereby optimizing cost-efficiency.

The authors categorized cloud services into four distinct types. The first classification is the public cloud, characterized by its accessibility to all interested parties, as implied by the name. The second classification, the private cloud, restricts access to designated groups or organizations. Meanwhile, the community cloud represents a shared cloud environment catering to two or more entities with similar cloud requirements. Lastly, the hybrid cloud leverages a combination of at least two disparate cloud providers, amalgamating different cloud types into a cohesive framework.

Furthermore, Huth and Cebula underscored several disadvantages associated with cloud computing. These drawbacks encompass a lack of transparency regarding the physical location of stored data and the entities with access to it. Security risks were also highlighted, with cloud-based data storage raising concerns in comparison to traditional storage solutions. Additionally, cloud platforms were noted as attractive targets for cyberattacks, further compounding the security challenges associated with this technology.

3 Case study process

To comprehensively analyze the case study and gain insights into Business Finland's organizational history, a detailed examination of the organization's background is necessary. This chapter provides an overview of the factors that led to the establishment of Business Finland and sets the context for the subsequent chapters.

3.1 Case study presentation

Business Finland was established, as a unified organization encompassing investments, internationalization, innovations, and tourism promotion. Business Finland has currently 760 employees and is in 40 foreign locations. Prior to this, these activities were carried out by two separate organizations known as Finpro and Tekes. Tekes had relatively few internal processes in place before the merger. Tekes primarily focused on providing funding services to its customers, and from an integration architecture perspective, its scope was primarily limited to funding-related activities. Initially, integrations were developed using a point-to-point approach with the utilization of Java. Over time, the experience gained from these point-to-point integrations led Tekes to migrate towards adopting an Enterprise Service Bus (ESB) to standardize their integration practices. The chosen software to implement the ESB was Mulesoft, which proved to be a reliable solution with few encountered issues. However, Tekes lacked a clear integration strategy, resulting in a limited assessment of alternative ESB technologies available at that time. On the other hand, Finpro relied solely on point-to-point integrations and did not consider ESB as a viable integration solution during that period.

Currently, Business Finland does not have a clear integration strategy and all of the current issues are inherited from the old organizations.

The primary objective of this thesis is to explore the history of Business Finland as an organization and examine how its evolution aligns with the case study at hand. In order to achieve this objective, a series of interviews were conducted with individuals who possess expertise in the thesis problem domain and have varying levels of knowledge regarding the organization's history.

3.2 Research process

The interviews were conducted using Microsoft Teams, providing a convenient platform for one-on-one interactions. The chosen approach for the interviews was qualitative and semi-structured, allowing for in-depth exploration of the research questions. The interview questions used throughout the study can be found in Appendix 1. The list of interviewed persons can be found in Table 3.

Table 3. Interviewed persons.

ID	Job title	Experience in the current position (counting also Finpro and Tekes)	Experience in the organization in total (counting also Finpro and Tekes)
1	Chief Information Officer	7 years	16 years
2	Information System Architect	3 years	3 years
3	Data Architect	4 years	4 years
4	Data Engineer	4 years	4 years

4 Migration needs and implementation in the case organization

4.1 Business Finland's establishment

The year 2018 Business Finland was established, a unified organization encompassing investments, internationalization, innovations, and tourism promotion. Business Finland has currently 760 employees and is in 40 foreign locations. Prior to this, these activities were carried out by two separate organizations known as Finpro and Tekes. During the interview process, it was revealed that Tekes had relatively few internal processes in place before the merger. Tekes primarily focused on providing funding services to its customers, and from an integration architecture perspective, its scope was primarily limited to funding-related activities. Initially, integrations were developed using a point-to-point approach with the utilization of Java. Over time, the experience gained from these point-to-point integrations led Tekes to migrate towards adopting an Enterprise Service Bus (ESB) to standardize their integration practices. The chosen software to implement the ESB was Mulesoft, which proved to be a reliable solution with few encountered issues. However, Tekes lacked a clear integration strategy, resulting in a limited assessment of alternative ESB technologies available at that time. On the other hand, Finpro relied solely on point-to-point integrations and did not consider ESB as a viable integration solution during that period.

Upon the fusion of the organizations, the integration efforts between Finpro and Tekes were primarily executed using point-to-point integrations. Unfortunately, as Interviewee 1 pointed out, the focus at that time was solely on the functionality of individual applications and systems, with little consideration given to their compatibility and overall integration strategy. Consequently, integration implementations lacked a cohesive and strategic approach. Additionally, it was mentioned that prior to the establishment of Business Finland, the overall environment was relatively manageable due to its smaller scale. However, after the fusion, custom applications were replaced by out-of-the-box solutions, leading to a significantly larger and more complex environment. This expansion presented challenges in terms of maintenance, as the increased number of applications strained the available resources.

Following the organizational fusion, Business Finland acquired new applications and systems, necessitating a re-evaluation of integration strategies. Notably, the adoption of Azure as the public cloud provider prompted the exploration of ways to effectively leverage Azure for integration purposes. As the focus shifted towards integrating the newly acquired technologies, questions emerged regarding how best to incorporate Azure into the integration landscape. This transition marked a shift from a purely functional perspective to a more comprehensive consideration of cloud-based integration possibilities.

The establishment of Business Finland brought together Finpro and Tekes, each with its own integration practices and challenges. The initial point-to-point integrations employed by Tekes eventually led to the adoption of an ESB, specifically Mulesoft, which proved to be reliable. However, a lack of a clear integration strategy hindered Tekes from exploring alternative ESB options. Finpro primarily relied on point-to-point integrations before the fusion, with little consideration given to ESB. The fusion of the organizations introduced new complexities, necessitating the replacement of custom applications with out-of-the-box solutions and the consideration of integration strategies for the expanded environment. The adoption of Azure as the public cloud provider further influenced the integration landscape, shifting the focus towards leveraging Azure for seamless integration capabilities.

