Received: 28 January 2023 | Revised: 17 June 2023 | Accepted: 27 June 2023

DOI: 10.1002/spe.3241

RESEARCH ARTICLE

WILEY

Microservice reference architecture design: A multi-case study

Mehmet Söylemez[1] | **Bedir Tekinerdogan**[2] | **Ayça Kolukısa Tarhan**[1]

[1]Hacettepe University, Computer Engineering Department, Ankara, Turkey

[2]Wageningen University & Research, Information Technology, Wageningen, The Netherlands

Correspondence
Bedir Tekinerdogan, Chair Information Technology Group, Wageningen University & Research, P.O. Box 8130, 6700 EW, Wageningen, The Netherlands.
Email: bedir.tekinerdogan@wur.nl

Abstract

Microservice architecture (MSA) is an architectural style that is designed to support the modular development of software systems within a particular domain. It is characterized by the use of small, independently deployable services, which can be developed and deployed autonomously. The benefits of MSA include improved scalability, fault-tolerance, and ease of deployment and maintenance. However, developing MSA for a specific domain can be challenging and requires a thorough consideration of various concerns such as service boundaries, communication protocols, and security. To support the easy development and guidance of an MSA for practitioners, we present a reference architecture design for MSA. The reference architecture has been designed after a comprehensive domain analysis of MSA in the literature and the MSAs of key vendors. This analysis has been used to identify best practices and common patterns that can be applied to the development of MSA. Additionally, relevant architecture viewpoints have been selected to model the corresponding architecture views, providing a clear and comprehensive understanding of the architecture. To validate the proposed reference architecture, a multi-case study research approach has been used. In this approach, two industrial case studies have been used to demonstrate the practical applicability of the proposed reference architecture. The results of these case studies have shown that the reference architecture can be used to effectively guide the development of MSA in real-world scenarios.

KEYWORDS

case study research, microservice architecture, reference architecture

1 | INTRODUCTION

Software systems are continuously evolving to meet the increasing demands of modern applications. Monolithic architecture, a prevalent approach in software development, tightly integrates all components and their associated functionalities within a single codebase.[1] While this approach may work well for small systems, it becomes increasingly cumbersome

Abbreviations: API, Application Program Interface; AWS, Amazon Web Services; CI&CD, Continuous Integration & Continuous Delivery; MSA, Micro Service Architecture; SOA, Service-Oriented Architecture.

for larger, more complex applications. This is because all components are tightly coupled and dependent on each other, making it difficult to make changes or updates without affecting the entire system. Additionally, monolithic architecture can experience significant maintenance and deployment problems, as all components must be present for the code to be executed or compiled.

To address these limitations, Microservice architecture (MSA) has emerged as an alternative architectural style. MSA enables the modular development of loosely coupled, highly cohesive services. It promotes autonomy, allowing for continuous delivery, independent deployability, and improved scalability.[2-5] By breaking down applications into smaller, decoupled services, MSA offers numerous advantages, including the freedom to choose the most suitable technologies and programming languages.[6] Service-oriented architecture (SOA) is another development-level variant of MSA that adheres to the principle of separating concerns and relies on clearly defined boundaries between services. However, SOA relies on heavy-weight middleware such as enterprise service busses, which can undermine the autonomy of services.

While MSA offers many benefits, practitioners still face many challenges such as orchestration of microservices, defining optimal boundaries of microservices, ensuring data consistency and distributed transaction management, versioning, and distributed tracing.[2,7,8] To address these challenges, this study proposes a reference architecture for microservices and a method for deriving an application architecture from it.

The motivation behind this study is to design a robust reference architecture that captures the best practices and experiences from both industry and academia. To achieve this, we conducted a comprehensive domain analysis of MSA in the literature and studied the microservice architectures implemented by key vendors. By examining the experiences and lessons learned from major cloud vendors, who have successfully deployed microservices at scale, we gain valuable insights into the practical challenges and effective solutions encountered in real-world implementations. Major cloud vendors have been at the forefront of adopting and implementing microservice architectures at scale. They have invested significant resources in research and development, exploring various architectural styles and paradigms to build scalable, resilient, and secure distributed systems. While these vendors may not specialize exclusively in microservices, their experience in developing and deploying large-scale distributed systems provides valuable insights for designing an effective microservice reference architecture. By examining the practices and approaches employed by major cloud vendors, we can gain insights into the practical challenges and solutions encountered during the implementation of microservice architectures. This analysis helps inform the design of our reference architecture, ensuring that it incorporates industry best practices and real-world considerations.

The reference architecture design process employed a domain-driven architecture approach, utilizing feature diagrams to create a domain model encompassing the common and variant features of MSA. Architectural viewpoints were utilized to ensure a holistic and comprehensive design. To validate the effectiveness and feasibility of the proposed reference architecture, a multi-case study research approach was conducted.

The contributions of this study include a domain-driven architecture design approach that can be utilized to design a microservice reference architecture, a reference architecture designed for deriving microservice application architectures, and the validation of both the approach and reference architecture through a multi-case study research. Overall, this study presents a comprehensive approach to designing and implementing microservice architecture that addresses the challenges and concerns associated with this architectural style. The proposed reference architecture and methodology provide a solid foundation for practitioners to build and deploy robust, scalable, and maintainable microservice-based applications. The multi-case study research provides valuable insights into the feasibility and effectiveness of the proposed approach and reference architecture, making it an important reference for researchers and practitioners in the field of microservices.

The rest of this article is organized as follows: Section 2 presents the background, Section 3 describes the research methodology and development method of microservice reference architecture, Section 4 explains the results of multi-case study, Section 5 presents the related work, Section 6 discusses the results of our research by addressing threats to validity, and finally, Section 7 concludes the article.

2 | BACKGROUND

2.1 | Microservice architecture

MSA was first described by Lewis and Fowler in a famous article.[4] Since then, it has gained a lot of attention in both academia and industry. Lewis and Fowler defined MSA as "an approach for developing a single application as a suite

of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API." However, this is not the only definition of MSA. For example, Newman defines microservices as εsmall autonomous services that work together, modeled around business domaine.[3] Despite the different definitions, all of them emphasize the small and autonomous nature of microservices.

One of the main reasons why MSA has become so popular is because it accelerates the software development process. This is due to the autonomous nature of microservices. The autonomy of each service emerges from the decomposition of the complex domain into smaller subdomains and components, which can be developed, deployed, versioned, tested, and managed independently.[9] With MSA, it is possible to have highly cohesive and loosely-coupled services by applying the separation of concern principle. To achieve this, it is important to determine the boundaries of each microservice. Microservices should be organized around business capabilities, which simplifies identifying service boundaries. The most famous approach for identifying boundaries is business-driven development. This approach helps identify bounded contexts and decompose a domain into subdomains.[10-12]

MSA also provides great benefits in adapting to new requirements and change management. In a monolithic architecture, changes require the whole system to be rebuilt and completely redeployed, while with MSA, only the affected services need to be rebuilt and deployed independently. Microservices are highly maintainable and testable,[11] which gives great agility to a software system. It is easier to change the business direction according to customer needs and it also allows practitioners to select the most appropriate technology for the customer's needs. This eliminates the long-term commitment to a specific technology stack since each microservice can be implemented in a different technology stack that best fits its functional and nonfunctional requirements.

Another advantage of MSA is the ability to provide an infrastructure where microservices, as the units of scaling, can be scaled independently according to their requirements and the interests they face.[6] Scalability is not only about adapting to a growing amount of work by adding resources, but also about operating the system efficiently while preserving the quality.[13,14] Additionally, practitioners can make decisions about each service independently. In other words, different scaling policies can be applied to microservices depending on their runtime metrics and states. All these abilities make the applications highly scalable and available.

The fact that MSA includes loosely-coupled services allows the system to be more fault tolerant.[15] Generally, specific and relevant services are affected by a failure and only those services need to be rebuilt and deployed. This prevents the whole system from being unavailable and makes the architecture more reliable against any failure.

Systems are always open to changes as scenarios evolve and requirements change. This is an expected behavior in software development process. However, managing these changes better has become much more sustainable and applicable with MSA. Since every microservice is a small business process and represents a small aspect of business functionality, it is easy to adapt to new changes.[16] However, all these conveniences are the outcome of an evolutionary process. This process starts with determining the boundaries of microservices and shaping them around the business capability and continues with the creation of DevOps practices and the evolution of the organization accordingly. The next step is to have an elastic infrastructure and automate that infrastructure to the possible extent by creating continuous integration & continuous delivery (CI&CD) processes. With the automated infrastructure, many advanced deployment techniques can be used, and projects become ready for using MSA.[9]

Despite the advantages listed above, it is still difficult for software teams to implement MSA in distributed projects, and for practitioners to guide teams to successful MSA adoptions.[10,11,13-15,17] The notion of MSA is complicated in terms of distributed service, identification, management, and maintenance, which is one of the key reasons for its complexity. As a result, successful MSA adoption necessitates a thorough awareness of the issues and potential solutions.

2.2 | Architecture views

Software architecture is a fundamental aspect of the software development process that involves the design and organization of the components and relationships within a computing system. The purpose of software architecture is to provide a blueprint for the system that addresses the concerns of stakeholders and ensures that the system is maintainable, reliable, and adaptable to changing requirements. It is a critical tool for identifying and mitigating potential issues during the development of a system, as well as for guiding the overall development process.

The process of software architecture begins with identifying the stakeholders of the system, which can include individuals, teams, or organizations that play a significant role in determining the system's requirements. These

stakeholders have different perspectives and concerns, and it is essential to consider these when designing the architecture of the system. To address these concerns, software architecture is typically defined using different architectural views.[12,18]

An architectural view is a representation of the system that describes the components and their relationships from a particular perspective. This allows for the creation of views that align closely with the concerns of stakeholders, resulting in an architecture that appeals to all of them. Additionally, each stakeholder can define or analyze the architecture using the views that pertain to their interests.

There are various approaches to documenting software architectures, with the most recent being the views and beyond approach (V&B). The V&B approach involves dividing views into four main styles: module, component and connector, allocation, and hybrid which provides a combination of styles. Each style addresses the architecture from a different perspective and addresses different concerns. The module style focuses on implementation details, the component and connector style deals with the interactions between software components, and the allocation style addresses the allocation of software components.

In conclusion, software architecture is an essential part of the software development process that plays a critical role in the design and organization of a computing system. It addresses the concerns of stakeholders and ensures that the system is maintainable, reliable, and adaptable to changing requirements. With the right approach, software architecture can be effectively documented and represented, making it more reusable and easy to maintain.

3 | MICROSERVICE REFERENCE ARCHITECTURE

3.1 | Method to develop reference architecture

As stated before, software architecture is a critical aspect of software development, as it involves defining the structures and relationships within a computing system. This includes identifying and preventing potential issues that may arise during the development process and making the system more maintainable and reliable. The decisions made during the architecture phase can greatly impact the entire development process of the system.

