Cloud Architecture Evaluation

Master of Science (Tech) Thesis
University of Turku
Department of Computing
Software Engineering
2023
Arttu Salmijärvi

The originality of this thesis has been checked in accordance with the University of Turku quality assurance system using the Turnitin OriginalityCheck service.

UNIVERSITY OF TURKU
Department of Computing

Arttu Salmijärvi: Cloud Architecture Evaluation

Master of Science (Tech) Thesis, 56 p.
Software Engineering
April 2023

Cloud computing has introduced numerous ways to build software systems in the cloud environment. The complexity of today's system architectures require architecture evaluation in the designing phase of the system, in the implementation phase, and in the maintenance phase. There are many different architecture evaluation models. This thesis discusses three different evaluation models: architecture tradeoff analysis method, cost-benefit analysis method, and AWS Well-Architected framework. The AWS Well-Architected framework is deeply evaluated by performing an architectural evaluation for the case study software: Lixani 5. This thesis introduces and compares the opportunities for cloud architecture evaluation by literature review, case study, and interviews with experts.

The thesis begins with introduction to cloud computing, cloud architecture models and architecture evaluation methods. An architecture evaluation for a case study software is then carried out. This thesis also contains interviews with experts, producing knowledge on how the system architecture is being evaluated in the field. The research methods used in the thesis are literature review, case study, and expert interviews. This thesis attempts to describe and assess the architecture evaluation models by using the research methods. In addition, this thesis introduces and discusses the case study software – Lixani 5 – and its architectural decisions.

Based on research in the thesis it was noted that all three studied software architecture evaluation models are suitable options for reviewing software architecture. All models included positive and negative aspects and none of them was seen as superior compared to the others. Based on the interviews with experts it was noted that there are also multiple other efficient ways to evaluate the system architecture than the models discussed in the thesis. These ways included a technology audit template and a proof-of-concept culture.

Keywords: software architecture, software architecture evaluation, cloud computing

Contents

Figures

1 Introduction

Software architecture is the structure of the system, the model that is targeted when a software is implemented. The architecture is overall an important part of the software development process, but it comes even more important when the software grows. The bigger the software is the more complex the architecture becomes. Architecture evaluation is a way of inspecting the architectural decisions in the start of the development, it is a chance to be constantly aware of the architectural state, and it can proactively prevent major turnarounds in the development.

Cloud computing has introduced a large number of different use cases for building software systems in the cloud. There are many options to choose from when the system architecture model is selected. Currently, more and more software systems are migrating to the cloud environment. This thesis tries to analyze the architecture evaluation methods that can be used to evaluate the architecture in the cloud and provides an example usage on one of the methods.

Academic motivation of the thesis is to research architecture evaluation methods in the cloud environment. The motivation of the thesis for the related company - Lixani - is to conduct an architecture review for the application that the company is developing, Lixani 5. In addition, a general documentation of the software architecture will be provided in the thesis.

Research questions of the thesis will be following:

- RQ1: How the cloud software architecture can be evaluated?

- RQ2: How efficient are the software architecture evaluation tools provided by cloud companies? Are they more beneficial than the traditional evaluation models?

The research methods used in the thesis are literature review, case study, and interview. Literature review is conducted to gain information about cloud computing in general, software architecture models, and software architecture evaluation methods. A case study is implemented to provide a real-life example of architecture evaluation for an example system. Interview is the third research method of the thesis. Three IT-professionals are interviewed with different questions related to the architecture evaluation. The interview sessions were discussions that loosely followed the pre-determined structure.

The thesis is structured in a following way. Chapter two focuses on cloud computing in general. It uses the existing literature about the subject to provide proper knowledge. The definition, characteristics and main elements of cloud computing are introduced.

Chapter three introduces four different software architecture models: N-tier, Serverless, Micoservices, and Monolithic. The architectures are considered using a literature review of the subject. For every model, an example architecture will be proposed including the suggestions on which cloud services can be used implementing them.

Chapter four considers three different architecture evaluation methods: Architecture Tradeoff Analysis Method (ATAM), Cost-Benefit Analysis Method (CBAM), and AWS Well-Architected framework. The literature review will be conducted to gain information about the methods. All the evaluation methods are introduced and analyzed in a general way.

Chapter five focuses on the case study. The company - Lixani - is first introduced and information about the case study application is produced. Chapter will include a

brief overview of the software architecture and its main elements. After introducing the software, the actual architecture review will be conducted, and its results are examined. The interview section of the thesis follows the architecture review. During the section, the general information about the interviewees is introduced and the interview questions and the answers by the interviewees are presented. The interview section consists of four questions related to the architectural evaluation.

Chapter six introduces the results of the case study. It is done by evaluating the results of the literature review and the case study. The chapter analyzes the conclusions of the architectural review done in the chapter five and provides conclusions regarding the interview part of the thesis.

Chapter seven is the general conclusion part of the thesis. The chapter analyzes generally what has been done during the thesis, the research questions, and the answers to the research questions are introduced. Also, the chapter will provide future study suggestions about the subject.

2 Cloud Computing

This chapter is a general introduction to cloud computing. First, the definition and characteristics of cloud computing are discussed. Then, different levels of cloud computing are introduced and, finally, the cloud computing deployment models are discussed.

2.1 Definition and Characteristics of Cloud Computing

Traditionally, companies have had their own IT infrastructure as physical servers that have been acting as a platform for their applications and data storage. Simply, cloud computing means that the IT infrastructure, platform or application is rented from a cloud provider such as Google Cloud, Amazon Web Services or Microsoft Azure. Cloud computing offers a way to quickly launch new services or instantly scale the resources that are already in use. [1]

Cloud computing refers to providing, managing, and provisioning IT infrastructure, platform, and applications as services over the internet. In cloud computing, the users have an instant access via internet to a shared pool of IT resources. Cloud computing is a constantly evolving subject that have different definitions. National institute of standards and technology (NIST) definition of cloud computing is "A model for enabling ubiquitous, convenient, on-demand network access to a shared

Figure 2.1: Cloud computing categories

pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [2]. Another definition for cloud computing is "By using virtualized computing and storage resources and modern web technologies, cloud computing provides scalable, network-centric, abstracted IT infrastructures, platforms and applications as on-demand services. These services are billed on a usage basis." [3]. As Figure 2.1 displays, the cloud computing can be divided into three main sections: infrastructure as a service, platform as a service, and software as a service. All these sections are discussed later in this chapter.