4.2 Business Finland’s integrations

After Business Finland was established, it meant that the integration processes of two different organizations were also blended. All the interviewees pointed out that integrations were more or less point-to-point integrations during that time and there was not any clear strategy for how Business Finland’s integrations should be implemented. Interviewee 1 commented that even though Mulesoft was still used for certain parts of the applications those integrations were still point-to-point and a clear strategy was not implemented on how the ESB should be used. The main reason for this was that there was not any planning or development for APIs even though microservice architecture was adopted already 10 years ago. The old legacy was seen as a problem. It was hard to implement architecture or have proper funding for the work. Maintenance and cost efficiency need improvement compared to the past. Interviewee 4 also said that even though Mulesoft was used it did not serve other than internal integrations and all the integrations that were implemented outside of the

monolith were done by using SSIS integrations. These SSIS integrations were described as "glue" to repair integrations on demand. Interviewee 3 also commented on SSIS integrations that the main problem with them was that it was too personified, and the expertise was not widely adopted in the organization or the partner who developed the integrations.

Expertise overall was not spread throughout the organization and all the interviewees commented that everything in the organization was working in silos and each silo had its own ways to implement its integrations. The reasons identified were a lack of strategy that could be handed over to the projects, a lack of leadership from the organization's side, the partners having too much responsibility for the implementations, and a lack of ownership. As the integrations were created on demand for the projects without a strategy, Interviewee 2 described that it also meant inefficiency in that each integration for the same data needs to be built, and tested, requiring resources and time. Because of the silos, Interviewees 2 and 4 described that currently it has been hard to monitor integrations as the monitoring practices are not centralized and partners may or may not have built monitoring for integrations. Interviewee 4 also added that it has not been clear what should be monitored, and the problem has not been fixed as when problems occur in the integrations it has been always integration fixing and the monitoring problem has not been fixed at the same time. The problems that have occurred have affected the whole organization as the same data is very often replicated from the master data side to many different business applications. The problem in this example case was that the data quality was not monitored which resulted in each application integrated to the master data having broken data that cannot be used. Also, testing was seen as a problem because the test cases had been built for end-user testing and unit testing. Testing was not implemented from a data point of view.

Interviewee 2 pointed out that documentation has been out of shape and because of the silos it depends on the application domain how detailed the integration documentation is. This resulted in a scenario where it is hard to recognize application dependencies on each other and the same problem intensifies as Interviewee 4 commented that the testing pattern is too narrow to handle dependency testing. As the environment has been growing bigger over the years, Interviewee 2 saw that it has been harder and harder to collaborate with different external data source partners and internal data sources. Some cases were identified where the technology limited the integration possibilities as the older legacy applications were not designed to be integrated and it became impossible to integrate them. Interviewee 4

described the problem raised from the multivendor environment where it is not clear anymore who is responsible for integrations as the organization has only bought applications and the procurement process did not take integrations into concern. This resulted in procurements where integrations were not budgeted nor resourced. The organization's IT is too small to help the procurement teams with the problem and the business side does not know everything that has to be taken into concern when buying a new application. Interviewee 1 added that as the development work is outsourced the coordination work should be managed inside the Business Finland organization. The problem with internal coordination has been a lack of expertise and time. As there was not any strategy made for the work it resulted in leadership inside individual projects. Interviewee 2 felt that one major problem is Business Finland's network infrastructure which is very complex and problem-solving has been hard. Also, the network problems have been hard to identify.

4.3 Improvement Needs and Actions

During the years following the fusion certain needs have been raised to develop Business Finland as an organization and also some actions have been done already. Interviewee 1 described that Business Finland chose Azure as a public cloud environment and the positive experiences following it started questions inside Business Finland if Azure should be used for integrations in the future. Also, he stated that the future solution should be more maintainable and cost-effective. Interviewee 4 said that in the meanwhile Business Finland made some solutions as temporary fixes also Azure API Management was piloted, and some point-to-point integrations were done at the same time using modern technologies on Azure. The modern technologies have been enough, only a few surprises were related to some technical limitations such as file size. At that time the solutions were built based on a need and they should be transformed into centralized solutions and architecture that can be reused throughout Business Finland's projects. Interviewee 2 added related to the same topic that the future solutions must be scalable to all Business Finland's domains to renew the architecture and to solve the problems that have occurred. Interviewee 3 also said that the future solutions must be scalable, and the architecture needs to be designed also from an integration point-of-view. All of the interviewees saw a need for a clear integration strategy that would be followed which includes also a strategy for the existing integrations.

4.4 Migrating to iPaaS

During the interviews, it was found that Business Finland did not have a clear strategy to implement and maintain integrations. Lack of strategy lead to silos that then affected visibility in the organization. Many of the integrations were point-to-point even though for example ESB has been used for years. To resolve these problems, it became clear that firstly an integration strategy must be chosen, and it should give Business Finland a clear strategy that will be implemented on the projects. To choose this strategy it was identified that Business Finland has had positive experiences with Azure as a public cloud vendor and there have been already some projects that have utilized Azure in the project works. The future solution also had to be cost-effective and maintainable compared to the past. The solution should help with older applications as well and be suitable for Business Finland's complex network infrastructure.

From the literature review, we found out that iPaaS is a cost-efficient integration strategy that utilizes the cloud. It would also give Business Finland a centralized platform for future integrations and refactoring of older integrations and help with the visibility problem if all integrations in the future are built on the iPaaS platform. As the platform is in the cloud it would resolve the networking problem inside the cloud. As it is most logical to use Azure's iPaaS capabilities the strategy supports even more Business Finland's actions as the hybrid environment is already available and it would give connectivity to both environments and help Business Finland to utilize iPaaS at its best.

Because of the thesis length, it would be nearly impossible to follow the full implementation of iPaaS practices in an organization. Instead of implementing and following the implementation work in this thesis, we are focusing on the implementation of centralized API management.

This thesis endeavors to implement API management practices within the Business Finland organization, drawing upon insights gleaned from an extensive literature review. The findings from this review highlight the significance of adopting an iPaaS (Integration Platform as a Service) approach, leveraging the synergistic capabilities of cloud computing, Infrastructure as Code (IaC), and DevOps methodologies.