The decision for developing a reference architecture follows the steps as shown in as depicted in Figure 1.[19] The reference architecture serves as a foundation for creating an application architecture. If a practitioner cannot find a module

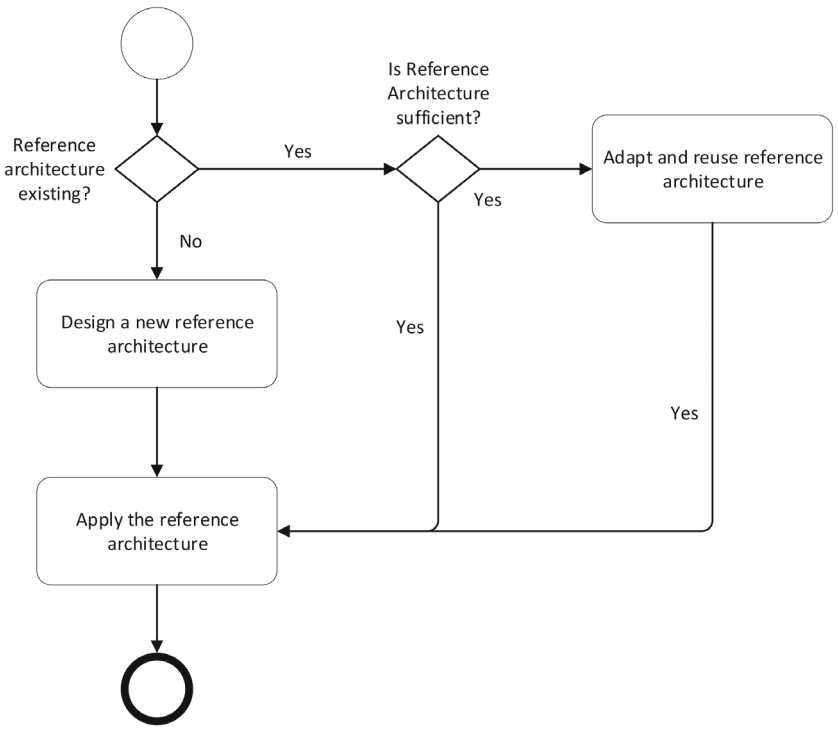


FIGURE 1 Three scenarios of using reference architecture to derive an application architecture

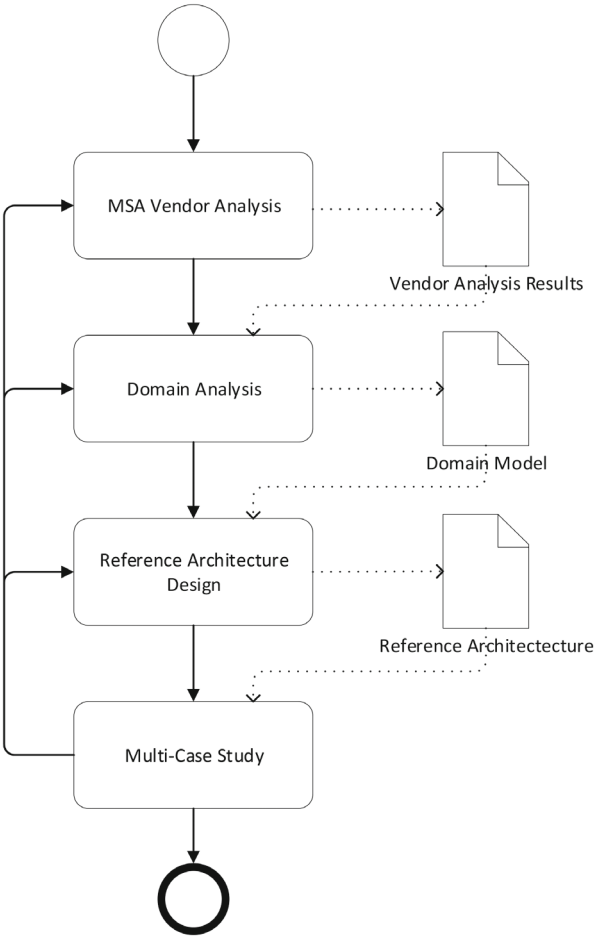


FIGURE 2 Development methodology of microservice reference architecture

that addresses a specific feature they are concerned about, they can add a new module to the reference architecture. On the other hand, if a suitable module already exists, practitioners can either use it as is or modify it to meet their specific needs through decomposition or combination with other modules, among other methods. Ultimately, this results in the development of an application architecture.

One specific area of software architecture is the development of reference architectures for microservices. In this section, we outline the process we followed to develop a reference architecture for microservices. The process, as illustrated in Figure 2, begins with analyzing the components, services, and architectures of MSA offered by major cloud providers such as Amazon, Google, and Microsoft. This includes examining challenges, opportunities, emerging technologies, and new trends in MSA. Despite the growing popularity of MSA, there are still limited resources available that fully address all aspects of modeling MSA.[20,21]

Following the MSA vendor analysis, we performed a domain analysis.[16] This activity aims to store domain knowledge for the engineering team to use in architecture development. Domain analysis comprises two primary activities: domain scoping and domain modeling. Domain scoping involves identifying the information sources and domains, while domain modeling represents domain knowledge in a reusable format. One methodology for domain modeling that can be used is feature modeling.[22] The domain model is represented using feature models that can show common and variable features of a product or system, and the dependencies among variable features. A feature diagram has four basic feature types: (1) mandatory features which are so-called must have/must include, (2) optional features which can have/or not components, (3) alternative features (XOR) in which case it must include one of the possible components, and (4) and/or features that at least one component should be included in.

After the domain analysis, we designed the reference architecture of MSA using viewpoints. The architecture design can be represented using architecture design viewpoints. Several architectural viewpoints are defined to address

different stakeholder concerns. In this study, we adopted the layered view, decomposition view, and deployment & service-oriented architecture view. The reference architecture should meet certain characteristics to be beneficial, including being understandable to all stakeholders, accessible and read/seen by most of the company, handling significant domain concerns, having satisfactory quality, being current, and maintained, and offering value to the business.

Finally, we conducted a multi-case study to evaluate the artifacts of the reference architecture. The feedback from the case study can be used to enhance the reference architecture. Two case studies were conducted to apply our reference architecture, and our recommendations, lessons-learned, and evaluations are shared in Section 5.

3.2 | Analysis MSA vendors

In this section, we provide an analysis of the services offered by the major cloud providers, Amazon AWS, Google Cloud Platform, and Microsoft Azure. We have provided an overview of the solutions they offer for various functions in Table 1. As depicted in Table 1, the cloud providers offer a wide range of services for different features. These services are provided by the cloud provider and can be utilized by making the necessary configurations. The services are also designed to be compatible with one another and can be seamlessly integrated to work together, resulting in increased efficiency. Additionally, it is noteworthy that all three providers are continually developing their services in parallel, with each provider offering similar services. For a more thorough evaluation of the services, including performance, availability, use cases, and cost analysis, one can conduct a detailed analysis to determine the best fit for their specific needs.

3.3 | Domain analysis

Family feature model is used to store domain knowledge to use in architecture development. In our previous work, we have conducted a feature-driven analysis of MSA.[16] We have identified 10 key features along with sub-features. Table 2 shows the key features extracted from our feature-driven domain analysis study.

3.4 | Reference architecture

A reference architecture is a comprehensive representation of the architecture of a system or a collection of systems, which is used as a guide for developing application architecture. It captures the key principles, elements, and relationships of the architecture, and is designed to ensure that the architecture is maintainable, scalable, and aligned with the needs and requirements of all stakeholders. After the family feature model with domain analysis has been developed, the next step is to create the reference architecture. The reference architecture is built on the foundation of domain knowledge, which is acquired through domain engineering activities. To increase its usability, the reference architecture is documented with various viewpoints that address the concerns of all stakeholders. To achieve this, we have chosen the V&B framework, developed by Clements et al.,[18] which is a well-established and widely recognized method for documenting software architecture.

The V&B approach includes a variety of defined styles, also known as views, that address different concerns of stakeholders. To determine the most appropriate styles for our study, we conducted a survey with 50 practitioners who have experience in software architecture design and distributed architecture. These participants were working in various positions at different software companies. All 17 viewpoints of the V&B approach were presented to them. We began by providing detailed information on each view to the participants, and then asked them to indicate the extent to which they were concerned with each defined view in the V&B approach. Participants were given three options to choose from: “d” for detailed information, “s” for some detailed information, and “o” for overview information. Table 3 presents the survey results for the views with at least one “s” or “d,” where “d” indicates that detailed information is requested by the participant, “s” indicates that some detailed information is requested by the participant, and “o” indicates that overview information is requested by the participant.

TABLE 1 MSA vendor analysis

Feature		AWS	Google Cloud	Microsoft Azure
Communication style	Async communication	AWS MQ AWS SNS AWS SQS AWS Kinesis	Google Dataflow Google Pub/Sub	Azure Queue Storage Azure Service Bus Azure Event Grid Azure Event Hubs
	API gateway	AWS API Gateway	Google Apigee	Azure API Management
Service orchestration		AWS ECS AWS EB AWS EKS	Google Cloud Run Google App Engine Google Kubernete Engine	Azure Container Instances Azure App Service Azure EKS
Service orchestration	Deployment/CI and CD	AWS CodeDeploy AWS CodeBuild AWS CodePipeline	Google Cloud Build	Azure Devops
	Deployment/serverless function	AWS Lambda AWS Step Function	Google Cloud Function Google Cloud Composes	Azure Durable Azure Functions
	Deployment/container	AWS Fargate AWS EKS	Google Cloud Run Google Kubernete Engine	Azure Container Instance Azure Kubernete Service
	Deployment/VM	AWS EC2	Google Compute Engine	Azure VM
	Auto-scaling	AWS EC2 Auto-scaling	Google Computer Engine Auto-scaling	Azure Virtual Machine Scale Set
	Load balancing/server-side	AWS ELB	Google Cloud Engine Load Balancing	Azure Load Balancing
	Service discovery/server-side	AWS Route 53 AWS Cloud Map AWS ELB	Google Cloud DNS Google Cloud Engine Load Balancing	Azure DNS Azure Load Balancing
Service mesh and sidecar pattern		AWS AppMesh	Google Anthos Service Mesh	Azure Service Fabric Mesh
Observability	Log analysis	AWS Elasticsearch Service AWS Redshift AWS Quicksight AWS Athena	Google Elasticsearch Service Google BigQuery	Azure Elasticsearch Service Azure PowerBI Azure Data Lake Analytics
	Exception tracking	AWS CloudWatch	Google Cloud Debugger Google Cloud Trace	Azure Application Insights Azure Monitor
	Log aggregation	AWS CloudWatch	Google Cloud Logging	Azure Application Insights Azure Monitor
	Audit logging	AWS CloudTrail AWS Config	Google Audit Logs Google Cloud Asset Inventory	Azure Monitor
	Distributed tracing	AWS X-Ray	Google Cloud Debugger Google CloudTrace	Azure Monitor
	Monitoring	AWS CloudWatch	Google Cloud Monitoring	Azure Monitor
Provisioning and configuration management		AWS CloudFormation	Google Cloud Deployment Manager	Azure Resource Manager Azure VM extensions Azure Automation
Security		AWS Cognito	Google Firebase Authentication	Azure Active Directory B2C

TABLE 2 Description of key features of MSA derived from vendor and domain analysis (adapted from Reference 16)

ID	Concept	Description
1	Data management and consistency	Ensures the quality of distributed data management and consistency between microservices. Explores techniques for handling data management and consistency challenges in microservice architectures.
2	Communication style	Addresses the importance of establishing stable communication channels between microservices and external components. Investigates different communication methods to enable effective communication in microservices.
3	Service orchestration	Focuses on critical concerns such as auto-scaling, service discovery, resource management, load balancing, and container availability and deployment. Provides methods and concepts for comprehensive handling of these concerns.
4	Decomposition	Guides the fundamental stage of microservice design, determining how the domain model is divided into individual microservices. Explores best practices and strategies for achieving effective decomposition.
5	Service mesh & sidecar pattern	Encompasses the sidecar pattern and the concept of service mesh. Establishes a dedicated infrastructure layer for communication among services, providing resiliency, fault tolerance, and advanced capabilities.
6	Observability	Focuses on obtaining system information, such as performance metrics and component status, to ensure system sustainability. Enables proactive actions and issue prevention based on gathered insights for improved system management.
7	Provisioning and configuration management	Covers the process of setting up system infrastructure and managing resources. Explores specialized tools and approaches for provisioning resources efficiently. Configuration management ensures the system remains in the desired and consistent state.
8	Security	Encompasses various aspects of security, including authentication, authorization, and protection of data in motion and at rest. Addresses the need for robust security measures to ensure secure access, data confidentiality, and integrity across multiple services.
9	Testing	Explores testing strategies suitable for distributed architectures in microservices. Enhances testability and maintenance capabilities by developing appropriate structures for testing use cases spanning multiple services.
10	Resilience and fault tolerance	Acknowledges the inevitability of failures and designs systems to handle them effectively. Focuses on building resilient systems that can withstand and recover from failures, ensuring fault management and maintaining system stability.

TABLE 3 Survey results for selecting architectural design styles

Main stakeholder	Layered view (style)	Decomposition view (style)	Deployment view (style)	Service-oriented architecture view (style)
Software architect	d	d	d	d
Software developer	s	d	s	d
DevOps team member	d	d	d	d
QA	o	d	o	o
Project manager/CTO/Engineering manager	o	d	o	o

We conducted a comprehensive analysis of the architectures offered by key vendors in the field of MSA. We observed that these architectures were often vendor-specific and lacked the use of architectural views. To address this, we identified and highlighted the key components and relationships that are essential in a vendor-agnostic and platform-agnostic microservice architecture. Through our vendor analysis, we evaluated the solutions provided by three major vendors (AWS, Google Cloud, and Microsoft Azure) in the MSA domain and determined which solutions effectively address specific problems.

Furthermore, we leveraged the results of our domain analysis process to inform the design of the reference architecture. Our domain analysis, which included research of academic and multi-vocal literature, as well as expert interviews, enabled us to model the domain using a feature-driven approach. We integrated all of these inputs into the development of our reference architecture, ensuring that it aligns with the needs of stakeholders and is maintainable, scalable, and aligned with industry best practices.