Cloud computing has several central principles that should come true to define a service as cloud computing. NIST defines a list of five essential cloud characteristics [2].

- **On-demand self-service:** Refers to the instant access to the resources as a self service. Deploying, managing, and monitoring resources are available for customers without any human interaction with the provider.

- **Broad network access:** The resources are available over the network from

anywhere, at any time. Access is provided using standard mechanisms and client platforms.

- **Resource pooling:** Cloud computing uses a multi-tenant model, where the resources are pooled to enable simultaneous service provision for multiple users. While using the multi-tenancy, the actual resources are still scaled to the demand of individual users. The client has information about the area where the resources are stored (country, city) but no control or knowledge about the specific location. [3]

- **Rapid elasticity:** Cloud computing resources are elastic and therefore rapidly scalable up or down depending on the needs of the consumer. Launching of resources can be done rapidly in any quantities and for the consumer, the cloud computing resources appear to be unlimited. [2]

- **Measured service:** Resources are monitored and measured accurately in different ways such as storage usage, processing capacity, and bandwidth. For the consumer, this means a chance to optimize the resource usage. Provider uses measuring to provide billing since the cloud resources are often billed on a usage basis. Also, the active measurement provides transparency for both parties.

Virtualization

In cloud computing, abstracting underlying resources and simplifying their usage is called virtualization or resource virtualization. Simply, it means that multiple extracted virtual or artificial resources can be living in one physical resource such as computer or server. It is a central part of cloud computing. Virtualization creates a layer of abstraction between application and the underlying resources. [4] In virtualization, the virtual resource and its state can be saved and migrated to

another server. Virtualization simplifies the usage and replication of resources, and keeps users isolated from each other. Thus, it provides elasticity to cloud computing services. [1]

2.2 Cloud Computing Service Types

Cloud computing can be divided into three main service types: infrastructure as a service, platform as a service, and software as a service. All of these types include a variety of different typical services in the cloud environment that are introduced in this section.

2.2.1 Infrastructure as a Service (IaaS)

Infrastructure as a service (IaaS) is the lowest level of cloud computing. In IaaS, the user gets an abstracted view of hardware such as computers, networks, storage systems, databases etc. For example, by using AWS Elastic Compute Cloud (EC2), the user can deploy a server with a specific operating system, a given number of CPU cores, and almost any amount of memory and storage. Access to these resources is given over the internet using a command line or a web user interface. Resources can be monitored and managed using these interfaces. Launching, starting, stopping, and rebooting services can be quick operations when using the IaaS. Also, scaling required capacities and defining network topologies can be done using the interfaces. The base thought of IaaS is that the user has no access to control the actual infrastructure but manages the amount of resources, operating systems, memory, storage and a limited amount of networking components [2]. Some examples of IaaS services are AWS EC2 (servers), AWS S3 (mass storage), Dropbox (mass storage), Azure Virtual Machines (servers), and Google Cloud SQL (databases). [3]

2.2.2 Platform as a Service (PaaS)

Platform as a service (PaaS) is the mid-level of cloud computing. PaaS resources are targeted to application developers by providing programming environments and execution environments. Applications can be created using programming languages, libraries, services, and tools on top of these environments. An example of a programming environment is Django Framework that is a framework that extends the Python programming language. Execution environment such as Google App Engine is then used as a platform to run the created application. [3]

The advantage of PaaS is that the application developers can focus more on the actual building of application and they do not have to concern the underlying infrastructure since they have very little access to modify it. The consumer has still some possibilities to affect the infrastructure such as modifying configuration settings for the application hosting environment. In addition to Google App Engine, some other PaaS resources include Heroku and AWS Beanstalk. What all of PaaS services have in common is that they offer a very straightforward and short path from developing the application to finally deploying it to internet. After the deployment of the application, the cloud providers offer ways to monitor the application and scale it based on the needs of the consumer.

2.2.3 Software as a Service (SaaS)

Software as a Service (SaaS) focuses on the end users of cloud applications. SaaS services can be seen as web applications that are built and operated on top of the PaaS and IaaS resources. [3] Applications can be accessed simultaneously from multiple sessions and they can be running on the same or different hardware resources. [4] The consumer has no control on any part of the cloud infrastructure or application capabilities. SaaS applications are available from different client devices through a web browser or a program interface. SaaS services are always running the recent

version and thus, the client doesn't have to buy new versions since SaaS products are mostly charged using a subscription model. SaaS applications vary a lot from simple applications to complex systems. Common SaaS applications are Microsoft Office 365 (Word, Excel, Teams, Outlook) and Google web product family (Gmail, Drive, Docs, Sheets). Other example is Customer Relationship Management (CRM) system provided by Salesforce.

2.3 Cloud Computing Deployment Models

Cloud computing deployment models are categorized based on the type of their environment. The types of cloud computing users can also be explained from these different categories.

- **Public cloud:** The cloud resources can be used by anyone. The cloud is usually accessible in a self-service manner using a web portal. In public cloud, the actual cloud infrastructure is located in the cloud providers' premises. Examples of major cloud providers are Amazon Web Services, Microsoft Azure, and Google Cloud.

- **Private cloud:** The cloud infrastructure is in the use of single organization. It can be seen similar to a company's intranet since the services are provided internally. [4] Often a private cloud is implemented for security reasons when a company wants to have a full control of their critical data and functions.

- **Hybrid cloud:** The cloud infrastructure is a combination of cloud infrastructures. The different cloud entities are tied together with proprietary methods. [2] An example of hybrid cloud can be the following: An organization is handling its certain functionalities in the public cloud whereas the critical operations are handled in the private cloud.