By harnessing the power of cloud computing, the proposed implementation aims to unlock a host of benefits for Business Finland. The iPaaS approach facilitates seamless integration, scalability, and flexibility, enabling the organization to effectively manage and orchestrate its APIs within a cloud-based environment. The cloud infrastructure offers a robust and reliable foundation, ensuring optimal performance, high availability, and the ability to handle varying workloads.

Furthermore, the utilization of Infrastructure as Code (IaC) practices enables the implementation to automate the provisioning and configuration of the necessary infrastructure components. This approach ensures consistency, repeatability, and version control, eliminating manual errors and discrepancies. Through IaC, the thesis aims to establish a solid foundation for API management practices, streamlining the deployment process and enhancing overall operational efficiency.

In addition, the adoption of DevOps methodologies is crucial in fostering collaboration and enabling continuous integration, delivery, and deployment. The integration of DevOps practices promotes effective communication and collaboration between development, operations, and other stakeholders, fostering a culture of agility, rapid iterations, and continuous improvement. By embracing DevOps principles, the implementation seeks to enhance the speed, quality, and reliability of the API management practices within the Business Finland organization.

The combined implementation of cloud computing, IaC, and DevOps practices, as informed by the literature review, offers a comprehensive framework for effective API management within Business Finland. This approach empowers the organization to leverage cutting-edge technologies and industry best practices, facilitating streamlined operations, enhanced scalability, and accelerated innovation.

4.5 Requirements for API management solution

The following requirements were constructed based on interviews, project requirements, literature review, and the current state of Business Finland’s Azure environment. During the literature review it was seen that iPaaS is the most modern integration strategy currently available so the API management solution should be from a provider that offers good iPaaS

capabilities. From API management literature review it was clear that API management supports organization with managing, publishing, and controlling APIs mainly. For this reason, we chose to praise such features that enable the main tasks of API management, such as the features to enforce security standards and provide API catalogue capabilities. RESTful, SOAP, and GraphQL was seen as the most used API protocols which was the reason to choose all three as a requirement. Hybrid infrastructure requirement was taken from the interviews to support the whole Business Finland organization and not just cloud capabilities.

Table 3. API management solution requirements.

ID	Requirement	Description
R1	The solution should support and enable the adoption of iPaaS (Integration Platform as a Service) practices.	It should be cloud-based and capable of incorporating various iPaaS components and practices within the same cloud environment. This ensures seamless integration and compatibility with other cloud-based services.
R2	The solution should enhance documentation practices by providing the ability to create and maintain an API catalog.	This catalog serves as a repository for API documentation, enabling better organization, discoverability, and accessibility of APIs for developers and consumers.
R3	The solution should improve API security standards.	It should offer features such as API proxying, allowing for the implementation of independent authentication and authorization mechanisms specific to the API. This enhances security by decoupling API security from the backend web API.
R4	The solution should be able to handle different levels of security in terms of data sensitivity.	It should provide mechanisms to classify and differentiate APIs based on the sensitivity of the data they handle. This ensures that appropriate security measures are applied to protect classified data.
R5	The solution should be able to handle cloud and on-premises environments.	It should be capable of managing APIs in a unified manner, regardless of whether they are deployed in the cloud or on-premises. This simplifies the solution architecture and allows for consistent management across

		different deployment environments.
R6	The solution should have logging capabilities to monitor the environment.	It should provide the functionality to log and track API calls, system performance, and other relevant metrics. This helps ensure the health and performance of the environment and facilitates the monitoring of key performance indicators (KPIs).
R7	The solution should have the ability to publish APIs in different architectural styles, including RESTful, SOAP, and GraphQL APIs.	As found out from the literature, these three architectural styles are commonly used for building APIs, and the solution should support all of them to accommodate various application and integration scenarios.

By addressing these requirements, the API management solution can effectively support the adoption of iPaaS practices, improve documentation practices, enhance API security, handle different levels of data sensitivity, manage diverse environments, provide logging capabilities, and enable the publication of APIs in multiple architectural styles.

4.6 Proposed solution

In order to propel Business Finland toward embracing future iPaaS (Integration Platform as a Service) practices, a comprehensive solution has been proposed. Central to this solution is the implementation of Azure API Management as a key technical component, offering robust support for Business Finland to leverage the API management lifecycle and engage relevant stakeholders, as outlined in Tables 1 and 2. By adopting Azure API Management, the solution empowers Business Finland with a powerful platform to efficiently manage and govern their APIs, streamlining the development, deployment, and consumption processes.

Furthermore, Azure API Management provides the added advantage of facilitating on-premises integration, thereby simplifying the overall architecture of the solution and creating a more consolidated environment. This capability ensures seamless connectivity and integration between on-premises systems and the cloud-based API management

infrastructure, enabling Business Finland to efficiently leverage its existing resources while harnessing the benefits of a scalable and flexible iPaaS solution.

To ensure the solution effectively meets the specific requirements of Business Finland, a meticulous mapping exercise has been undertaken, aligning each requirement with the corresponding capabilities offered by Azure API Management. This mapping process, as detailed in Table 4, serves as a comprehensive reference point, enabling a clear understanding of how the proposed solution satisfies the identified needs.

By implementing Azure API Management, Business Finland gains access to a robust set of features and functionalities that facilitate efficient API governance, security, monitoring, and scalability. The solution empowers Business Finland to effectively manage its APIs, streamline collaboration with stakeholders, and foster innovation through enhanced integration capabilities. With Azure API Management serving as a cornerstone of the proposed solution, Business Finland is well-equipped to embrace future iPaaS practices, enabling seamless integration and driving digital transformation initiatives forward.

Table 4. API management requirement mapping against Azure API Management capabilities.