3.4.1 | Family feature model

In our previous research,[16] we presented a feature driven model (Table 4) that we derived from our extensive domain analysis. Incorporating this model into the reference architecture as a guide for identifying the components that should constitute the application architecture can be a valuable approach for practitioners. By defining the necessary components and their relationships in the system as a whole, and then generating the application feature model from the family feature model, this approach ensures that the architecture is aligned with the needs of the stakeholders, maintainable, and scalable.

3.4.2 | Decomposition view

The decomposition view plays a vital role in the microservice reference architecture by providing a clear depiction of the system's segregation of responsibilities. It adheres to the principle of separation of concerns and aims to divide the system into distinct modules, each with its own sub-modules. This view aligns with the "Module Style" within the "Views and Beyond" approach, emphasizing the modular nature of the architecture.

In our proposed method, we generate the decomposition view by leveraging the family feature model as a guiding framework. The family feature model allows us to identify features as implementation units, although it is important to note that some features may be regarded as design patterns without a direct implementation unit equivalent. Furthermore, multiple features can converge to form an implementation unit, highlighting the flexibility and versatility of the microservice reference architecture.

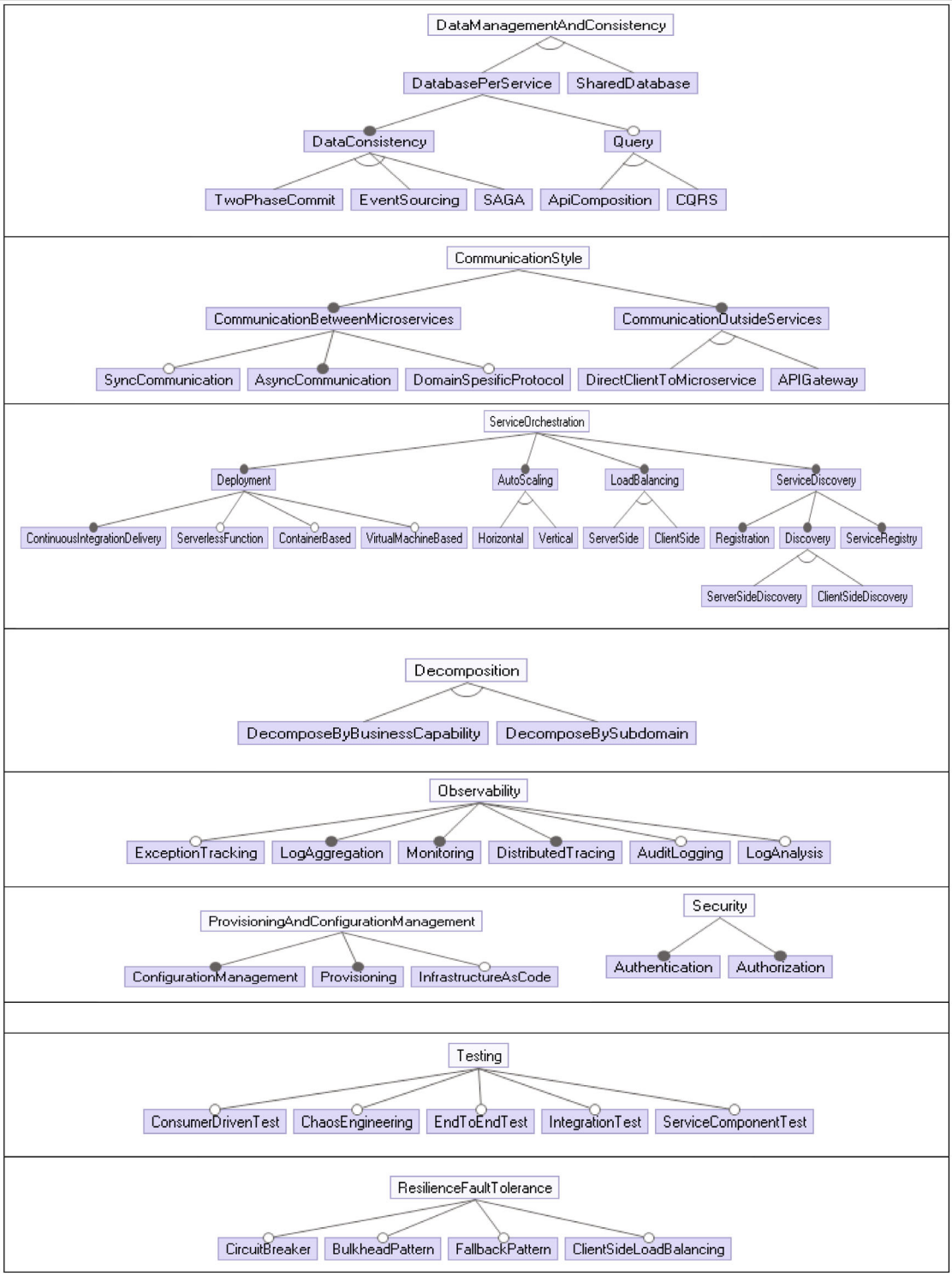
Figure 3 illustrates the decomposition view of the microservice reference architecture, employing distinct colors to represent the main modules. Under each color, various sub-modules are grouped together. This view provides a comprehensive representation of the system's sub-modules, serving as a valuable guide for other architectural views. Its inclusion of the overall structure of the system enhances the understanding and utilization of the architecture as a whole.

3.4.3 | Layered view

The layered architecture is a common modular style in software design. It is created by ordering the modules specified in the decomposition view according to different logical layers, each of which serves a specific purpose and provides a cohesive set of services.[18] The layered view of the microservice reference architecture is illustrated in Figure 4. As seen in the figure, the layered view includes several sidecar layers such as testing, authorization, and authentication. These layers are associated with one or more of the horizontal modules and are represented vertically. In a microservice architecture, the first request is handled by the communication with the outside layer, and then it is forwarded to the appropriate service through service discovery and load balancing processes. Business rules are spread across multiple services and communication between services is done through the communication between microservices module.

In a distributed environment, instances of a microservice can be found on multiple nodes and are constantly dynamic, making it crucial to distribute the load equally. Client-side service discovery is used to distribute the load equally to microservices and to achieve a more available and resilient system. The service registry contains information about the IP and port of each service. Data consistency is also an important aspect of a microservice architecture. One or more techniques from the Data management and consistency layer are used to handle local transactions and provide data consistency. Other sidecar layers are responsible for specific tasks, as outlined in Section 3.2.

TABLE 4 Family feature domain model of MSA



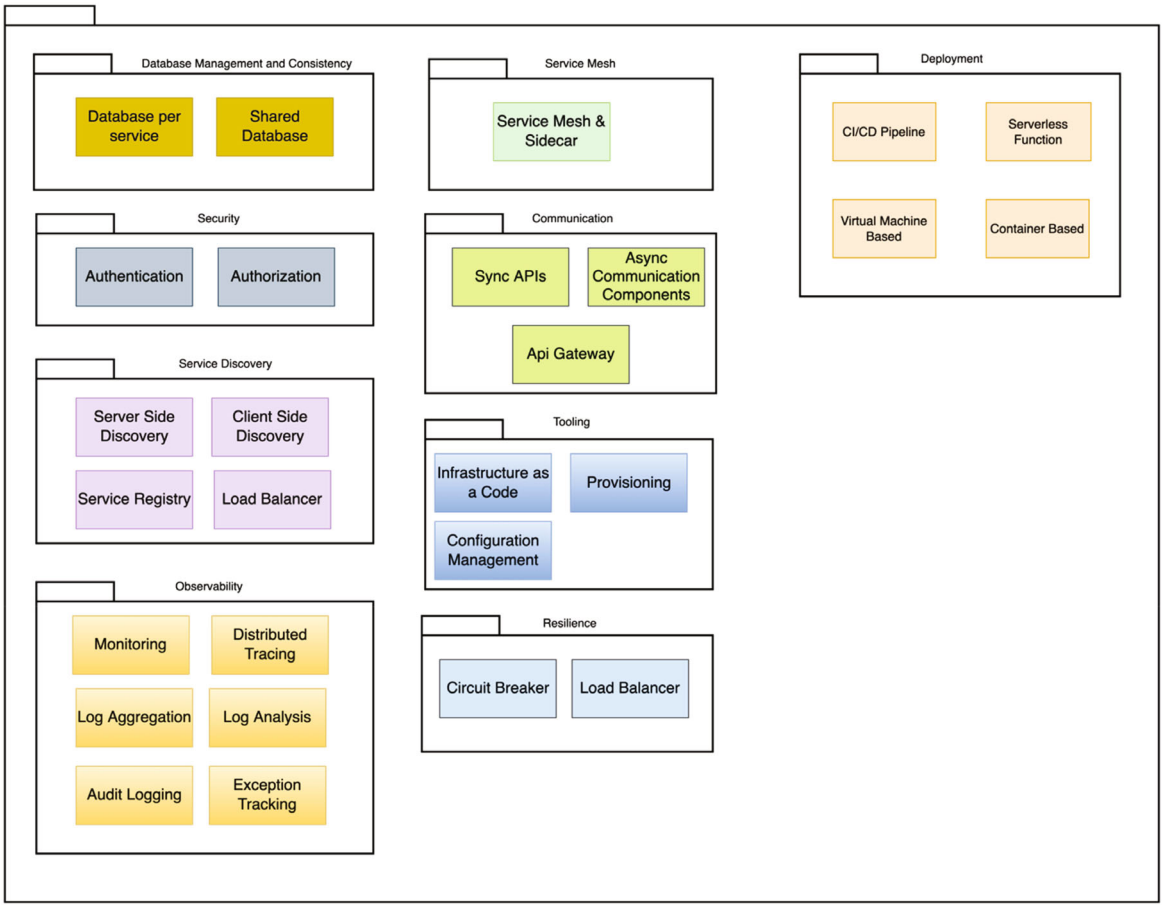


FIGURE 3 Microservice reference architecture—decomposition view

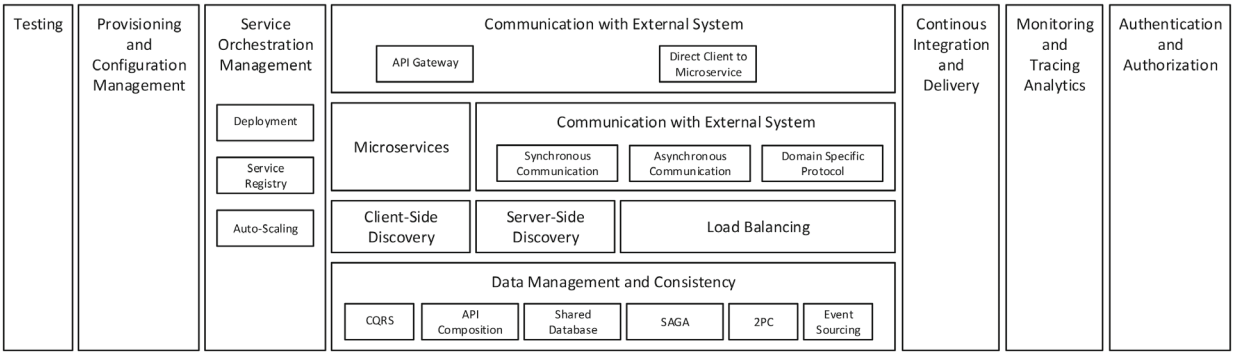


FIGURE 4 Microservice reference architecture—layered view

3.4.4 | Deployment and SOA view

The deployment and service oriented architecture view is a combination of two architectural styles: the deployment view, which uses the allocation style, and the service oriented architecture view, which uses the component & connector (C&C) style. These styles are often used in conjunction with distributed systems.[18] While the layered and decomposition views focus on the modular decomposition of a software system and how these modules interact with each other in terms of layers, the deployment view deals with how these modules are allocated to the hardware of the computing platform.[18] These modules are native to the C&C style.[18]