- **Community cloud:** The cloud infrastructure is provided for exclusive use of a community. The cloud resources are provided by one or multiple organizations inside the community or by a third party. An example of a community cloud is a large housing complex that provides cloud services for its residents. [4]

- **Virtual private cloud (VPC):** Virtual private cloud is a simulation of private cloud inside public cloud infrastructure. An example of VPC is a health organization providing private services to its users but hosting their system and data in the public cloud provider. The VPC resources are usually physically isolated to obtain privacy. [4]

3 Cloud Architecture Models

This chapter focuses on software architecture models especially in the cloud environment. The concept of software architecture is defined and four different software architecture models are discussed. The architecture models discussed are N-tier architecture, serverless architecture, microservices architecture and monolithic architecture. Also, one reference architecture illustration is given for each of the architecture models.

Software architecture can be defined in many ways. ISO/IEC/IEEE 42010 standard definition for software architecture is: "Fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution." [5]. Software architecture gives an overall description of the choices made in the architectural process. Architecture is an overlay on the fundamental concepts and stakeholders of the software.

3.1 N-Tier Architecture

One commonly used architecture model in modern software applications is N-tier architecture. N-tier architecture consists of multiple tiers that communicate with each other. Tiers are organized hierarchically and every tier has two main tasks. They offer services for the hierarchically upper level and requests services from the lower level. [6] Common example of N-tier architecture model is 3-tier architecture. The three tiers are presentation tier, business logic tier, and data access tier. Pre-

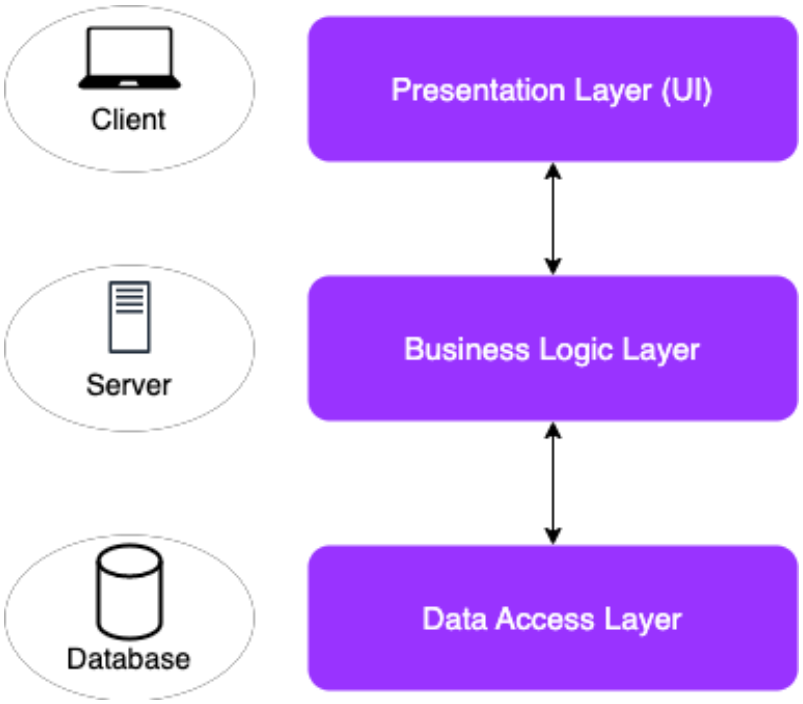

Figure 3.1: N-tier architecture example

sentation tier provides the client for the user whereas business logic tier is acting as application server and the data access tier is providing database access for the application server. When systems have become more complex, also the architectures have evolved. It is common to add multiple tiers between these three tiers to handle different tasks. [7]

In 3-tier architecture model, the tiers are separate units running usually on their own machines. The presentation tiers' main activity is to provide the user interface (UI) for the software user. It is the visible part of the system for the user. The presentation tier communicates with the business logic tier over the internet using HTTP requests. The business logic tier handles the coordination of the software and handles all the logical tasks. When the business logic tier needs services from the data access tier, the requests to the tier are done often by using HTTP requests. The data access tier simply store and retrieve data from the database and provides data to the upper tiers.

3.1.1 Architecture Example

An example architecture for the N-tier model can be seen in Figure 3.1. The example architecture is a high-level illustration of architecture that could be used, for example, for a simple web application. The described architecture is using a 3-tier implementation. The presentation tier is providing the user interface, the business logic tier is providing logical features, and the data access tier is providing data storing and retrieval.

The example architecture could be implemented, for example, in the AWS cloud environment by choosing commonly used technologies for this kind of model. The user interface could be deployed in AWS CloudFront[1] that offers services for content delivery. The business logic tier could be served in AWS EC2[2] server instance. The implementation for data access tier could be implemented to the AWS Relational Database Service (RDS) [3].

3.1.2 Advantages and Disadvantages

The N-tier architecture has its basis in separating the different architectural parts from each other. This comes with advantages and disadvantages. The advantages that the N-tier architecture model provides are related to the scalability, security, manageability, and flexibility. The flexibility and manageability are results of the tier model. All tiers can be changed and monitored independently. The possibility to add more tiers is advantage related to the flexibility. Scalability is another result from the independent tiers. One tier can be scaled to use much more hardware resources if it becomes a bottleneck and this can be done easily especially in the cloud environment. Security is enhanced when there are multiple tiers separating

[1]AWS CloudFront: https://aws.amazon.com/cloudfront/

[2]AWS EC2: https://aws.amazon.com/ec2/

[3]AWS RDS: https://aws.amazon.com/rds/

the application critical components from each other. Also, since the data is multiple tiers away from the presentation tier, the data is more protected. [7]

The N-tier architecture model comes also with its disadvantages considering, for example, performance and complexity. The N-tier architecture divides the system into multiple tiers. The communication between multiple tiers can provide higher latency that can affect negatively to the performance of the system. The more tiers are added the more it may affect on the latency and provide complexity. [6]

3.2 Serverless Architecture

Serverless is a cloud computing architecture model where the application logic is implemented using several different functions. It is also known as Function as a Service (FaaS). The functions are stateless, independent implementations that are launched when they are needed. Functions are hosted by a third-party cloud operator that provides the needed hardware resources and autoscaling opportunities for functions to be launched at any time. Thus, the serverless developers can focus on the application developing, rather than using time to manage server software and hardware. The serverless architecture is often implemented using a HTTP server launching different functions that handle the application logic. Inside the application workflow, the functions can invoke and communicate with each other. Serverless implementations are often combinations of big number of small functions where each of them has their own specific job. Also, given the functions' single execution principle, the functions are easy to be triggered, for example, as a cron job. Other methods to trigger functions are when the file is added to storage service, a change in cloud database, or an item entering the message system. [8] Regarding the name "Serverless", the architecture actually contains a HTTP server but unlike in regular web-server architecture, in serverless, the HTTP server acts as a middleware for launching different application logic functions as their own services. [9]