ID	Azure API Management capability
R1 The solution should support and enable the adoption of iPaaS (Integration Platform as a Service) practices.	Azure offers a set of iPaaS solutions, and Azure API Management is one of them. It falls under the category of iPaaS (Integration Platform as a Service) along with other services such as Azure Service Bus, Azure Logic Apps, and Azure Functions. (Cestari et al. 2020)
R2 The solution should enhance documentation practices by providing the ability to create and maintain an API catalog.	Azure API Management provides an out-of-the-box developer portal. This portal automatically handles the API catalog based on the OpenAPI specification of the API and environment configuration. It simplifies the management and documentation of APIs for developers. (Microsoft, 2023a).
R3 The solution should improve API security standards.	Azure API Management can proxy existing APIs or new APIs and supports various authentication methods, including OAuth 2.0. This allows for secure access control and authorization of API requests. (Microsoft, 2023b).
R4 The solution should be able to handle different levels of security in terms of data sensitivity.	In addition to the discussed authentication methods, Azure API Management can publish APIs in private networks within Azure or on on-premises networks. This flexibility allows for the hosting and integration of APIs in different network environments. (Microsoft, 2023c and 2023d).

R5 The solution should be able to handle cloud and on-premises environments.	Azure API Management supports the option to host multiple gateways in different networks if necessary. This capability enables distributed deployments and better alignment with network architecture requirements. (Microsoft, 2023c).
R6 The solution should have logging capabilities to monitor the environment.	The environment of Azure API Management can be configured to log to various logging systems, such as Azure's own Log Analytics Workspace and Azure Application Insights. This allows for comprehensive monitoring, logging, and analysis of API calls and system behavior. (Microsoft, 2023d).
R7 The solution should have the ability to publish APIs in different architectural styles, including RESTful, SOAP, and GraphQL APIs.	Azure API Management has the capability to publish APIs in different formats, including SOAP, RESTful, and GraphQL APIs. This flexibility accommodates a wide range of API types and protocols. (Microsoft, 2023f).

4.6.1 Project implementation

The project was successfully executed in the year 2023. Throughout the implementation phase, although the developer portal remains unpublished, substantial progress has been made in establishing the technical infrastructure required for leveraging a centralized API management instance. The implementation itself encompasses three distinct environments, namely the development, test, and production environments. The overarching objective is to deploy these environments using Infrastructure as Code (IaC) and DevOps methodologies, ensuring their fundamental similarity while accommodating variations in specific parameters.

By adhering to IaC principles, the project team aims to automate and standardize the provisioning and configuration of the three environments. This approach guarantees consistency and eliminates discrepancies that may arise due to manual intervention or ad hoc setups. Through the utilization of IaC, the project endeavors to achieve a robust and reproducible environment provisioning process.

In parallel, the adoption of DevOps practices becomes pivotal in fostering collaboration and streamlining the development, testing, and deployment workflows. By embracing DevOps principles, the project team seeks to foster a culture of continuous integration, delivery, and deployment. This entails establishing efficient communication channels, employing version control systems, and automating various aspects of the software development lifecycle.

The underlying rationale behind implementing these environments with IaC and DevOps methodologies lies in their ability to enhance agility, scalability, and reliability. By treating infrastructure as code, modifications, and scaling can be easily orchestrated, providing greater flexibility in accommodating evolving requirements. Furthermore, the integration of DevOps practices facilitates seamless collaboration between development, testing, and operations teams, resulting in faster feedback loops, rapid iterations, and improved overall software quality.

Ultimately, the envisioned outcome of this project is the establishment of a robust and harmonized environment framework, with the development, test, and production environments aligned in structure and behavior while accommodating specific parameters unique to each environment. This unified approach not only simplifies the management and maintenance of the environments but also facilitates effective monitoring, troubleshooting, and scalability.