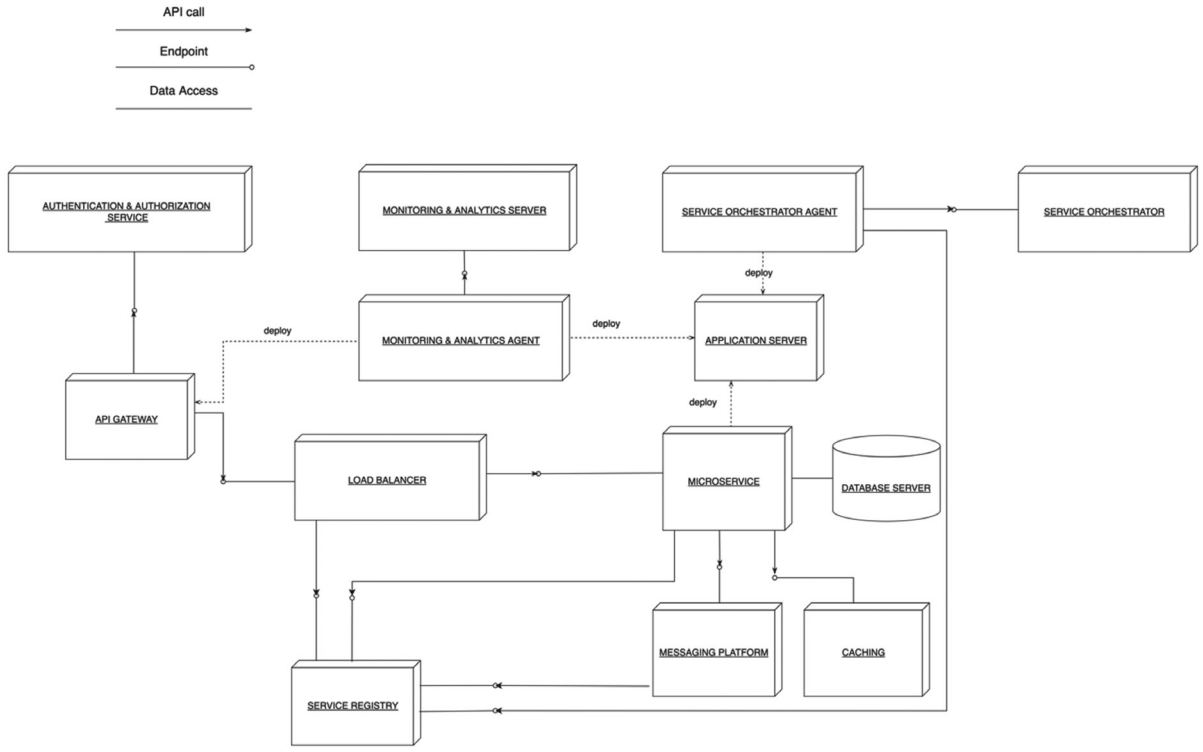


FIGURE 5 Microservice reference architecture—deployment & service oriented architecture view

In distributed and service-based architectures, each service can be independently deployed and distributed. Consequently, they can be represented as separate, autonomous deployable components, as depicted in Figure 5. The service oriented architecture style, a part of the C&C style, establishes relationships between distributed services through the interfaces they provide. Components within the distributed architecture offer services to other components, while other components consume the services provided to them.

Figure 5 presents the combined view of the deployment view and service-oriented architecture view. As illustrated, each component can be independently deployed to the designated machine in the cloud or on-premise. Each component serves as a deployable unit, and some components are deployed to the given machine, as depicted in the figure. Services communicate with each other through the application programming interfaces (APIs) they provide, ensuring the integrity of the system. The selection and allocation of appropriate hardware resources are essential considerations in designing a robust and scalable system. However, in the specific context of Figure 5, the emphasis is on visualizing the high-level architecture and the interactions between the deployable components and services, rather than providing an exhaustive representation of the entire hardware infrastructure. Table 5 provides detailed information about each component and its relationships.

3.5 | Method to derive application architectures

The process of creating an application architecture using the family feature model and microservice reference architecture is illustrated in Figure 6. This approach, which is the result of domain engineering activities, involves analyzing the requirements collected from stakeholders and selecting relevant features from the family feature model to develop the application feature model. The reference architecture and family feature model can also be updated based on feedback received during the development of the application architecture. The views and styles of the reference architecture are then used to derive the application architecture, aligning the software components with the features in the family feature model. This ensures that the application architecture is tailored to the specific needs of the stakeholders, while also adhering to industry best practices and standards.

TABLE 5 Explanation of each component in deployment and service oriented architecture view

1	API gateway	It is a single-entry point for all clients and a place between clients and backend services. It is responsible for transmitting client requests to the backend services. It usually acts as a load balancer and forwards the request to the appropriate backend services. In addition, it performs authentication and authorization controls on the incoming request, and if the request is not secure, it cancels the forwarding request to the backend services. The benefit of this is, for example, when communicating with the outside over https, http can be used inside. Also, metrics are collected from the incoming request by performing some tracing and logging operations on API gateway. If rate limiting is desired, it can be done via API gateway. But the most critical point to consider when using it is the single point of failure situation. If a single instance is deployed, this risk is too high. In order to rule out this risk, a load balancer that will understand the load coming to API gateways can be included in the system, and if the API gateway is down, a new instance must take over. It is not the only method for microservices to communicate with outside. If desired, clients can directly communicate with backend (BE) services, but this is not a recommended method because all clients have access to all services, creating a security vulnerability and considering that BE services are dynamic, and IP and port information are also not fixed. Such cases are the limitations of this method.
2	Authentication & authorization	It is responsible for authenticating and authorizing users to reach the backend services.
3	Load balancer	It is used for distributing the requests to backend services which is considered healthy by looking up Service Registry. In order to eliminate the single point of failure, a new instance of load balancer must take over in the case of failure. The load balancer is a crucial component to increase the availability and scalability of the system. It can be used as two different deployable components with API gateway, or it can be used without API gateway in some cases. In this case, it is possible that the concerns addressed in the API gateway will be moved to the backend services, but this is not a recommended approach.
4	Service registry	It is a key value database containing network information of services. The API it provides is used by many components as shown in Figure 5.
5	Service orchestrator	It is the most comprehensive component, addressing lots of critical concerns, such as auto-scaling, service discovery, resource management, load balancing, container availability, resiliency and deployment. It focuses on the methods and concepts to handle all of these concerns by the help of the agents in the application servers.
6	Service orchestrator agent	It is for caring about the state of the services and sharing this information with the service orchestrator.
7	Messaging platform	It is used in asynchronous communication between microservices. It manages messaging by providing APIs for producing messages and consuming messages by other microservices. Information on which node will consume the messages can be obtained from the service registry. Successfully handling the cases where it is important to consume the messages in the same order as they are produced and trying to consume the message again in case of failure are the subjects of this module.
8	Caching	It is high-speed storage of a subset of data. It is used for many purposes, especially by microservices, but could be used by other components.
9	Microservices	They are small, loosely coupled and independent services organized around business capability. They are deployed to the application servers. They can produce or consume messages for interaction with other microservices. Besides that, they can use service registry to distribute the synchronous requests to other services.
10	Application server	It is a server that hosts microservice applications
11	Monitoring & analytics agent	It is used for collecting information about the modules where it runs. It is responsible for sharing this information with the Monitoring and Analytics Server.
12	Monitoring & analytics server	It is responsible for storing all kinds of logs, metrics, traces, events and so on. Besides, visualizing and querying the logs are also among its responsibilities. It also provides some services to set up alerting mechanisms.
14	Database server	It is a server that runs database application.

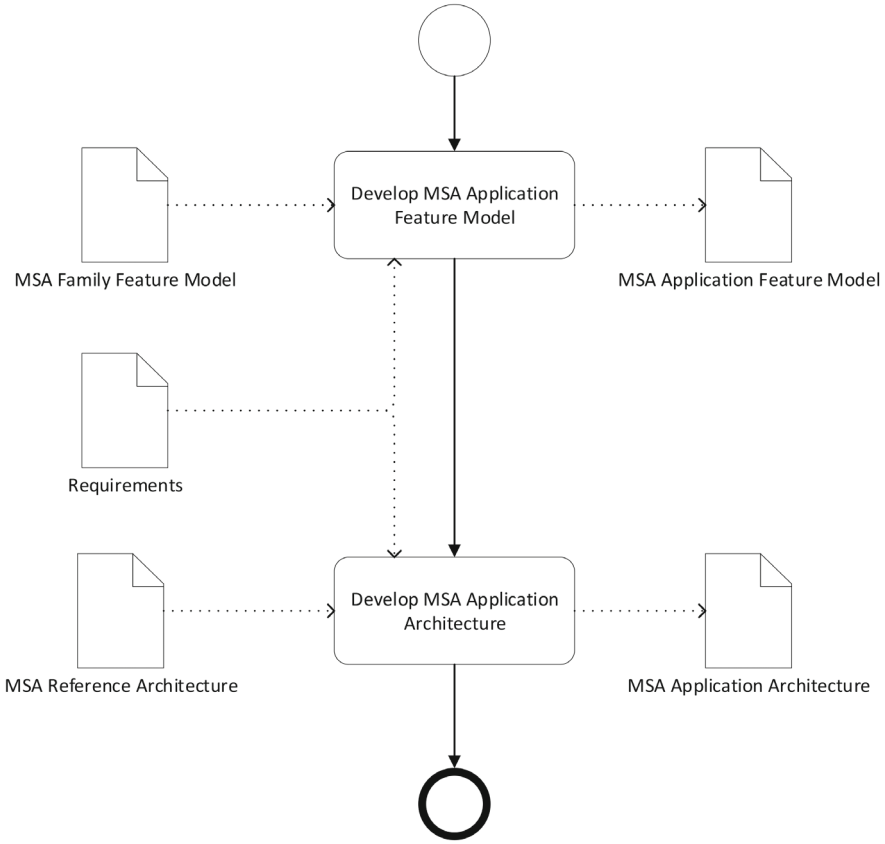


FIGURE 6 Method to derive application architecture from reference architecture

4 | MULTI-CASE STUDY RESEARCH

In this section, microservices reference architecture is evaluated using case study research. In this evaluation, our architecture designs based on viewpoints and feature diagrams are used. Section 4.1 describes the case study protocol. Sections 4.2 and 4.3 present the results from the two cases, respectively.

4.1 | Case study protocol

To validate our approach, we have adopted the case study empirical evaluation protocol as discussed by Runeson and Höst.[23] The protocol consists of the following steps: (1) case study design, (2) preparation for data collection, (3) execution with data collection on the studied case, (4) analysis of collected data, and (5) reporting. Table 6 presents the design of the multi-case study.

We also have adopted holistic and multi-case design as suggested by Yin.[24] The two cases within the multi-case design have emerged in two different software companies and have been prospective studies in which the companies intended to migrate their monolithic applications to MSA. As previously stated, the goal of the multi-case study evaluation is to aid in the design and development process. The purpose of this case study is to assess the architecture designs that have been established, as well as the development process. Data is collected by interviewing the architects, developers and site reliability engineering (SRE) team members and by observing the design and development process.

We discuss the results of both cases by applying a qualitative data analysis approach using radar charts. We have used both direct and indirect data analysis. For direct data analysis, we have conducted semi-structured interviews with architects, developers and SRE team members using the predefined questions in Table 7. The questions have included a 5-point Likert scale (from "1: strongly disagree" to "5: strongly agree") for possible answers. In addition, one more explanation has been requested for each question.

TABLE 6 Case study design

Case study design activity	(For the two cases in the multi-case study)
Goal	Assessing the effectiveness and practicality of the reference architecture
Research questions	How effective is the adopted microservices reference architecture? How practical is the adopted microservices reference architecture?
Data collection	• Observation of the process and systems • Meetings • Indirect and direct data collection through semi-structured interviews (mix of open and closed questions)
Data analysis	Qualitative data analysis using Radar Charts

TABLE 7 Questionnaire for qualitative data analysis

Questions	
1	What is your opinion on the provided application architecture?
2	Do you think that the provided recommended application architecture is of high quality?
3	Do you think that the reference architecture is of high quality?
4	Is the method and the reference architecture sufficient to derive the application architecture?
5	Do you think that the method is practical?
6	Will you use the method again?
7	Do you think that the application of the method can provide a competitive advantage to the organization?
8	Has the usage of the method enhanced your knowledge on MSA?
9	Do you have any suggestions for improving the method?
10	Do you have any suggestions for improving the feature models?
11	Do you have any suggestions for improving the reference architecture?

Both cases in the multi-case study research have followed the steps listed below:

1. An initial interview with architects, SRE team members and developers was planned first. The purpose of this interview was to get a sense of the participants' early ideas and experiences with MSA.
2. In the second phase, we provided a brief presentation on the proposed method's purpose. We also briefly detailed how the approach works as well as the ultimate result.
3. In the third phase, we applied the approach to both cases (elaborated in Sections 4.2 and 4.3).
4. The researchers assessed the architecture design that emerged from the method's application to the prospective cases in the fourth phase.
5. In the fifth phase, the researchers conducted a post-interview to determine the method's effectiveness and practicality.
6. In the sixth phase, the researchers analyzed data from the initial interview and the post interview. The assessment was completed separately but reviewed together to determine the lessons learned.

4.2 | Case study—transportation management

The COVID-19 pandemic has led to a shift towards online activities, including transportation services. As a result, the transportation industry has become more competitive and companies are looking for ways to effectively manage and monitor their processes. This has increased the demand for transportation management systems that are user-friendly, efficient, and financially feasible.