Serverless cloud function principle offers the developers a way to add application logic as new functions without the need to touch the functions that are already in use. This feature leads to looser coupling and lower rigidity in code. Serverless computing is using a pay-as-you-go pricing model. Consumers are charged based on the cloud resource usage of individual functions. CPU time and allocated memory are parameters that are usually the base of pricing. The cloud provider handles the scaling of resources. The automatic scaling is based on the invocations of functions. If there is a lot of invocation traffic for functions, the number of instances of these functions on demand is increased and vice versa. [9]

There are different types of applications that benefit from the serverless architecture model. Due to the pay-as-you-go pricing model serverless is often implemented in smaller applications. In these occasions the user only pays for the time there are application users using the software. Due to the pricing model and automatic scaling, applications that have often small usage but occasional usage bursts benefit from the model. The serverless model is not a best solution for applications that handle large files or websockets since the functions have a maximum amount of time that they can run after a single invoke.

3.2.1 Architecture Example

The example of a simple serverless architecture model can be seen in Figure 3.2. The example proposes an overall architecture overlay for simple note writing application. The user interface accesses the application logic using the HTTP server. Application logic contains functions for creation, reading, updating, and deleting (CRUD) a note. Functions then interact with the database. The example is lacking the usual serverless methods where the functions invoke and interact with each other.

The architecture example could be implemented to any cloud providers environment. Using AWS as an example, the following services could be in use: API

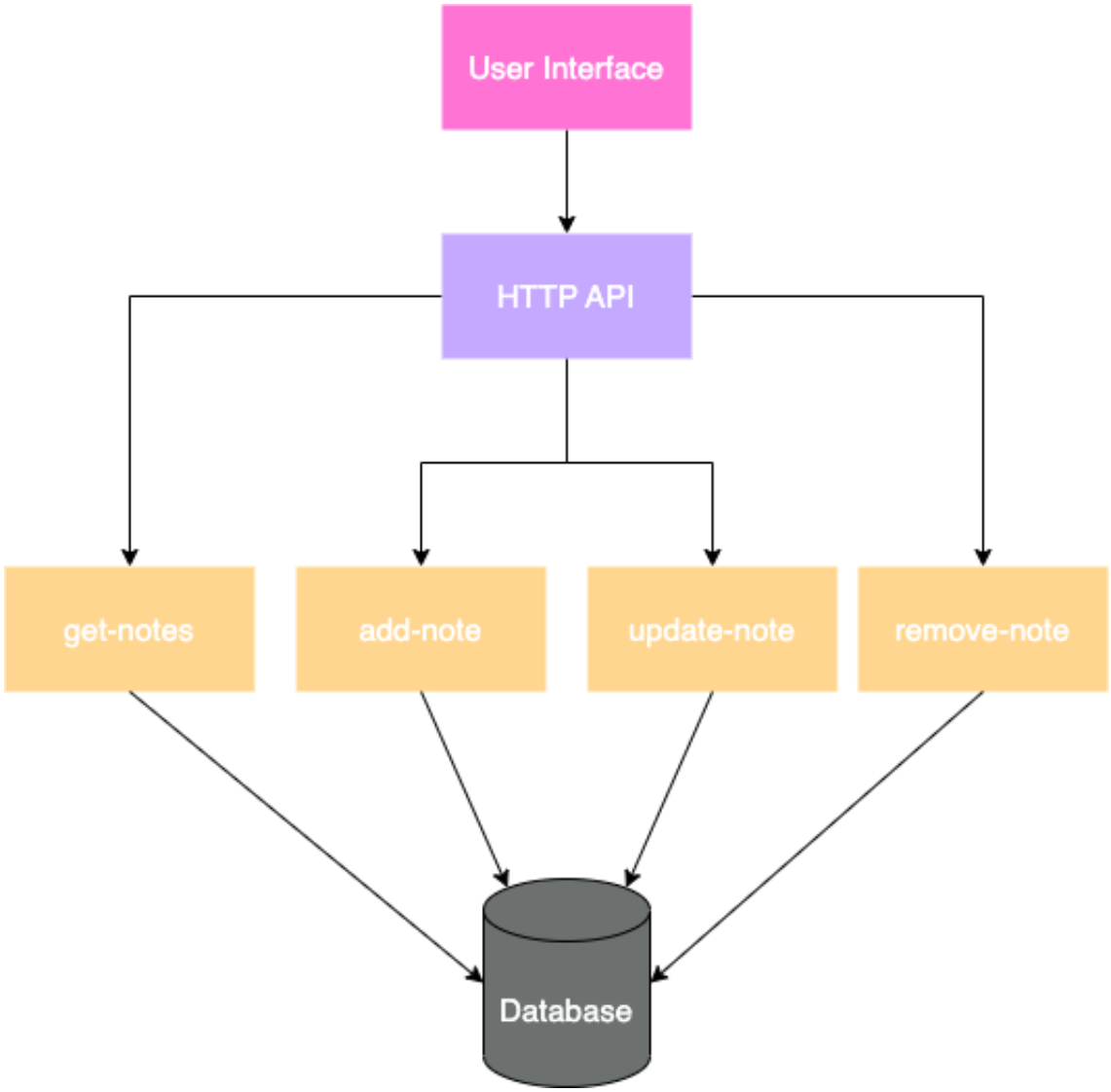

Figure 3.2: Simple serverless architecture

Gateway [4] as a HTTP server which invokes Lambdas [5] that are the logic functions. The Lambdas would then interact with database such as relational database service (RDS) [6] or key-value database service (DynamoDB) [7]. For logging purposes, CloudWatch [8] would be the solution. [10]

3.2.2 Advantages and Disadvantages

Serverless architectures can be developed to be cost-effective with their pay-as-you-go billing model. AWS Lambda service pricing consists of the number of requests and the resource usage in CPU time and used memory. The functions are single implementations with a responsibility for a one specific action. The application structure regarding the permissions to interact with the application database and storage is taken care of efficiently. Every function has a set of permissions that are specifically defined. The security can be handled efficiently using this feature.