4.6.2 Solution architecture

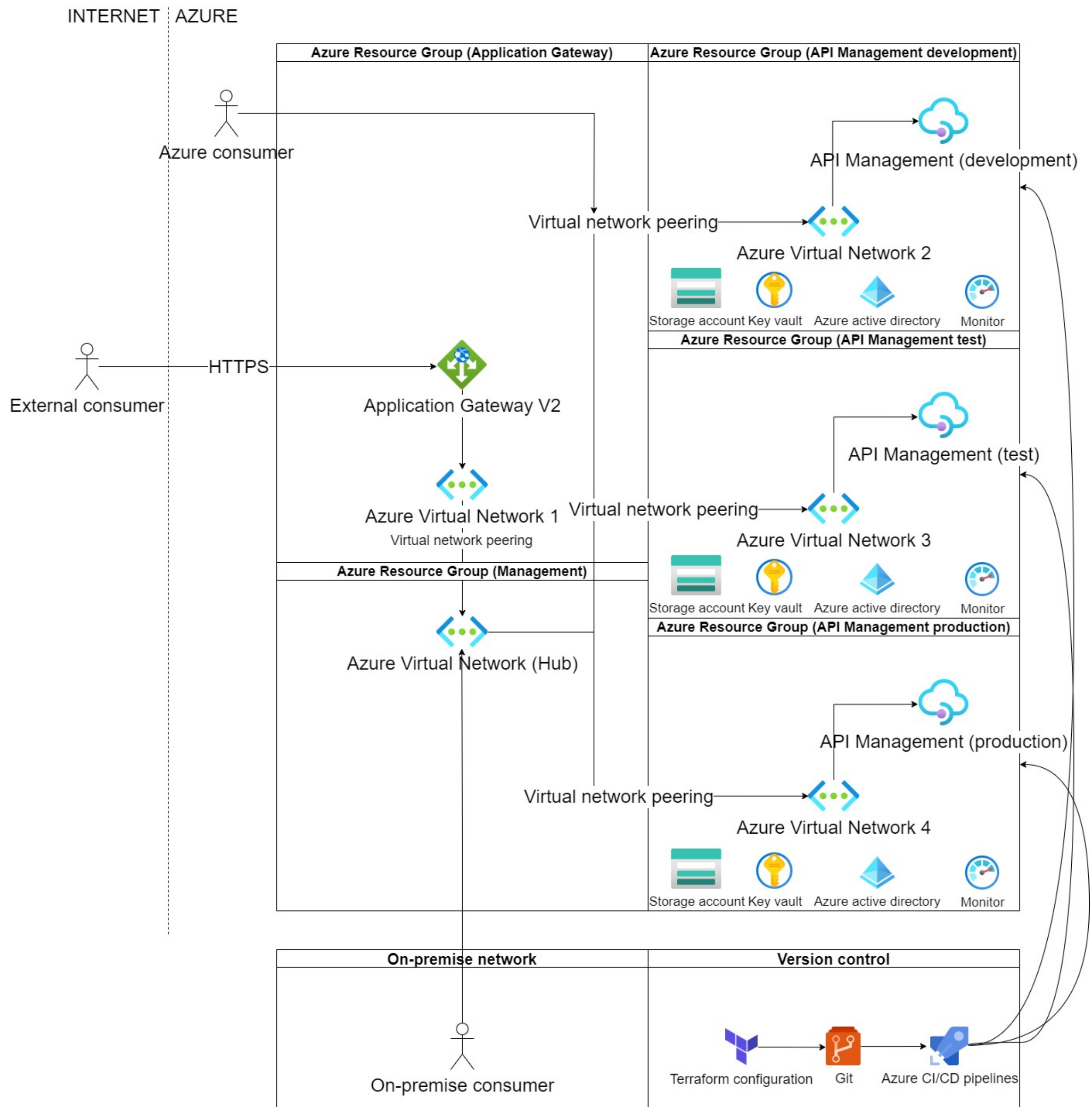

Figure 2. Solution architecture overview.

In accordance with the illustrative representation depicted in Figure 2, the solution architecture has been meticulously constructed to cater to the diverse needs of three distinct consumer categories: external, Azure, and on-premises. Commencing with external consumers, their connectivity is established via the internet employing the HTTPS protocol. This connection is directed towards the application gateway, which offers a connection probe to the API management instance through peering facilitated by the hub network. Subsequently, Azure consumers possess the ability to directly connect through the Azure

virtual network, leveraging peering capabilities as well. It is important to note that the dissimilarity between external and Azure consumers lies in the fact that the connection for Azure consumers remains exclusively within the Azure network, effectively treated as internal traffic. This form of connection is particularly advantageous when sensitive or classified data is transmitted within the confines of the Azure environment, eliminating any exposure to external networks. On the other hand, the on-premises consumer segment connects to the solution architecture through a VPN gateway, which subsequently lands on the hub network and establishes peering with the destination API management network. It is worth mentioning that an alternative deployment approach involving a dedicated gateway instance exclusively for on-premises connectivity could have been pursued. However, during the course of this project, such an addition was deemed unnecessary as the existing infrastructure successfully facilitated internal connectivity from on-premises to Azure.

Within the Azure infrastructure, several crucial components have been meticulously integrated to ensure optimal functionality. Firstly, each environment is equipped with its own dedicated API management instance, individual network configuration, and a designated storage account responsible for hosting crucial elements such as the Terraform state file. Furthermore, key vaults have been incorporated to effectively manage and secure secrets and certificates, while maintaining seamless integration with Azure Active Directory for comprehensive access control implementation and identity management specific to the API management instance. To ensure effective monitoring and observability, the architecture encompasses monitoring components including Azure Log Analytics Workspace, which facilitates comprehensive logging capabilities, and Azure Application Insights, which monitors the HTTPS calls and ensures robust performance monitoring.

By meticulously integrating these diverse components and adopting a comprehensive architectural approach, the solution ensures a robust and secure environment for each consumer category while facilitating efficient management, monitoring, and scalability.