In response to this demand, we have provided a software company with a reference architecture for developing a transportation management solution. The development team, consisting of 11 members including two SREs, one architect, one designer, one business analyst, and six developers, have decided to deploy their services on the Amazon Web Services (AWS) platform. Our reference architecture is designed to be platform-independent, making it a suitable solution for this company. The application architecture derived from the case study protocol is discussed in further detail in the following subsections.

4.2.1 | Application feature model

This application feature model (Table 8) is derived by selecting the features required for this specific case from the family feature model of MSA presented in our prior study[16] and summarized in Section 3.3

4.2.2 | Decomposition view

The decomposition view of the transportation management system is based on the family feature model, as outlined in Section 3.4.2. This view is derived by taking into account the definitions and information provided in the feature model, including common and variable features. Figure 7 illustrates the derived decomposition view of the system.

As the stakeholders' understanding of the system evolves, they will be able to include additional components in the design. Currently, there is no need for a query component, but a Database per Service approach will be utilized for data management and consistency. Monitoring and log analysis will be implemented to track activities, and infrastructure will be configured for load balancing, service discovery, and auto-scaling. The modules used in other parent modules are also depicted in Figure 7.

4.2.3 | Layered view

This layered view, like the decomposition view, is adapted from the reference architecture's layered view diagram shown in Figure 8. The layered view of transportation management system is depicted in Figure 8. Here, the decomposition view's modules are spread among the layers in the layered view.

4.2.4 | Deployment and SOA view

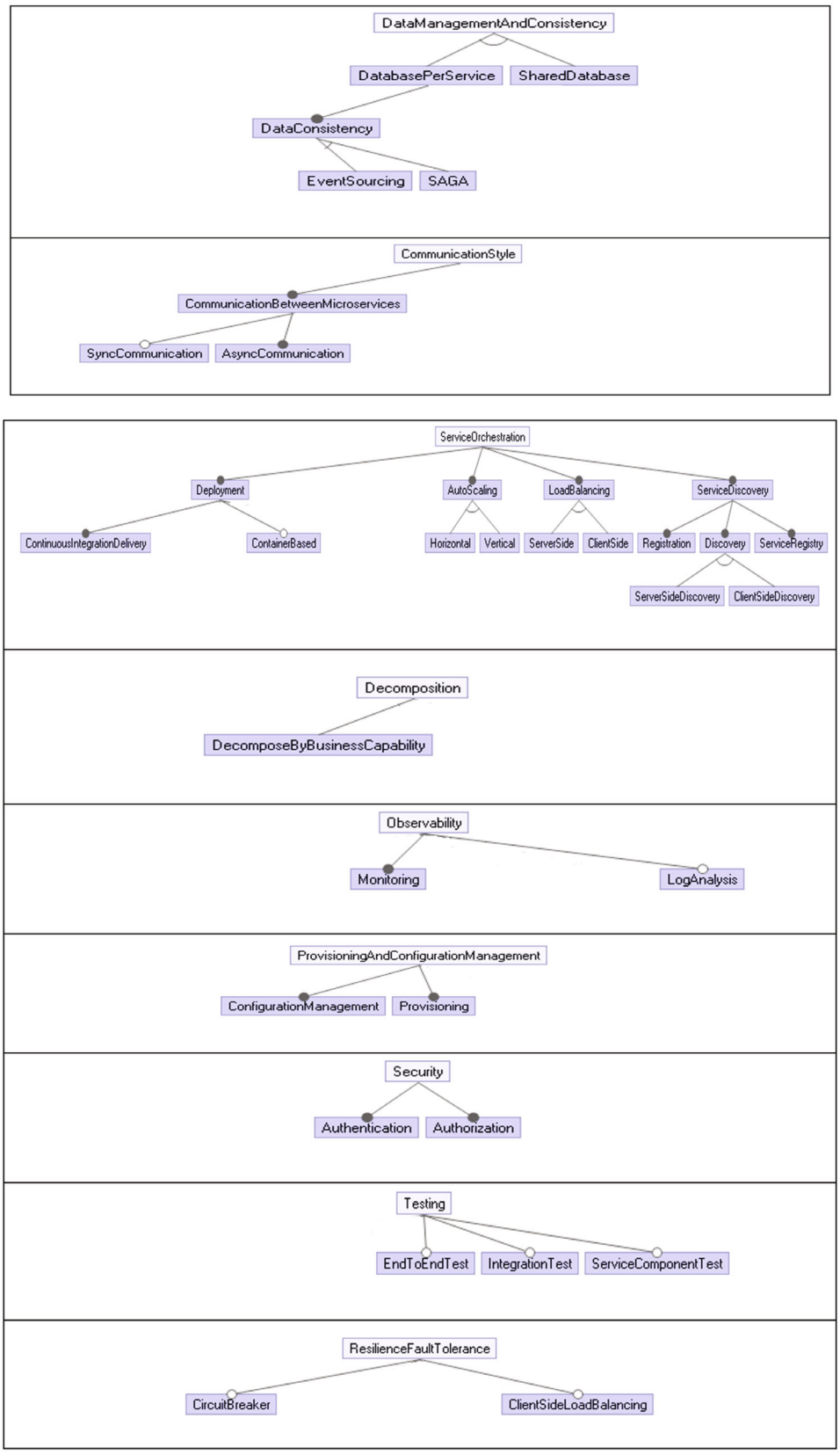
The deployment and service oriented architecture view of the transportation management system is shown in Figure 9. As seen from the figure, almost every deployable unit/module in the reference architecture is also included in the application architecture. Here, a specialized deployable unit is not preferred as an API gateway. AWS costs are the most important factor in reaching this decision. Instead, the load balancer will take over this responsibility. In addition, each microservice is responsible for performing authentication and authorization.

4.3 | Case study—remote team management system

The experienced pandemic has resulted in a significant shift towards remote work, with many companies now requiring solutions to effectively manage and coordinate their remote teams. As the demand for such solutions continues to grow, the competition within this industry has also become increasingly fierce. The software company we are working with aims to develop a solution that addresses a wide range of needs, including time management, project management, and reporting, to streamline the daily operations of remote employees.

We have provided our reference architecture to support the development of this solution. The development team is composed of nine individuals, including one SRE, one architect, one designer, one business analyst, and five developers. They have chosen to deploy their services on Google Cloud and make use of its available services. The application architecture resulting from the application of our case study protocol is outlined in the following subsections.