Serverless developers can fully focus on developing the application logic since the cloud infrastructure is taken care of by the cloud provider. The cloud provider offers an execution environment where the functions can be run. Typically, the environment supports a limited number of programming languages, but still containing the mainstream ones such as Node.js, Python, Java, and C#. [11]–[13] Also, the autoscaling of the resources helps developers to focus on developing instead of maintenance or launching the infrastructure.

The nature of serverless functions is that they are running only when they are invoked. When there is little traffic, cloud providers close down the instances that are not in use as part of the autoscaling. One well-known bottleneck in serverless

[4]AWS API Gateway: https://aws.amazon.com/api-gateway/

[5]AWS Lambda: https://aws.amazon.com/lambda/

[6]AWS RDS: https://aws.amazon.com/rds/

[7]AWS DynamoDB: https://aws.amazon.com/dynamodb/

[8]AWS CloudWatch: https://aws.amazon.com/cloudwatch/

functions is cold start delay. It can be defined as high start-up latency during the first invocation after a period of inactivity. [9] The first invocation after a period of time is time consuming since the runtime have to be prepared for the needs of the function. For example, the external libraries must be installed. Naturally, when the function is invoked frequently, the start-up latency is low. Application developers need to consider the cold start delays as part of the serverless development. This is developers' responsibility since the cold starts are part of the cloud providers resource autoscaling. Also, the delay result of cold starts is even bigger when the cold started function invokes another function that is in idle mode and has to be cold started.

Another feature of the serverless functions is that the functions are stateless meaning that they do not hold the state or share the state between other functions. [9] The lack of universal state across the functions makes the communication between different functions less efficient. The lack of shared state between functions forces the invoked function to fetch or produce all the needed data if it is not being passed forward from the invoker function.

3.3 Microservices Architecture

A microservice is a small autonomous service that has a single responsibility and can be deployed, scaled, managed, and tested independently. [14] The microservices architecture model consists of multiple independent microservices that are tied together. The microservices architecture offers application developers a method to build a bigger system in separate teams that are responsible for different microservices.

Typically, a microservice has one responsibility from the business logic point of view. Microservices are independent services that communicate with each other using lightweight techniques such as application program interfaces (API). Microser-

vices can be developed, deployed, scaled, managed, and tested autonomously. Scalability provides an opportunity for applications to grow rapidly. Since microservices are independent systems, the developers can use different programming languages and technologies inside of one bigger system. That provides flexibility for the development. [15]

Infrastructure automation is a key part of microservices architecture. A company using microservices need an automated continuous integration and continuous deployment (CI and CD) pipeline to produce microservices effectively. Pipelines can produce microservices from templates and deploy them automatically. Automated testing is also key part of microservices. The developers have to make sure that each individual microservices are working properly after they have been changed. [15]

In microservices, the cloud resource management is handled by the cloud provider. Developers can target their resources to the developing part – as in serverless architecture. The examples of cloud platforms for the microservices are AWS Elastic Container Service (ECS) and Microsoft Azure Kubernetes Service (AKS). [16], [17] Resource scaling is provided by the cloud provider.

Microservices architecture is used in bigger applications that are capable of splitting into small services that handle different parts of business logic. One certain use case for microservices is the migration from monolithic architecture to a fully scalable cloud architecture. Application logic can be rebuilt piece by piece without compromising the parts that are already in use. Microservices can be a natural direction to move for systems that have become too big to manage as a monolithic architecture. When the system is complex enough, it is hard to be changed without compromising all available features. [14]