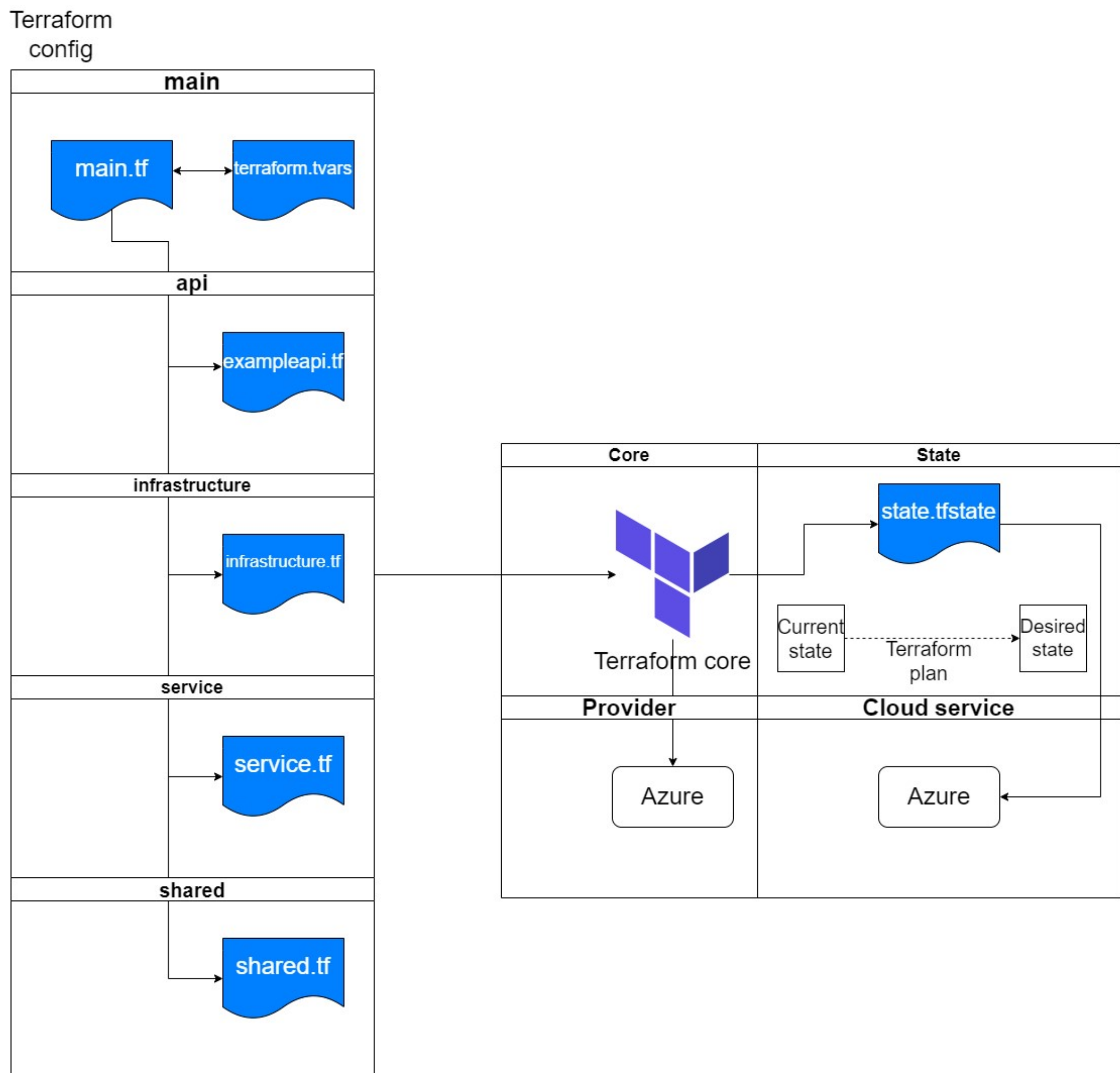

Figure 3. Terraform IaC architecture overview.

The implemented solution was meticulously constructed utilizing the power of Terraform Infrastructure as Code (IaC) to harness the numerous benefits highlighted in section 2.7. The entire solution is structured within the main.tf configuration file, which effectively serves as the core orchestrator, leveraging Terraform variables from the terraform.tvars file to facilitate the deployment of multiple instances encompassing development, test, and production environments. To ensure modularity and maintainability, each API within the solution possesses its own dedicated Terraform configuration file, specifically designed to deploy and configure the unique aspects of the respective API.

To configure the API behaviour and settings, the Terraform configuration file is seamlessly linked to an OpenAPI specification. This specification plays a pivotal role in defining crucial

elements such as the API path, comprehensive API documentation, and protocol settings, enabling efficient and accurate API configuration. Additionally, certain configurations, such as the API ID on the platform and gateway configurations, including vital components like authentication settings, backend authentication credentials, and error handling logic, are neatly organized within an XML file. This XML file serves as a repository for policy configuration that the gateway utilizes to enforce the desired behaviour.

The infrastructure configuration encapsulates all the necessary specifications for the supporting infrastructure components, as depicted in Figure 2. This encompasses essential elements such as the storage account, key vault, log analytics workspace, and application insights. These components work synergistically to ensure seamless operations, comprehensive security, and efficient monitoring and logging functionalities throughout the solution.

On the other hand, the service configuration pertains specifically to the configuration of the API management instance. This dedicated configuration file encompasses crucial aspects such as administrative settings, instance names, network settings, custom domain configurations, and the chosen pricing tier. A detailed rationale behind the selection of specific pricing tiers and accompanying justifications can be found in Table 5, providing transparency and insight into the decision-making process.

By employing Terraform IaC and adopting a modularized approach, the solution achieves a high degree of automation, consistency, and scalability. The ability to define infrastructure and configuration settings in code ensures reproducibility, simplifies maintenance and facilitates rapid deployments. Moreover, the clear separation of concerns within different configuration files enhances clarity and ease of management, enabling efficient customization and flexibility as per the requirements of each environment and API.

Table 5. API management pricing tiers.

Environment	Pricing tier	Reasoning
Development	Developer	The developer tier is suitable for non-production environments and offers an isolated instance compared to the consumption pay-as-you-go model. In the development environment, the primary requirement is virtual network integration support, which allows for

		both internal and external usage. This tier also provides the capability to publish on-premises APIs through a VPN gateway, facilitating connectivity between on-premises systems and the API management platform.
Test	Developer	Similar to the development environment, the test environment also benefits from the developer tier. It provides all the necessary features required in the production environment, making it suitable for testing purposes. However, it is important to note that as the load and usage increase in the future, it may be necessary to consider scaling up the pricing tier accordingly to ensure optimal performance.
Production	Premium	For the production environment, the premium tier is chosen as it offers the necessary features and capabilities required for a production-level API management instance. The premium tier allows for integration into the virtual network, which aligns with the architecture requirements. At the time of writing, there were no other available options that specifically supported the determined solution architecture, making the premium tier the most suitable choice for production usage.

By selecting the appropriate pricing tiers for each environment, the solution ensures the required functionality, scalability, and network integration, while also considering the cost-effectiveness of the chosen tiers. It is important to regularly review and evaluate the chosen pricing tiers based on the evolving needs and usage patterns of the solution to ensure optimal performance and resource allocation.

4.7 Discussion

In the current state, the solution has been working as we planned, and the chosen setup seems to be the right one. Some changes are planned to be implemented in the future, e.g., currently, the solution uses on-demand agents for deployment that caused issues with firewall openings when the pipeline accesses the Terraform state file. This problem is going to be solved by buying an independent server for API management's usage. The server will have open connections to the Azure storage account that hosts the state file. Currently, the on-demand agents have several hundred different IP addresses, so it is not viable to open the connections to all of the IP addresses. We fetch the current agent's IP address, and the automation allows the IP address on the firewall it causes issues with delays and Microsoft has not yet fixed an issue that happens when the agent and the destination are hosted on the same region. In this scenario, the on-demand agent uses a private IP that does not work with the firewall. All other regions use public IP which works better in this case.

Other vendors provide the same kind of features, such as Boomi and IICS described in Chapter 2.3. These solutions would have most likely worked as well but it would have meant that Business Finland would have needed to invest in another cloud service besides Azure as there are already existing services and some iPaaS capabilities tested which was found out in interviews, chapter 3.3. If Business Finland had gone for these services, it would have complicated the architecture as a cloud-to-cloud connection would be needed and it is unknown how these solutions would have offered private networking solutions between the consumer on Azure and API management solutions on another cloud. The solution used already existing infrastructure such as a VPN gateway to on-premises which would mean that the separate provider for API management also needs infrastructure that raises implementation costs. It was also said in Chapter 2.3. that Microsoft Azure causes vendor locks for consumers, and it is fair to say that it is true. However, the vendor lock also gives benefits in many ways like building an internal way to connect APIs to provide the best security for confidential data.