TABLE 8 Application feature domain model



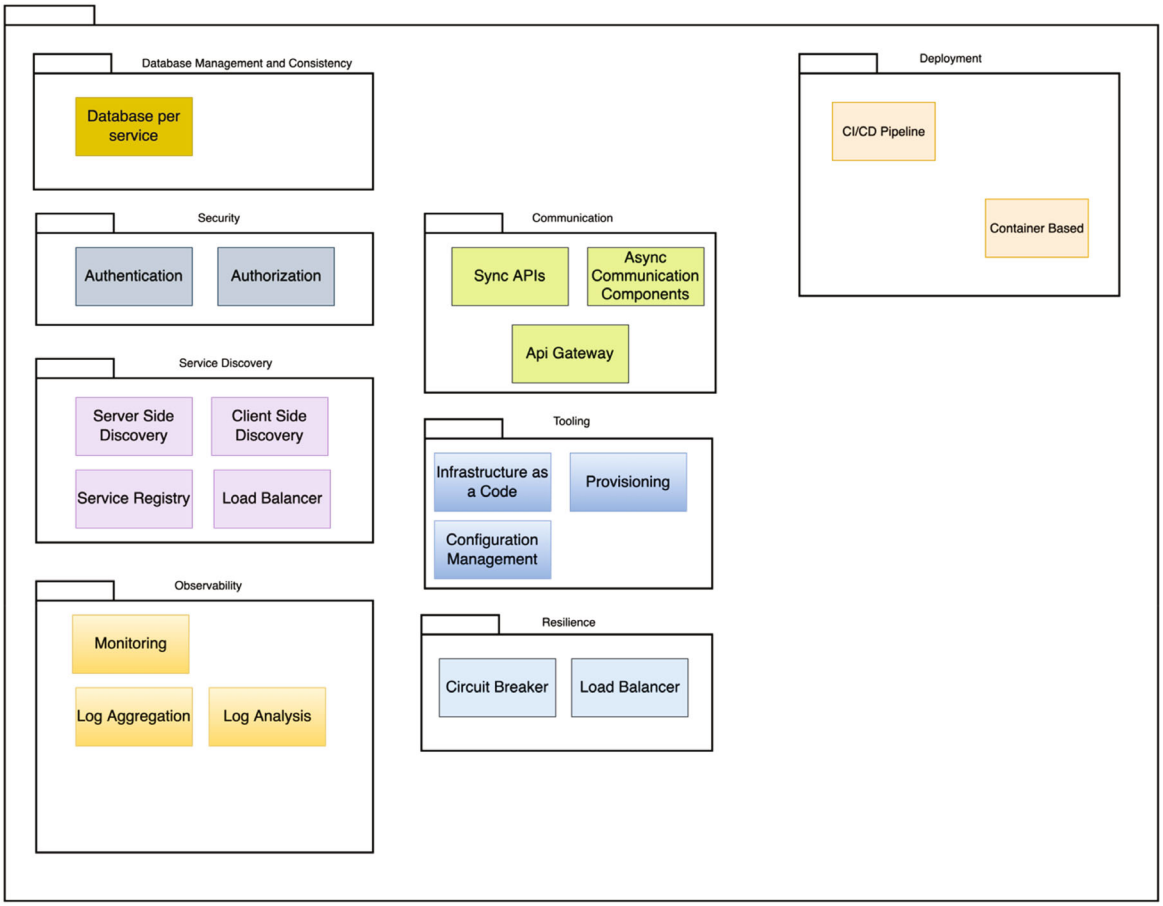


FIGURE 7 Decomposition view for transportation management system

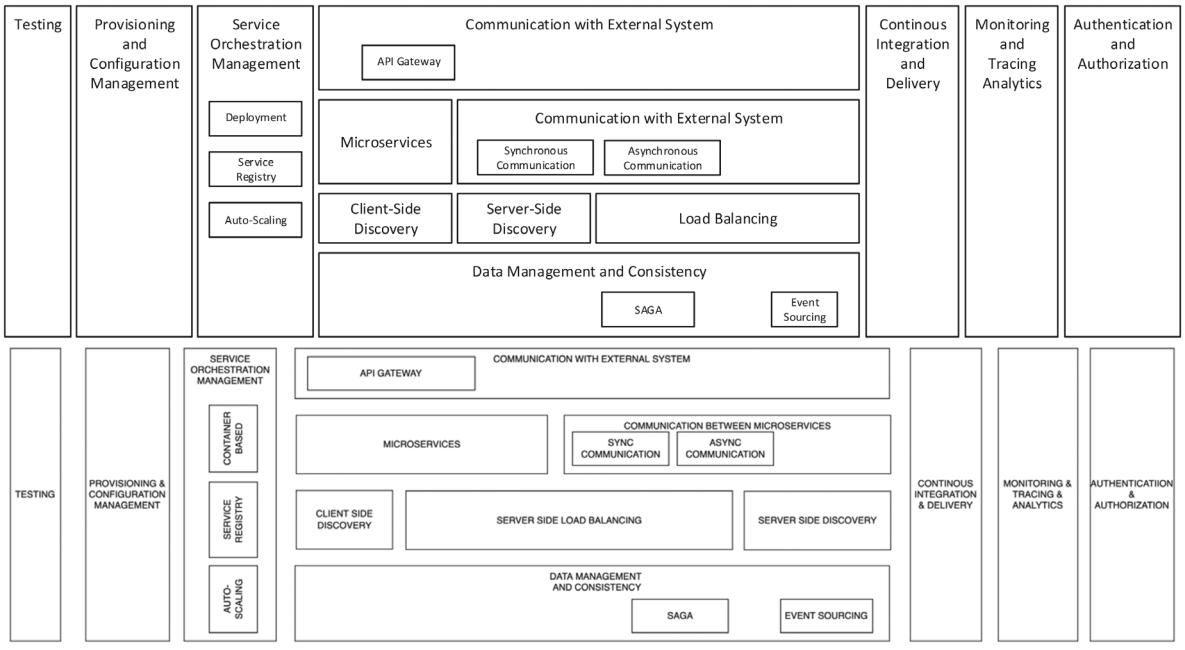


FIGURE 8 Layered view for transportation management system

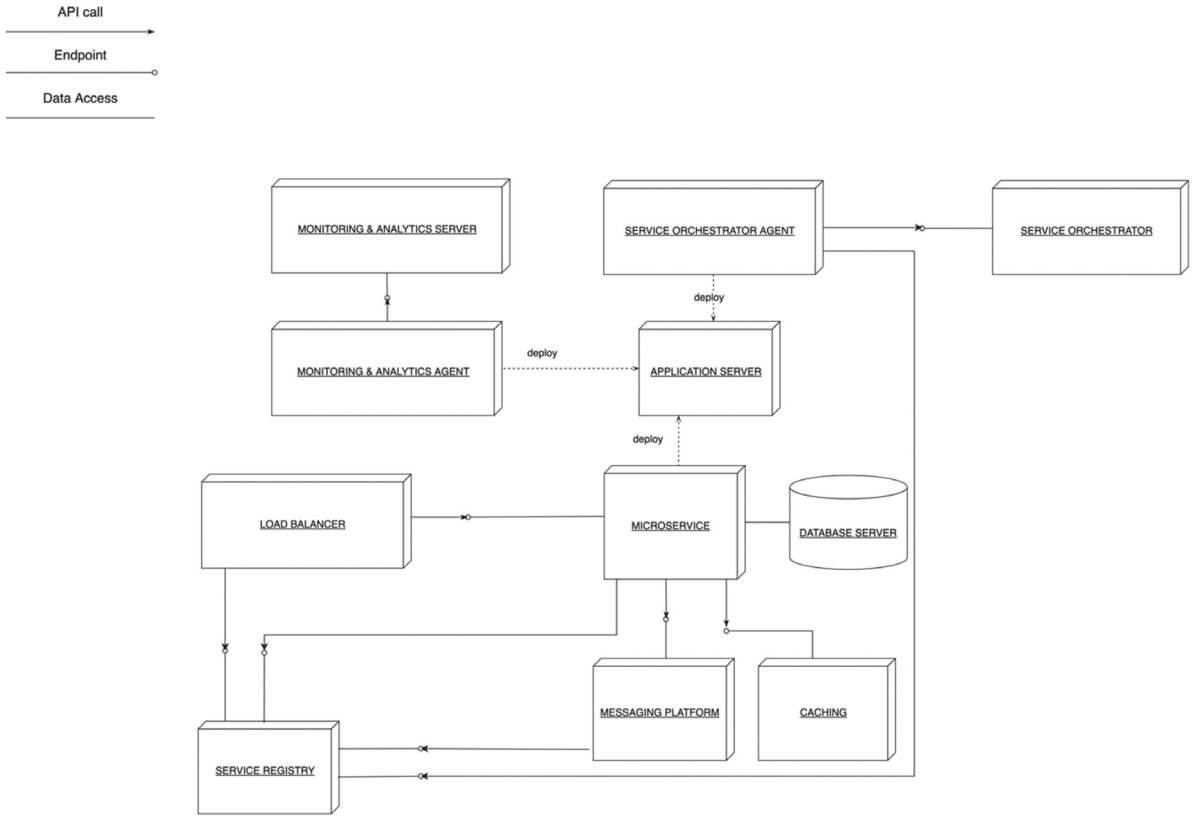


FIGURE 9 Deployment & service oriented architecture view for transportation management system

4.3.1 | Application feature model

The selected feature set for the remote team management case from the family feature model is shown Table 9.

4.3.2 | Decomposition view

Figure 10 shows the decomposition view of the remote team management system. Database per Service for data management and consistency is used. All monitoring modules will be used for monitoring activities. Infrastructures will be configured for load balancing, service discovery and auto-scaling. Besides, instead of through manual process, managing and provisioning of infrastructure will be handled through code. The modules used in other parent modules are also shown in Figure 10.

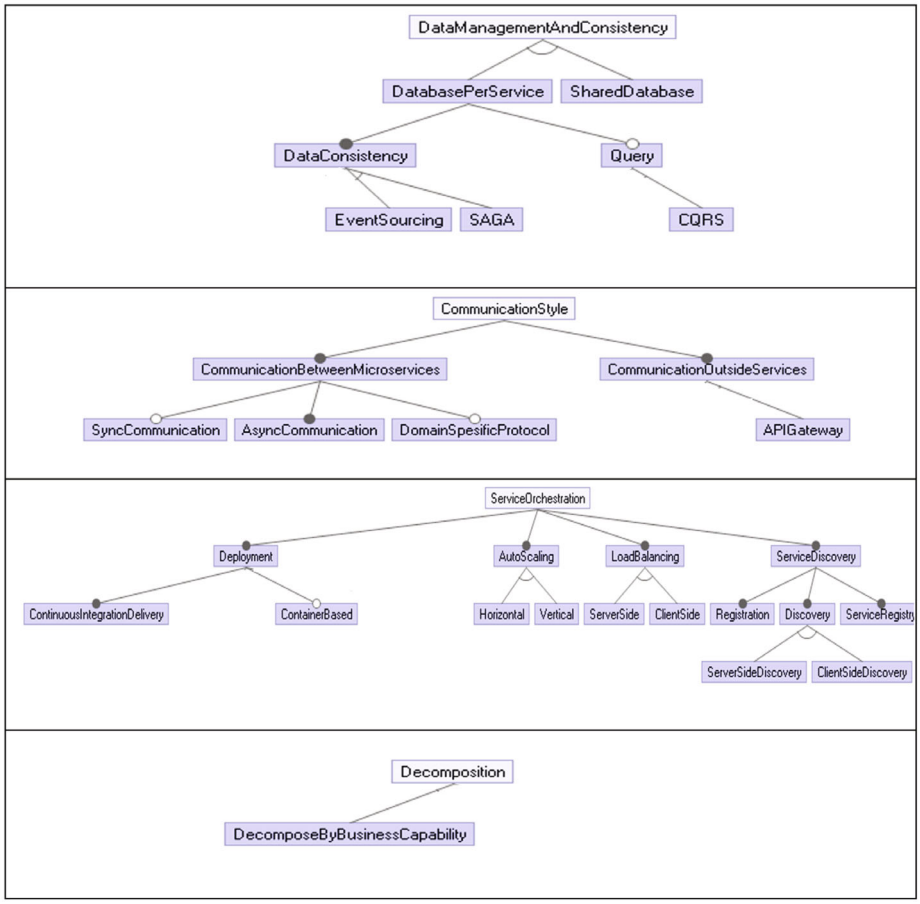
4.3.3 | Layered view

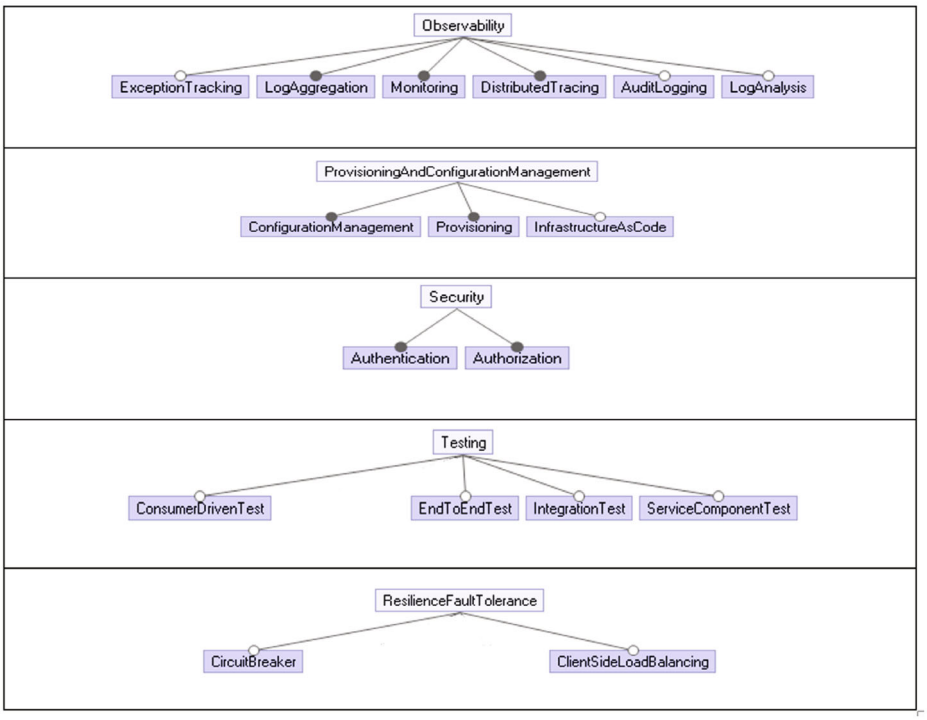
The layered view of remote team management system is shown in Figure 11. It is customized from the layered view of the reference architecture given in Section 3.4.3. Similar to the previous case, the layer view of the modules in the decomposition view is given in the figure.

4.3.4 | Deployment and SOA view

Almost every deployable unit/module in the reference architecture is also included in the application architecture for this case. Since load balancer capability of API gateway will be used here, it is not separately deployable. In addition,