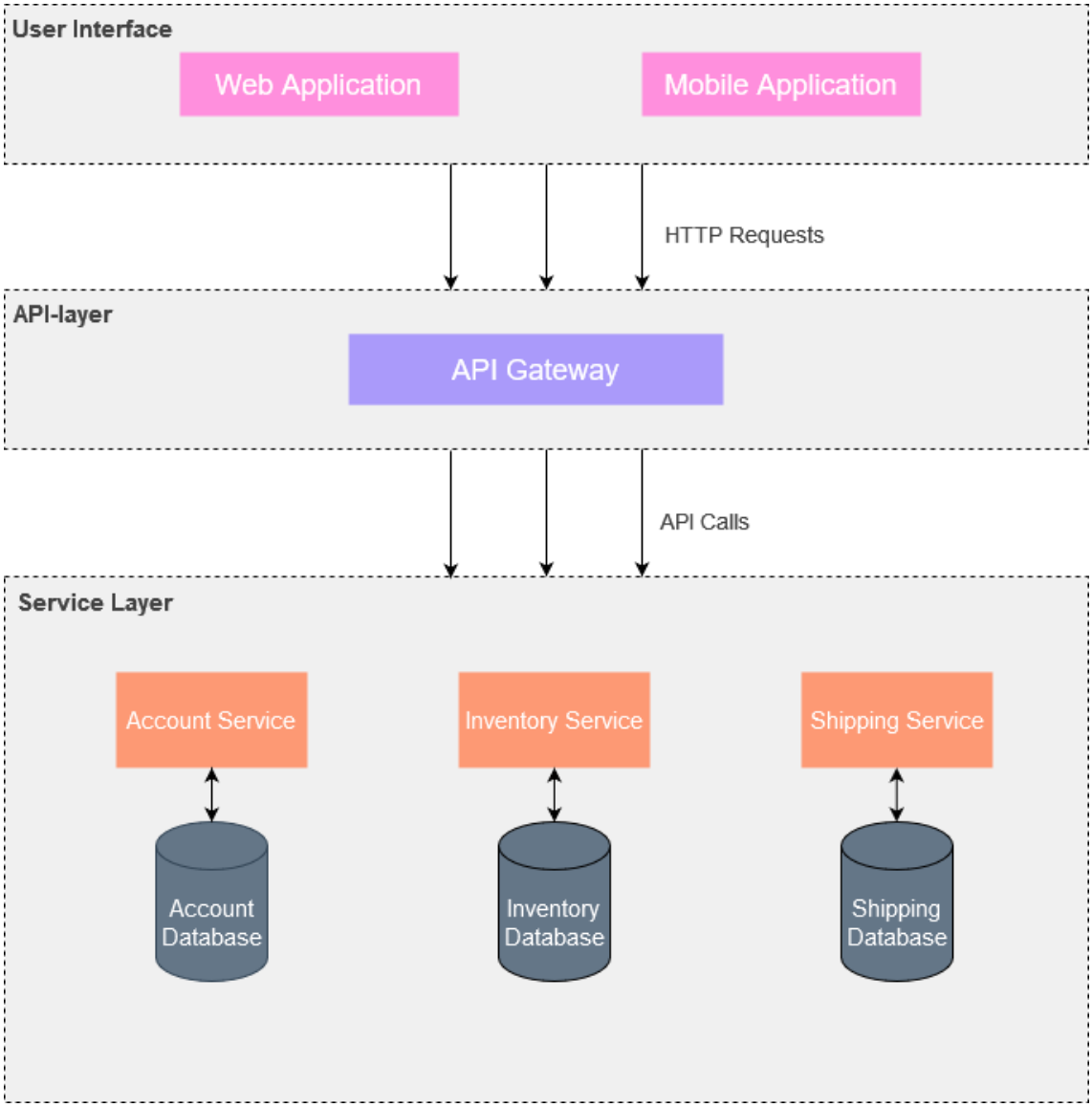

Figure 3.3: Microservices architecture example

3.3.1 Architecture Example

The example of a microservices architecture model can be seen in Figure 3.3. The example is an architectural suggestion for e-commerce application. It consists of two user interfaces and three different microservices that handle the business logic. User interfaces are mobile application and browser web application. Microservices consist of an account service, inventory service, and shipping service. All three microservices interact with their own databases. All of the independent services have a single responsibility which is one of the central principles in the microservices architecture model. This principle is based on the Unix philosophy on doing one thing and doing it well. [18]

Following cloud services could be used if the microservices architecture was conducted in the Microsoft Azure cloud environment. First, the Azure Kubernetes Services [9] can be used as containers for the microservices. The Azure API Management [10] could be used as API service for building the communication between the mobile application and the microservices. Finally, Data storage can be implemented with the use of Azure SQL Database [11].

3.3.2 Advantages and Challenges

The agility of the microservices is an advantage. They are independent implementations that can be deployed without redeploying the entire application. Microservices can be developed as their own units with and it can help to reduce the time to market length of different features. [19]

Microservices are easy to be scaled. Since each business logic responsibility is their own service, the more needed services can be scaled to use more resources than

[9] Azure KBS: https://azure.microsoft.com/en-us/products/kubernetes-service

[10] Azure API Management: https://azure.microsoft.com/en-us/products/api-management

[11] Azure SQL Database: https://azure.microsoft.com/en-us/products/azure-sql/database/

the others. Traditionally, the whole application should have been scaled whereas in microservices, only the specific services can be scaled. The technological flexibility and small code base are also advantages. Individual microservices can be developed using the technologies and development languages that fits best for services purposes. The relatively small code base of each service in contrast to the monolithic architecture can make the development of new features to be easier. [20]

The complexity of the microservices architecture is a challenge. Each microservice is independent and relatively small unit but the system containing multiple microservices is complex as a whole. Complexity has an effect on the data management and consistency of the microservices architecture. This comes from its distributed nature. Often, each service has their own database implementation and the data management is decentralized. Thus, the management of distributed transactions is difficult. [15] Handling the security is vital part of any software architecture. Every microservice inside the architecture exposes an individual entry point for external or internal communication. [15]

Usually, the microservices are small implementations and require communication with other microservices. The network latency and congestion can become a problem in the microservices architecture. Microservices are their own instances and the communication between them is done over the internet. The requirement of communication can cause high latencies and has to be considered during the application development. Latency can be even higher if the services are chaining the requests, for example, if microservice X calls microservice Y that calls microservice Z. [20]

3.4 Monolithic Architecture

Monolithic architecture is the traditional model for building applications. Monolithic architecture is a single unit that contains the whole system. The architecture consists of three different layers: the user interface, server-side business logic, and

database. The user interface is the browser or mobile app that the user interacts with. Server-side business logic handles all the logical functions that the application contains. Business logic layer interacts with the database. A single database contains all the application data. [21] The monolithic architecture is single logical executable and every deployment refreshes the whole application to a new version. [22]

Monolithic architecture consists of a single codebase. Any code change done by the application developer can affect the whole system since all the changes are done on the same codebase. [19] This emphasizes the importance of accurate testing.

Monolithic architecture can be deployed to cloud environment with virtual machines. Examples of cloud virtual machine providers are Amazon Elastic Compute Cloud (Amazon EC2) and Google Cloud Compute Engine. [23], [24] In monolithic architecture, the service developers have more ability to affect the underlying cloud resources. The cloud provider offers the infrastructure to the system based on the resources the client needs. For example, the AWS EC2 service provides a big number of different virtual machine instance types that vary in terms of CPU, memory, storage, and networking capacity.

3.4.1 Architecture Example

Simple monolithic architecture illustration can be seen in Figure 3.4. The example uses typical three-layer implementation of monolithic architecture. The user interface and business logic are located inside the same entity. Then, the business logic layer communicates with database to implement the data operations.