As almost every organization handles confidential data in 2023 it was proven with the chosen solution architecture that a single API management system can handle the majority of the use cases that are needed. The Business Finland solution does not utilize on-premises API gateway yet but it is possible to deploy it later if needed to route the data through on-premises

without a cloud gateway between. Other on-premises APIs can be deployed on the cloud gateway along with cloud APIs. This gives a good foundation when adapting to iPaaS architecture on full scale later.

In the future, it would be best to analyze the use of the iPaaS tools in a migration project and follow how the integration strategy can be adopted in an organization. API management gives the tools to utilize APIs in the best way possible which then gives the possibility to utilize iPaaS throughout the organization's APIs. This solution does not solve the underlying issues with integration strategies, but it was needed, and it helps to adapt to iPaaS in 2024. API management is a crucial component of iPaaS but the integration strategy cannot be resolved with technical components, the strategy is an organizational culture that has to be changed and that is something that will be started after this research.

For the issues that the previous ESB caused, we already solved many ownerships and documentation-based issues. Previously, we did not understand who owns the integrations and how these are documented. Because of API management capability, we have an owner for each API who is responsible for its development. Each API is documented because technically each publishment is made by using API specification. All the new integrations are now centralized through the API management, and we have multipurpose APIs that saved time and cost to develop new solutions. However as said, we do not have had any scenarios where other iPaaS components would have been needed. Currently our applications can integrate through API management, and we can manage the integrations that way. It is certain that in the future such cases will come and in those cases the solution architecture must be reviewed once again.

5 Conclusions

Throughout the course of our research, we unearthed the underlying causes of Business Finland's ambiguous integration strategy. The primary contributing factor to this predicament was the merger between Tekes and Finpro, two organizations each with distinct purposes that were not harmonized during the consolidation. Rather than embarking on a comprehensive integration effort, the existing architectural elements were simply transferred to Business Finland, perpetuating their separate, pre-merger functionality. Our interviews with key stakeholders also revealed that while some ad-hoc cloud-based integrations were utilized, they had not been institutionalized as part of a broader organizational strategy. It became evident that Business Finland must adopt Integration Platform as a Service (iPaaS) as the linchpin of its integration strategy moving forward, with the imperative need to document this strategy comprehensively to facilitate the realization of strategic iPaaS goals.

Significantly, the organization's infrastructure underwent a noteworthy transformation during the research, shifting towards a hybrid model that straddles the Azure cloud and on-premises environments. The API management solution that was constructed featured a VPN gateway to bridge the legacy on-premises systems and the burgeoning cloud infrastructure. This architectural decision was deliberate, and designed to pave the way for Business Finland to harness iPaaS in the future. With the VPN gateway in place, on-premises APIs can be securely shared through cloud-based API Management, obviating the need to recreate connectivity whenever a cloud service seeks to interact with the on-premises components.

As part of our interviews, it became evident that the experiences with Azure Cloud were largely positive, and select Azure iPaaS capabilities had been informally tested without a formal strategy. Consequently, it was decided that Business Finland should leverage these capabilities in the future, laying the foundation for API Management as the organization's inaugural, overarching iPaaS capability. This decision was reinforced by the fact that Azure had already been integrated into the organization and offered a robust infrastructure architecture conducive to building a hybrid API Management solution in the cloud while maximizing the utility of the existing infrastructure architecture.

Following the completion of this research, the organization stands on the threshold of a significant undertaking to embrace iPaaS as a unified strategy at the organizational level.

Initially, a structured approach is required to disseminate information to various projects already possessing a plethora of APIs. This ensures that they are equipped to publish their APIs through the centralized API Management service. Furthermore, the development of a comprehensive strategy is imperative to guide the integration landscape within Business Finland. This strategy will encompass the selection of iPaaS capabilities suitable for incorporation within Business Finland, outline the preferred architecture for cloud-based integrations, and delineate the prescribed utilization of the centralized Data Platform in the context of data integrations.

References

Achar, Sandesh. (2021). Enterprise SaaS Workloads on New-Generation Infrastructure-as-Code (IaC) on Multi-Cloud Platforms. Global Disclosure of Economics and Business. 10. 55-74. 10.18034/gdeb.v10i2.652.

Andersson, N. (2002). Design and Implementation of an Application Programming Interface for Volume Rendering. (Independent thesis, Linköping University, Department of Science and Technology).

Belqasmi, F., Singh, J., Bani Melhem, S. Y., & Glitho, R. H. (2012). SOAP-Based vs. RESTful Web Services: A Case Study for Multimedia Conferencing. IEEE Internet Computing, 16(4), 54-63. doi: 10.1109/MIC.2012.62.

Biehl, M. (2016). RESTful API Design, API-University Series. CreateSpace Independent Publishing Platform. ISBN: 9781514735169.

Bondel, G., Landgraf, A., & Matthes, F. (2022). API Management Patterns for Public, Partner, and Group Web API Initiatives with a Focus on Collaboration. In EuroPLoP'21: Proceedings of the 26th European Conference on Pattern Languages of Programs (Article 34, pp. 1-17). New York, NY: Association for Computing Machinery. doi:10.1145/3489449.3490012.

CITO Research. (2015). Cloud-based API Management: Harnessing the Power of APIs. Retrieved from https://j.mp/ms-apim-whitepaper

Erasala, N., Yen, D. C., & Rajkumar, T. M. (2003). Enterprise Application Integration in the electronic commerce world. Computer Standards & Interfaces, 25(2), 69-82. ISSN 0920-5489. https://doi.org/10.1016/S0920-5489(02)00106-X.

Gulledge, T. 2006. What is integration?. Industrial Management and Data Systems. 106. 5-20. 10.1108/02635570610640979.

Hartig, O. and Pérez, J., 2018. Semantics and Complexity of GraphQL. In Proceedings of the 2018 World Wide Web Conference (WWW '18). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 1155–1164. https://doi.org/10.1145/3178876.3186014.

Huth, A. G., & Cebula, J. (2011). "The Basics of Cloud Computing." Retrieved from https://api.semanticscholar.org/CorpusID:61962240.