TABLE 9 Application feature domain model





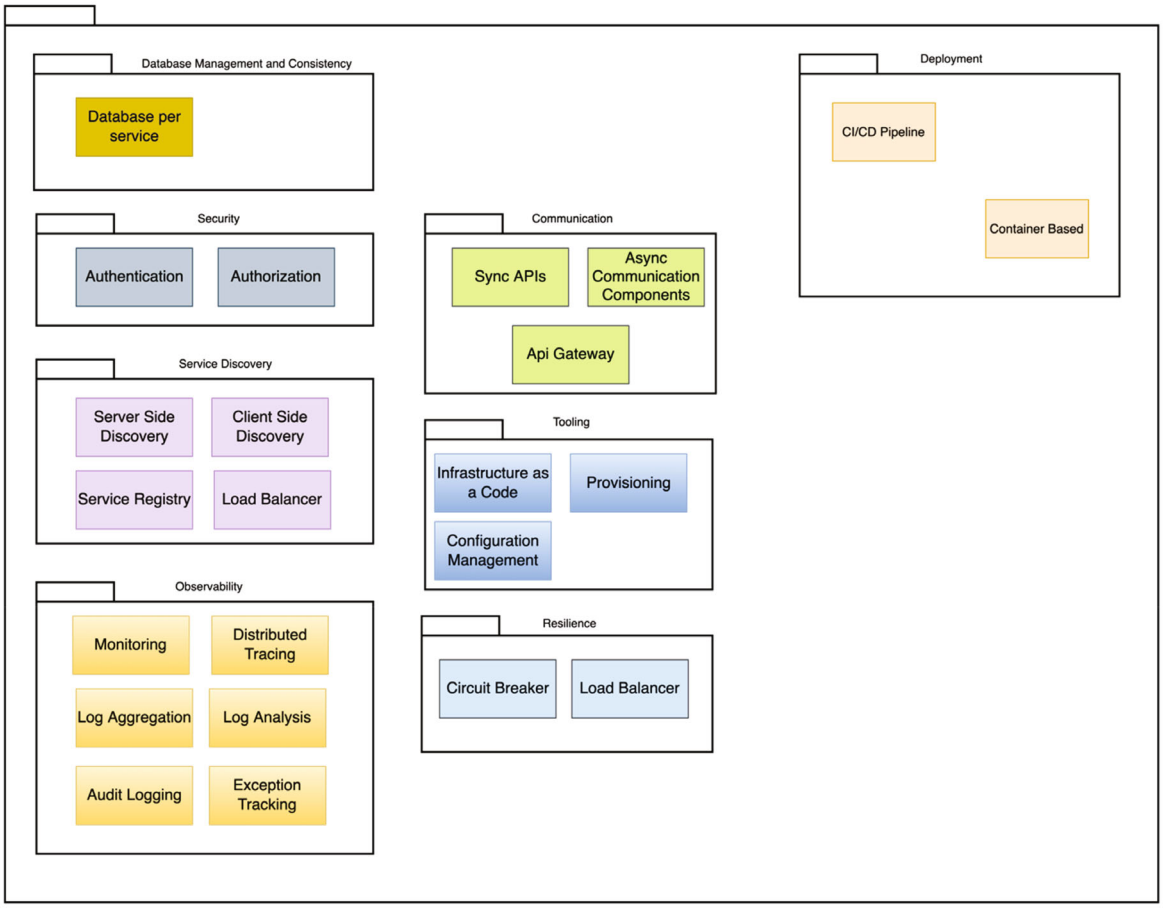


FIGURE 10 Decomposition view for remote team management system

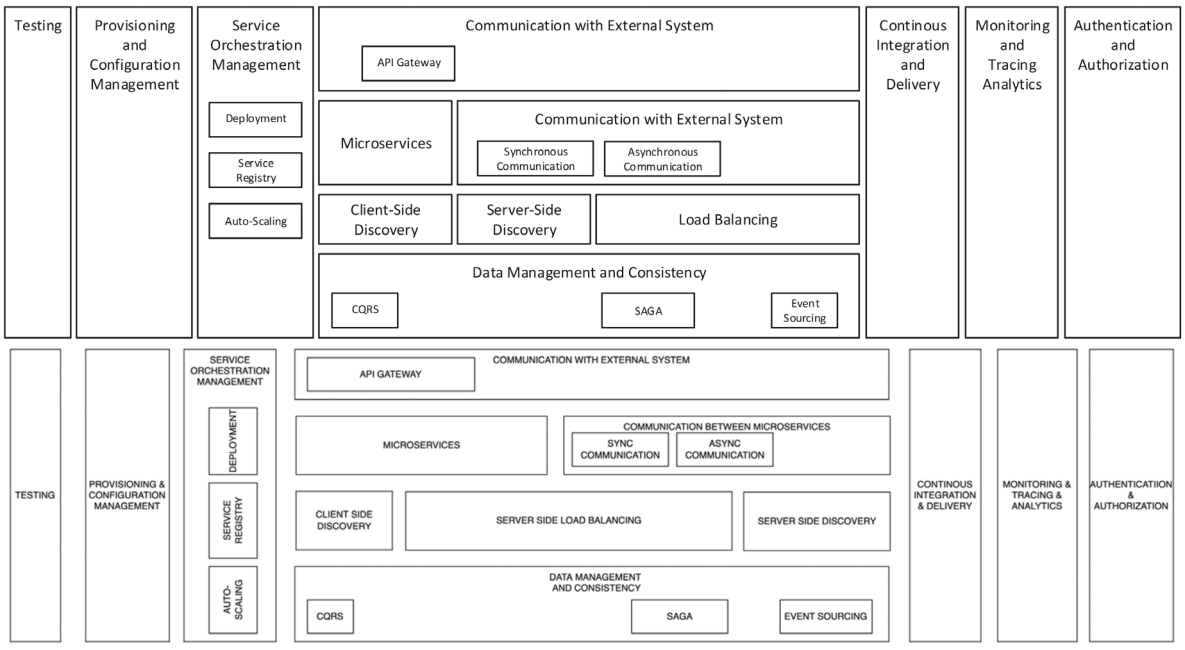


FIGURE 11 Layered view for remote team management system

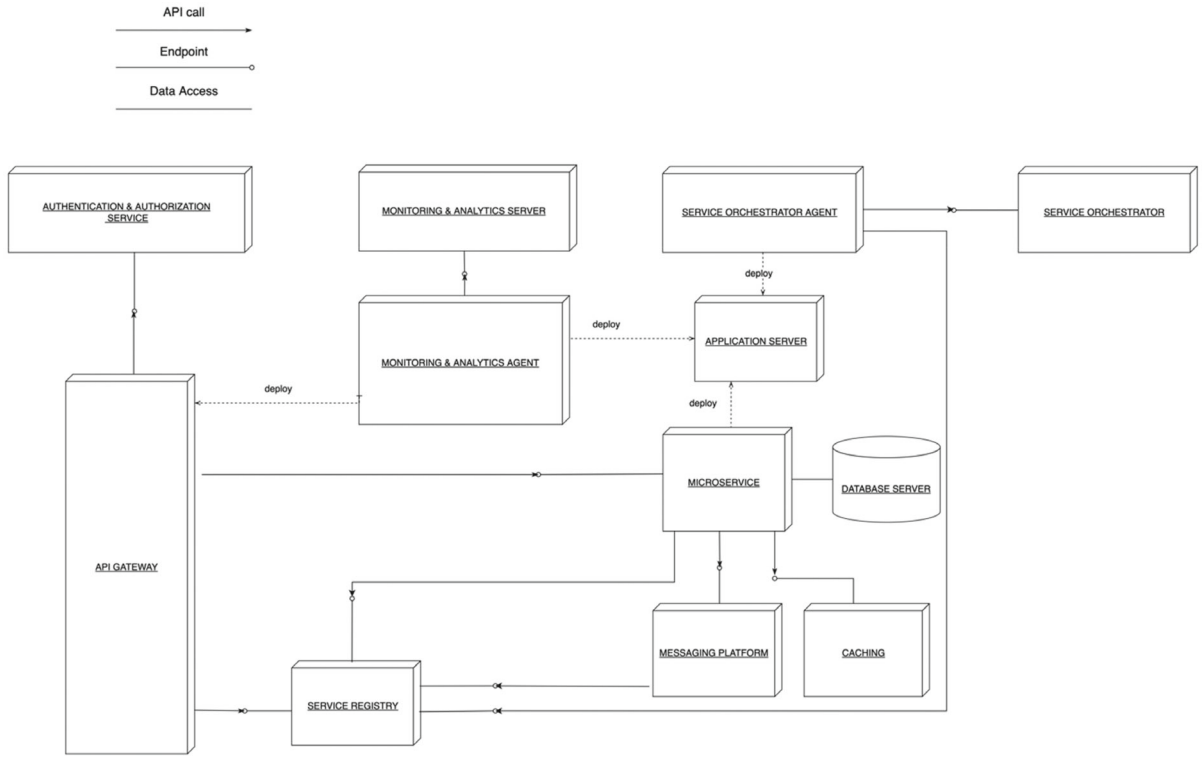


FIGURE 12 Deployment & service oriented architecture view for remote team management system

authentication and authorization will be performed through the API gateway. In addition, the caching module will be implemented due to the need for services to access various information quickly. Figure 12 presents the deployment & service oriented architecture view of remote team management system faster.

5 | DISCUSSION

5.1 | How effective is the reference architecture?

The previous sections have outlined how our reference architecture can be applied in a multi-case study design to create application architecture for various scenarios. By using the research questions defined in our case study protocol, we evaluated the results of two prospective cases. The interview results, presented in the form of a radar chart in Figure 13, demonstrate the effectiveness of our reference architecture in both cases. The scores of 4 and above in the answers to questions 1 to 3 for the first case study indicate that the reference architecture was successful in meeting the needs of the transportation management system. Additionally, the feedback received for questions 8 to 11 was overwhelmingly positive, with the development team praising the ease of adapting the architecture to the AWS environment and the guidance provided by the reference architecture.

The team members also noted that while architectural changes may be necessary in the future, they felt confident in their ability to follow the reference architecture and application architecture development methodology again. They suggested that future enhancements to the reference architecture could include additional views to address the changing needs of the industry. Overall, the team felt that the proposed approach significantly reduced the learning curve and provided clear concepts and methods to follow, making the process of adapting the application architecture more comfortable.

The results of the second case, as depicted in Figure 14, indicate that our reference architecture was highly effective in facilitating the development of an efficient remote team management solution. The scores of 4 and above in the answers to questions 1–3 demonstrate that the proposed approach was well-received by the development team. The interviewees noted that the reference architecture provided clear guidance and sufficient infrastructure support, which enabled them

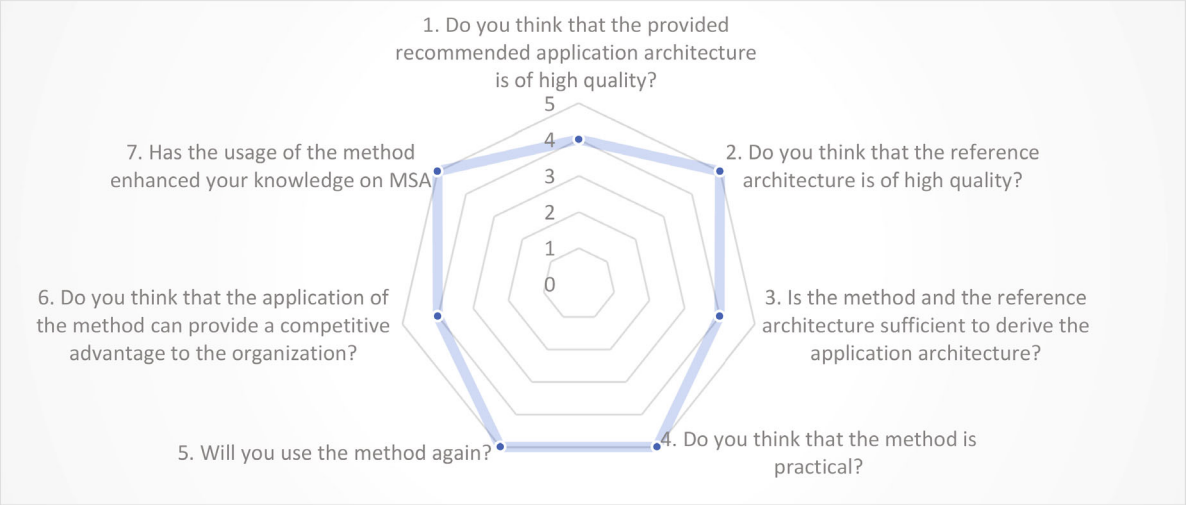


FIGURE 13 Interview results for transportation management system

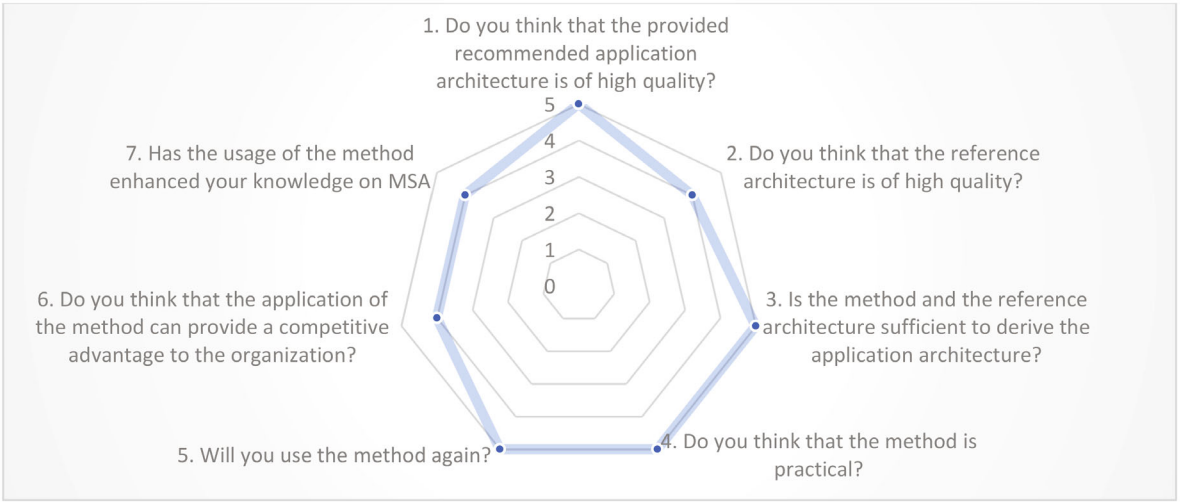


FIGURE 14 Interview results for remote team management system

to adapt the application architecture comfortably and reduce the learning curve. Additionally, the comments received from the developers and architects highlighted the benefits of the proposed approach, including the ease of understanding and the level of detail provided in the methodology for deriving the application architecture. As the architecture had not yet been implemented, the team members also noted that they would apply the same approach again in case any changes were needed during the implementation phase.

5.2 | How practical is the reference architecture?

The results of our multi-case study design reveal that the reference architecture and application architecture development approach were highly effective and practical for both case companies. As shown in Figure 14, the scores for questions 4 to 7 and open-ended questions were all 4 or higher, indicating a high level of satisfaction with the method.

In the first case, the interviewees found the family feature diagram to be a valuable tool for quickly identifying possible components and avoiding wasted time. They also noted that the reference architecture and application architecture

development approach streamlined the decision-making process and increased awareness of microservices architecture concepts. Additionally, they felt that the approach enabled them to make early decisions that would prevent negative impacts on the architecture in the future.

Similarly, in the second case, the interviewees found the approach to be useful and practical, and it gave them confidence in their work. They also provided suggestions for improving the reference architecture in the future, such as providing template application architectures in addition to the reference architecture to speed up implementation. Overall, the results of the study demonstrate that the proposed approach is a valuable tool for creating effective and practical application architectures.

5.3 | Threats to validity

Our reference architecture is designed to be general enough to be applied to a variety of application architectures. However, it is important to note that like other reference architectures, it may not provide all necessary information and may not cover certain specific features that were not anticipated. In this study, our focus was on demonstrating the effectiveness and practicality of our reference architecture and application architecture development approach. It was found to be useful in two different case companies, but we acknowledge that it is not exhaustive and further studies may be necessary to improve and expand it.

In our multi-case study, we have not focused on the implementation of the entire system due to the focus on the design phase and confidentiality reasons. However, future studies could be conducted to examine the implementation phase. The feature diagrams and reference architecture are both easily extensible, allowing for a more comprehensive reference architecture and family feature diagram to be developed in the future. Additionally, the reference architecture can be expanded by adding different architectural views as needed.