Monolithic architecture could be implemented with following cloud technologies in the AWS environment. An Elastic Compute Cloud (EC2) [12] instance could be deployed as Ubuntu server. The instance could include both application user inter-

[12]AWS EC2: https://aws.amazon.com/ec2/

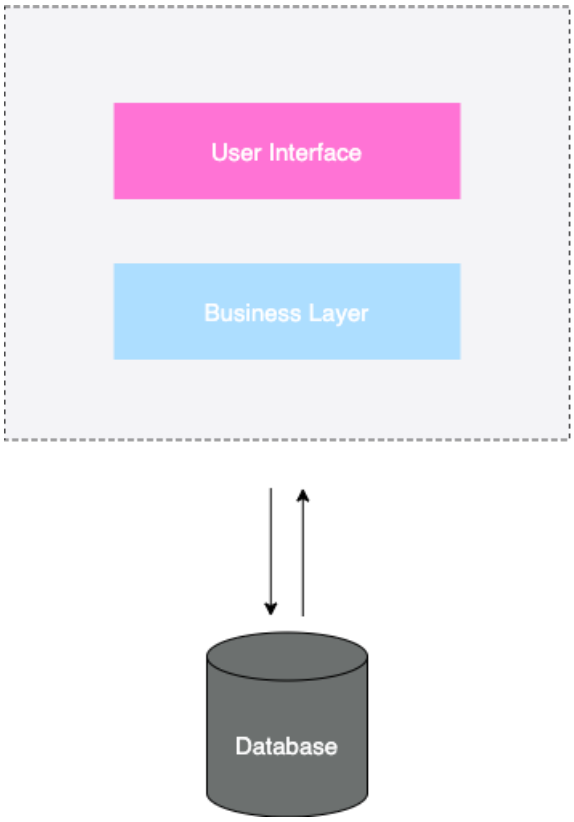

Figure 3.4: Monolithic architecture example

face and business logic as a Django framework system [13]. The used database service could be AWS Relational Database Service (RDS) [14].

3.4.2 Advantages and Challenges

Simplicity is one advantage of monolithic architecture. It is a viable solution for a small application or MVP. Monolithic applications tend to be easier to test, deploy, debug, and monitor than more complex architectures such as microservices. Also, the data is centralized in the single database and it does not have to be synchronized with other services. [22] When building a lightweight application with a small number of users monolithic architecture is a faster solution than a similar microservice one. [25]

Another advantage is the amount of configurations required in the monolithic environment. The amount of configurations can be smaller since there is only one environment that has to be configured. Also, the configurations can be more straightforward. [26] For example, microservices require configurations for their service deployment templates and possibly for each services individually.

When the application grows to be more complex the monolithic architecture encounters challenges. Scalability is not very efficient in the monolithic architecture. Using the microservices or serverless architecture, the specific services that needs to be scaled, for example, because of usage traffic can be scaled as their own services. When the monolithic application needs to be scaled to use more or less resources, the whole system has to be scaled. This might result in a waste of resources and can produce unnecessary costs. [19]

The complexity is also a problem in a code base. When the code base is large, the time needed to build and deploy the system becomes longer and eventually this

[13]Django Framework: https://www.djangoproject.com/

[14]AWS RDS: https://aws.amazon.com/rds/

can make the development processes slow. The application development becomes harder when the code base grows. Since its undistributed nature, the changes in one module can produce unexpected behaviours in many other modules. Applying new technologies becomes a challenge in a monolithic architecture. The whole system has to be upgraded to use the technology instead of just one service. [21] The difficulties in adopting new technologies can result in technical debt in the long run.

4 Architecture Evaluation

This chapter gives an overlay to software architecture evaluation in general. After the general overview, three different architecture evaluation models are introduced. The introduction is given to two traditional and one cloud-specific architecture evaluation models. The models are architecture trade-off analysis method, cost-benefit analysis method and AWS Well-Architected framework.

Software architecture has numerous different definitions as introduced earlier in the Chapter 3. It can be also defined as a sum of different architectural decisions. Architecture evaluation often evaluates these specific decisions. Software architecture evaluation is performed to identify the quality of architectural decisions made designing the overall structure of the software and to determine risks. Architecture evaluation can be done in any phases of the software developing including the start of the architecture design, development phase, or the maintenance phase. In the start of the architecture design, evaluation can be done by comparing the chosen architecture style to other ones and to discuss trade-offs that the current style could possibly be producing. In the development phase the evaluation can be done, for example, by evaluating how well the designed properties are eventually built. Software architectures are being constantly more complex and active evaluation can be a solution to preserve quality and clarity of the systems. [27]

There are numerous ways to perform software architecture evaluation. Traditional methods are often human-centric discussions between different stakeholders

such as architects, developers, and managers. Often the customers are also part of the stakeholders participating the discussions. The discussions are made to evaluate how well, for example, architectural decisions, patterns, and technologies preserve the quality of software attributes. They also try to identify risks and consider trade-offs between different approaches. Being human-centric, traditional methods rely on judgement of different professionals. [28] Examples of traditional approaches are architecture trade-off analysis method (ATAM), cost-benefit analysis method (CBAM), and scenario-based architecture analysis method (SAAM).

While the complex applications are continuously shifting towards public cloud, cloud providers have come up with novel cloud-specific techniques to evaluate software architectures. These new methods include from both Amazon Web Services and Microsoft Azure a service called Well-Architected.

4.1 Architecture Tradeoff Analysis Method (ATAM)

The Architecture Tradeoff Analysis Method (ATAM) is a scenario-based traditional system architecture analysis method. ATAM tries to produce understanding for the consequences of different architectural decisions while simultaneously paying attention to the quality requirements of the software. ATAM is usually done in the early phases of the architecture design process but can also be done in later phases. [29]

The process for executing ATAM is divided into four phases: presentation, investigation and analysis, testing, and reporting. The process is illustrated in the Figure 4.1. During the presentation phase, ATAM is introduced to the stakeholders together with the business goals of the system and the system architecture. The presentation phase produces understanding of what is going to be analyzed, how it is done, and what is the motive for doing the analysis. In the investigation and analysis phase, the architect presents the architectural approaches, stakeholders produce