Hyrynsalmi, S., Koskinen, K., Rossi, M. & Smolander, K. (2021). Towards the utilization of Cloud-based Integration Platforms. 1-8. 10.1109/ICE/ITMC52061.2021.9570235.

Hyrynsalmi, S. (2022). Pathway to the successful integration platform management. In: CEUR Workshop Proceedings. ICSOB '22: 13th International Conference on Software Business, November 8–11, 2022, Bolzano, Italy. https://ceur-ws.org/Vol-3316/phd-paper5.pdf.

Katayama, T., Nakao, M., Takagi, T. (2010). TogoWS: integrated SOAP and REST APIs for interoperable bioinformatics Web services. Nucleic Acids Research, 38(suppl_2), W706-W711. doi: 10.1093/nar/gkq386.

Matei, M. (2012). iPaaS: Different Ways of Thinking. Procedia Economics and Finance, 3, 1093-1098. ISSN 2212-5671. https://doi.org/10.1016/S2212-5671(12)00279-1.

Mathijssen, M., Overeem, M., & Jansen, S. (2017). Identification of Practices and Capabilities in API Management: A Systematic Literature Review. Journal of Systems and Software, 123, 221-237. doi:10.1016/j.jss.2016.11.043.

McKeen, J., & Smith, H. (2002). New Developments in Practice II: Enterprise Application Integration. Communications of the Association for Information Systems, 8, pp-pp. https://doi.org/10.17705/1CAIS.00831.

Menge, F. (2007). Enterprise Service Bus. Retrieved from https://programm.froscon.org/2007/attachments/15-falko_menge_-_enterpise_service_bus.pdf.

Microsoft. (2023a). Import API from OpenAPI Specification (OAS). Retrieved from https://learn.microsoft.com/en-us/azure/api-management/import-api-from-oas

Microsoft. (2023b). API Management - How to protect a backend with Azure Active Directory. Retrieved from https://learn.microsoft.com/en-us/azure/api-management/api-management-howto-protect-backend-with-aad

Microsoft. (2023c). Self-hosted gateway overview. Retrieved from https://learn.microsoft.com/en-us/azure/api-management/self-hosted-gateway-overview

Microsoft. (2023d). API Management using with internal VNET. Retrieved from https://learn.microsoft.com/en-us/azure/api-management/api-management-using-with-internal-vnet

Microsoft. (2023e). Observability in Azure API Management. Retrieved from https://learn.microsoft.com/en-us/azure/api-management/observability

Microsoft. (2023f). Add an API to an Azure API Management instance manually. Retrieved from https://learn.microsoft.com/en-us/azure/api-management/add-api-manually

Mulligan, G., & Gračanin, D. (2009). A comparison of SOAP and REST implementations of a service based interaction independence middleware framework. Proceedings of the 2009 Winter Simulation Conference (WSC), 1423-1432. doi: 10.1109/WSC.2009.5429290.

Ong, S. P., Cholia, S., Jain, A., Brafman, M., Gunter, D., Ceder, G., & Persson, K. A. (2015). The Materials Application Programming Interface (API): A simple, flexible and efficient API for materials data based on REpresentational State Transfer (REST) principles. Computational Materials Science, 97, 209-215. https://doi.org/10.1016/j.commatsci.2014.10.037.

Ranga, Virender & Soni, Anshu. (2019). API Features Individualizing of Web Services: REST and SOAP. International Journal of Innovative Technology and Exploring Engineering. 8. 10.35940/ijitee.I1107.0789S19.

R. H. Cestari, S. Ducos & E. Exposito. (2020), "iPaaS in Agriculture 4.0: An Industrial Case," 2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Bayonne, France, 2020, pp. 48-53, doi: 10.1109/WETICE49692.2020.00018.

Risimic, D. 2007. An integration strategy for large enterprises. Yugoslav Journal of Operations Research. 17. 10.2298/YJOR0702209R.

Schmidt, M. T., Hutchison, B., Lambros, P., & Phippen, R. (2005). The Enterprise Service Bus: Making service-oriented architecture real. IBM Systems Journal, 44(4), 781-797. doi: 10.1147/sj.444.0781.

Soomro, Tariq & Awan, Abrar. (2012). Challenges and Future of Enterprise Application Integration. International Journal of Computer Applications. 42. 975-8887. 10.5120/5708-7762.

Stünkel, P., von Bargen, O., Rutle, A., & Lamo, Y. (2020). "GraphQL Federation: A Model-Based Approach." The Journal of Object Technology, 19(2). https://doi.org/10.5381/jot.2020.19.2.a18.

Appendix 1. Interview Questions

- Can you share insights into how the company approached system and data integration in the past? What were the primary goals, and what outcomes were expected?

- What challenges arose during previous system and data integration initiatives, and how were they overcome? What were the overall results of addressing these challenges?

- From your perspective, where did past system and data integration strategies fall short? Can you provide specific examples and elaborate on their impact on the organization?

- Reflecting on past experiences, what improvements or modifications do you think could have enhanced the system and data integration process? Why do you believe these changes would have been beneficial?

- How crucial is data quality and consistency in the success of system and data integration? Have there been instances where data quality issues hindered integration efforts in the past? How could those situations have been handled more effectively?

- Integrating legacy systems with modern technology platforms can pose unique challenges. What strategies were previously employed to address these challenges? How successful were these strategies, and what alternative approaches could have been considered?

- Considering data security and privacy, were there specific challenges or concerns encountered during previous integration efforts? How were these issues handled, and what insights were gained from those experiences?

- Collaboration across different teams or departments is often necessary for successful integration. Were there any coordination or collaboration challenges in previous projects? How were these challenges mitigated, and what steps could have improved collaboration?

- Technology plays a vital role in integration. What platforms, tools, or solutions were used in the past for integration purposes? Were there any limitations or issues with these technologies, and how could technology be better utilized to improve integration outcomes?

- Looking ahead, what emerging technologies or trends do you think will impact system and data integration strategies? How can organizations proactively incorporate these trends into their integration processes?