To validate our approach, we employed a systematic, multi-case study research method. However, every empirical study faces a number of potential threats to its validity. For our multi-case study, the threats of construct validity, internal validity, and external validity were considered. Construct validity refers to the extent to which the measures used accurately reflect the researchers' intentions and the study objectives. The reference architecture development approach helps ensure construct validity to some extent. Table 10 lists the different threats to construct validity and the countermeasures used to minimize or eliminate their undesired effects.

Internal validity refers to the existence of a causal link between the treatment (our reference architecture development approach) and the response (positive feedback from the interviewees). The use of a reference architecture development approach and confirmation of feedback through multiple rounds of interviews help to establish internal validity.

External validity refers to the ability to generalize the study's results to other settings and populations. Despite selecting cases from different companies and vendors to increase external validity, more work is needed to further improve the generalizability of our findings.

TABLE 10 Threats to construct validity and countermeasures

Threat	Countermeasures
Interviewees' incorrect perceptions of the questions' descriptions	For the questions and answers, we used the ideas given by Kitchenham and Pfleeger.[25] We included thorough explanations to guarantee that each person's understanding of the questions is unique.
Incorrect understanding of descriptions of replies by interviewees, as well as incorrect answer selection	It might be difficult to tell the difference between "Strongly Agree" and "Agree." We defined each scale in detail for each Likert-scale question to mitigate this treat.
The interviewees' incorrect understanding of the open-ended questions	We double-checked the interpretation of the questions with those who were interviewed to mitigate this threat.
Researchers' incorrect interpretation of the interviewed people's responses	Both researchers were present in the interview to establish observer triangulation, which mitigated this threat.

6 | RELATED WORK

Few studies have addressed the architectural aspect of MSA-based development. Some come with a general reference architecture, while others come with an architecture that focuses on the specific aspects of MSA or the particular domains. In this section, we provide an overview of the current studies.

Yu et al.[26] discussed the key characteristics of MSA. They proposed a reference architecture with main building blocks and key components. They also emphasized some common issues and solution alternatives for them, while building microservice-based applications, such as the uncertainty in business ownership and communication problems. However, they are a bit far from today's concerns and issues because with the increase in the usage rate of microservices, more diverse and more critical concerns have started to emerge. Aside from that, even though this study provides general guidance by involving many buildings blocks such as service API registry, API proxy and so on it does not propose a systematic guidance to help practitioners choose the best-fit components for each concern while building microservices.

Baylov and Dimov[27] proposed a reference architecture for self-adaptive microservice systems. They focused on five basic components in their study. These were service consumer, service registry, service provider, service instance, and adaptation registry. Their interaction with each other and the purpose for which they exist were also explained in the reference model. The authors also defined how the reference architecture they proposed could be used through an example application. However, this study is lacking sufficient guidance for MSA based application development as well as evidence for the verification of the reference architecture.

Aksakalli et al.[28] have proposed a model-driven architecture that offers automated deployment alternatives for MSA based systems. The purpose of the architecture is to minimize the execution cost and also the communication cost between microservices, and to use cloud resources efficiently. The architecture consists of five main components. These are microservice data exchange metamodel, microservice definition and communication metamodel, microservice infrastructure metamodel, microservice runtime execution configuration metamodel, and finally, microservice deployment metamodel. The proposed architecture is implemented using genetic and minimum nodes algorithms in an example application and the alternative results of deployment are shared. This study proposes a model only for the deployment area, and the results are discussed along with the application of the proposed architecture.

In addition to academic studies, giant technology companies have also developed reference architectures, but mostly focused on their own technologies or services.[29-32] These reference architectures have addressed many critical building blocks in the system. In addition, sample application architectures have been shared with the reference architecture on how to use it within their own services or technologies. Although these sample architectures cannot be used fully in more complex architectures, they are considered to be useful for new starters. However, since these architectures are more technology-oriented and most of the services provided by giant technology providers are managed, practitioners could be trouble to comprehend the logic under the hood.

Based on the literature analysis presented in this section, we observe that the studies mostly focused on the use of MSA for particular concerns. Additionally, in few existing studies, the reference architecture was not dealt with comprehensively, and thus has not reached sufficient maturity to serve as a guide for developing MSA based application architectures.

7 | CONCLUSION

Microservice architecture is a popular approach in software development due to its agility and autonomy. However, despite the abundance of studies suggesting a reference architecture for MSA in the scientific literature, these studies often remain at an abstract level, lacking a clear method for deriving an application architecture. Our study aimed to address this gap by providing a systematic method for deriving an application architecture from a reference architecture.

To achieve this, we first conducted a market analysis to identify the reference architecture that best met the needs of our study. We then performed a systematic literature review to understand the challenges faced in the field of MSA and proposed solutions to these challenges. We adopted a domain-driven approach to define the family feature model of MSA, which represents common and variant features. Finally, we represented and shared the reference architecture using different architectural views, and provided an architectural design method for deriving the application architecture from the reference architecture.

To demonstrate the effectiveness and practicality of our approach, we applied a multi-case study research design. We used our reference architecture to design two specific application architectures: one for a transportation management

system and one for a remote team management system. The results of our case studies showed that both the reference architecture and the application architecture development method were beneficial for deriving concrete application architectures. The overall approach was found to be practical and effective.

Our study makes valuable contributions to both practitioners and researchers working on MSA. Practitioners can use the reference architecture and the application architecture development method to evaluate, derive, or improve their own application architectures. Researchers can use our study as a foundation for further assessing or improving the reference architecture. As an example, our future work includes studying implementation approaches for the application architecture. Additionally, our family feature model and characterization of features on a cloud provider basis can serve as a guide for companies that are currently working with a cloud provider but may be considering a change, highlighting which solutions are available on other cloud providers for each feature and how these solutions should be used. However, it's important to note that this guidance needs to be further evaluated with more comprehensive studies.

Our proposed reference architecture is specifically designed for microservice architecture, but it can also be adapted for use in distributed systems, as microservice architecture is a specialized version of distributed architecture. By using the method we presented, it is possible to develop a reference architecture for a broader range of distributed architectures. Distributed systems are complex and require many design decisions, having a reference architecture that considers these decisions and provides guidance can help developers to navigate through the implementation of distributed systems and make the process more structured and manageable.

AUTHOR CONTRIBUTIONS

Mehmet Söylemez: Methodology (lead); conceptualization (lead); writing – original draft (lead); formal analysis (lead); writing – review and editing (equal). **Bedir Tekinerdogan**: Methodology (lead); conceptualization (lead); writing – original draft (lead); formal analysis (lead); writing – review and editing (equal). **Ayça Kolukısa Tarhan**: Methodology (lead); conceptualization (lead); writing – original draft (lead); formal analysis (lead); writing – review and editing (equal).

FUNDING INFORMATION

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

CONFLICT OF INTEREST STATEMENT

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this article.

DATA AVAILABILITY STATEMENT

Data sharing is not applicable to this article as no datasets were generated or analysed during the current study.

ORCID

Bedir Tekinerdogan https://orcid.org/0000-0002-8538-7261

REFERENCES

1. Newman S. *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. O'Reilly Media, Inc.; 2019.
2. Zimmermann O. Microservices tenets. *Comput Sci*. 2017;32:301-310. doi:10.1007/s00450-016-0337-0
3. Newman S. *Building Microservices*. O'Reilly Media, Inc.; 2015 ISBN: 1491950358, 9781491950357.
4. Fowler M, Lewis J. Microservices. *Softw Pract Exper*. 2014;44(12):1407-1408. doi:10.1002/spe.2329
5. Francesco PD, Lago P, Malavolta I. Architecting with microservices: a systematic mapping study. *J Syst Softw*. 2019;150:77-97. doi:10.1016/j.jss.2019.01.001
6. Jamshidi P, Pahl C, Mendonca NC, Lewis J, Tilkov S. Microservices: the journey so far and challenges ahead. *IEEE Softw*. 2018;35:24-35. doi:10.1109/MS.2018.2141039
7. Josuttis N. *Soa in Practice: The Art of Distributed System Design*. O'Reilly Media, Inc.; 2007 ISBN: 0596529554.
8. Bondi B. Characteristics of scalability and their impact on performance. Proceedings of the 2nd International Workshop on Software and Performance (WOSP); 2000. doi:10.1145/350391
9. Benevides R. Istio on Kubernetes. Accessed June 2023. http://bit.ly/istio-kubernetes%0A
10. Evans E. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley; 2004.

11. Richardson C. *Microservices Patterns*. Manning Publications; 2018.
12. Bass L, Clements P, Kazman R. *Software Architecture in Practice, SEI Series in Software Engineering*. Addison-Wesley; 2003 ISBN: 9780321154958.
13. Jogalekar P, Woodside M. Evaluating the scalability of distributed systems. *IEEE Trans Parallel Distrib Syst*. 2000;11:589-603. doi:10.1109/71.862209
14. Blinowski G, Ojdowska A, Przybylek A. Monolithic vs. microservice architecture: a performance and scalability evaluation. *IEEE Access*. 2022;10:20357-20374. doi:10.1109/ACCESS.2022.3152803
15. Stefanko M, Chaloupka O, Rossi B. The saga pattern in a reactive microservices environment. Proceedings of the 14th International Conference on Software Technologies (ICSOFT 2019); 2019:483-490. doi:10.5220/0007918704830490
16. Shadija D, Rezai M, Hill R. Towards an Understanding of Microservices. Proceedings of the 2017 23rd IEEE International Conference on Automation and Computing (ICAC); 2017. doi:10.23919/IConAC.2017.8082018
17. Microservice Decomposition: A Case Study of a Large Industrial Software Migration in the Automotive Industry. Accessed June 2023. https://www.big.tuwien.ac.at/publication/tuw-292039/
18. Clements P, Bachmann F, Bass L, et al. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley; 2010 ISBN: 9780321552686.
19. Kassahun A. Aligning business processes and it of multiple collaborating organisations. 2017. doi:10.18174/414988
20. Giudici G, Milne A, Vinogradov D. Cryptocurrencies: market analysis and perspectives. *J Ind Bus Econ*. 2020;47:1-18. doi:10.1007/S40812-019-00138-6
21. Kitchenham B, Brereton OP, Budgen D, Turner M, Bailey J, Linkman S. Systematic literature reviews in software engineering – a systematic literature review. *Inf Softw Technol*. 2009;51:7-15. doi:10.1016/j.infsof.2008.09.009
22. Tekinerdogan B, Öztürk K. Feature-driven design of SaaS architectures. *Software Engineering Frameworks for the Cloud Computing Paradigm*. Springer; 2013:189-212. doi:10.1007/978-1-4471-5031-2_9
23. Runeson P, Höst M. Guidelines for conducting and reporting case study research in software engineering. *Empir Softw Eng*. 2009;14:131-164. doi:10.1007/S10664-008-9102-8
24. Yin RK. *Case Study Research: Design and Methods, Applied Social Research Methods*. SAGE Publications; 2009 ISBN: 9781412960991.
25. Kitchenham BA, Pfleeger SL. Principles of survey research. *ACM SIGSOFT Softw Eng Notes*. 2002;27:20-24. doi:10.1145/511152.511155
26. Yu Y, Silveira H, Sundaram M. A microservice based reference architecture model in the context of enterprise architecture. Proceedings of the 2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC); 2017:1856-1860. doi:10.1109/IMCEC.2016.7867539
27. Baylov K, Dimov A. Reference architecture for self-adaptive microservice systems. *Stud Comput Intell*. 2017;737:297-303. doi:10.1007/978-3-319-66379-1_26
28. Aksakalli IK, Celik T, Can AB, Tekinerdogan B. A model-driven architecture for automated deployment of microservices. *Appl Sci*. 2021;11:9617. doi:10.3390/APP11209617
29. Microservices Architecture on AWS - Implementing Microservices on AWS. Accessed April 26, 2022. https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/simple-microservices-architecture-on-aws.html
30. Microservice Architecture Reference Architectures 2017 | Red Hat Customer Portal. Accessed April 26, 2022. https://access.redhat.com/documentation/en-us/reference_architectures/2017/html/microservice_architecture/index
31. Microservices Architecture Design - Azure Architecture Center | Microsoft Docs. Accessed April 26, 2022. https://docs.microsoft.com/en-us/azure/architecture/microservices/
32. Microservices Architecture: Reference Diagram - IBM Cloud Architecture Center. Accessed April 26, 2022. https://www.ibm.com/cloud/architecture/architectures/microservices/reference-architecture/

How to cite this article: Söylemez M, Tekinerdogan B, Tarhan AK. Microservice reference architecture design: A multi-case study. *Softw: Pract Exper*. 2023;1-27. doi: 10.1002/spe.3241