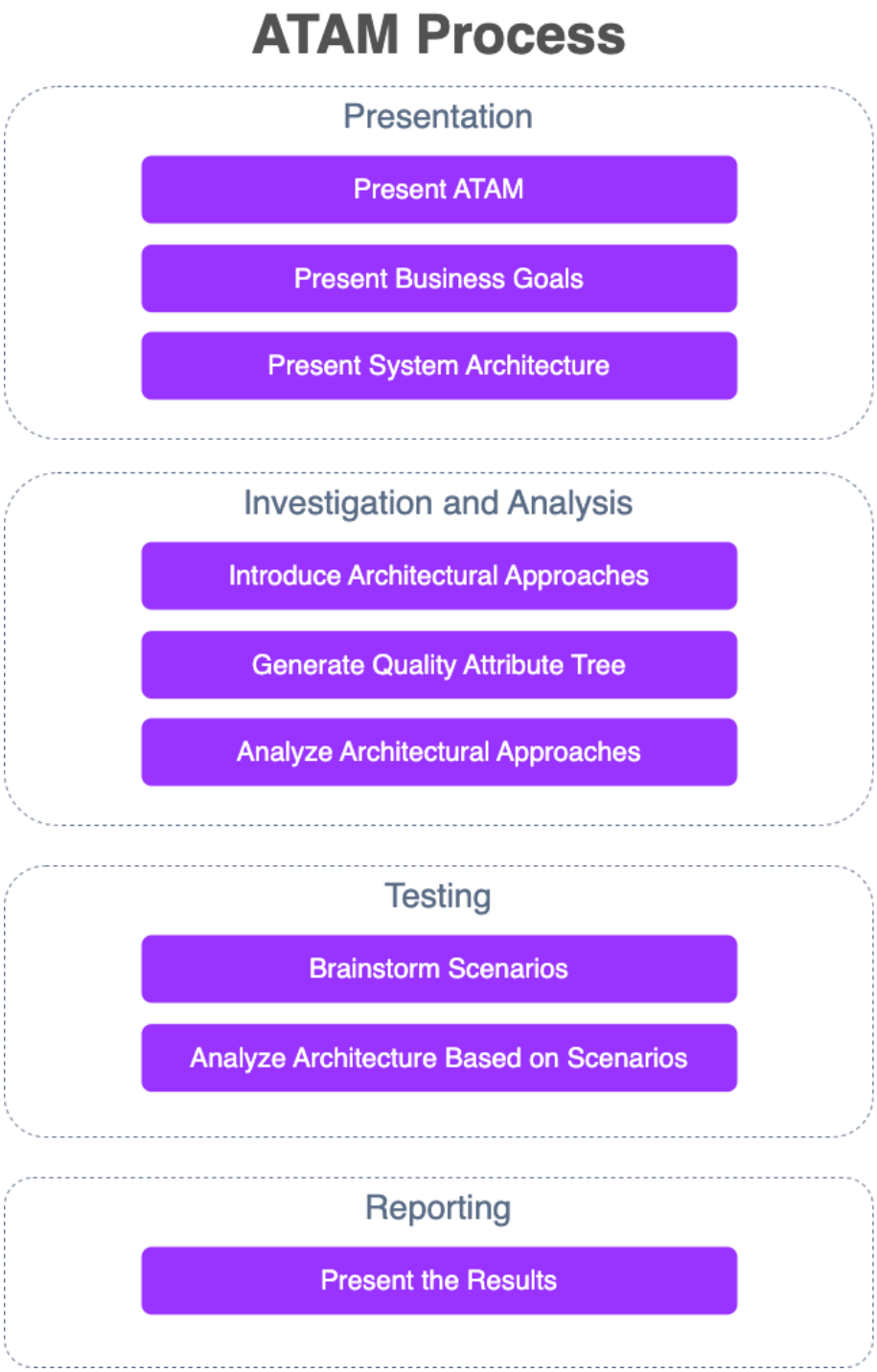

Figure 4.1: ATAM process

quality attribute tree, and the architectural approach is being analyzed. The quality attribute tree consists of different quality factors in the system, broken down into the level of scenarios. These scenarios are then prioritized. During the architecture analysis, the architectural approaches are being evaluated regarding the factors in the quality attribute tree. The testing phase consists of brainstorming more scenarios and analyzing the architecture based on the scenarios. Finally, the reporting phase presents the gathered information from earlier phases. The results consist of architectural approaches, tradeoffs, scenarios, quality attribute tree, risks, etc. The results can be documented to be used in the future. [30]

Analyzing system architecture using ATAM includes numerous benefits. The problems of the architecture can be noticed in the early phase. Architectural decisions are explicitly brought up and analyzed how well they align with the requirements. ATAM is human-centric and require active communication and contribution from experts of different areas. Thus, ATAM increases the communication between the stakeholders and provide clarity. Different ATAM phases and the results are documented and therefore the general knowledge about the system can be increased inside the organization. Also, well-documented process can help teams to perform ATAM in the future.

The issues in ATAM are related to its human-centricity. The process requires a lot of input from different stakeholders. If the stakeholders are not serious about the process, the results can have minor meaning compared to results of the process that is done thoroughly. The process requires active participation and preparation by the key stakeholders. [30] ATAM requires substantial amount of time from the key stakeholders to be implemented properly. Thus, ATAM can produce major costs.

4.2 Cost-Benefit Analysis Method (CBAM)

Cost-benefit analysis method (CBAM) is another traditional scenario-based architecture evaluation method. CBAM provides a cost-benefit analysis to help making architecture design decisions. CBAM offers economical perspective for designing process. It is a method that considers the best architecture design based on what potential costs and benefits the corresponding architecture provides. Costs and benefits provided by the architecture include, for example, effort, schedule, efficiency, and risks. As well as ATAM, CBAM is human-centric and requires quality input from different stakeholders. [31] The goal of CBAM is to produce a maximized difference between the received benefits from the system design and the costs that are required to implement the chosen design. [32]

Implementing the CBAM can be divided into two main phases: architectural strategy development and cost-benefit analysis. [31] The individual steps of the two phases can be seen in Figure 4.2. CBAM can be seen as economically extended version of ATAM and the similarities can be seen in the first phase. The architectural strategy development phase consists of creating and refining the scenarios and prioritizing them. After prioritizing, the quality attributes are described for the scenarios. Finally, the architectural strategies are being chosen or developed to serve the scenarios. Phase two, cost benefit analysis, consists of evaluating and comparing the chosen architectural strategies. Evaluation is being done by describing the utility value that the strategy produces, calculating the total costs and benefits of the strategy, and determining the return of investment (ROI) of the strategy. Eventually, the final architectural strategy is being chosen based on the evaluations. More accurate solutions can be achieved by performing another iteration of the two phases. In second iteration, the process can be extended by adding information about risk calculations, uncertainties and allocated technical resources. [32]

The use of CBAM as an architecture evaluation method includes numerous bene-

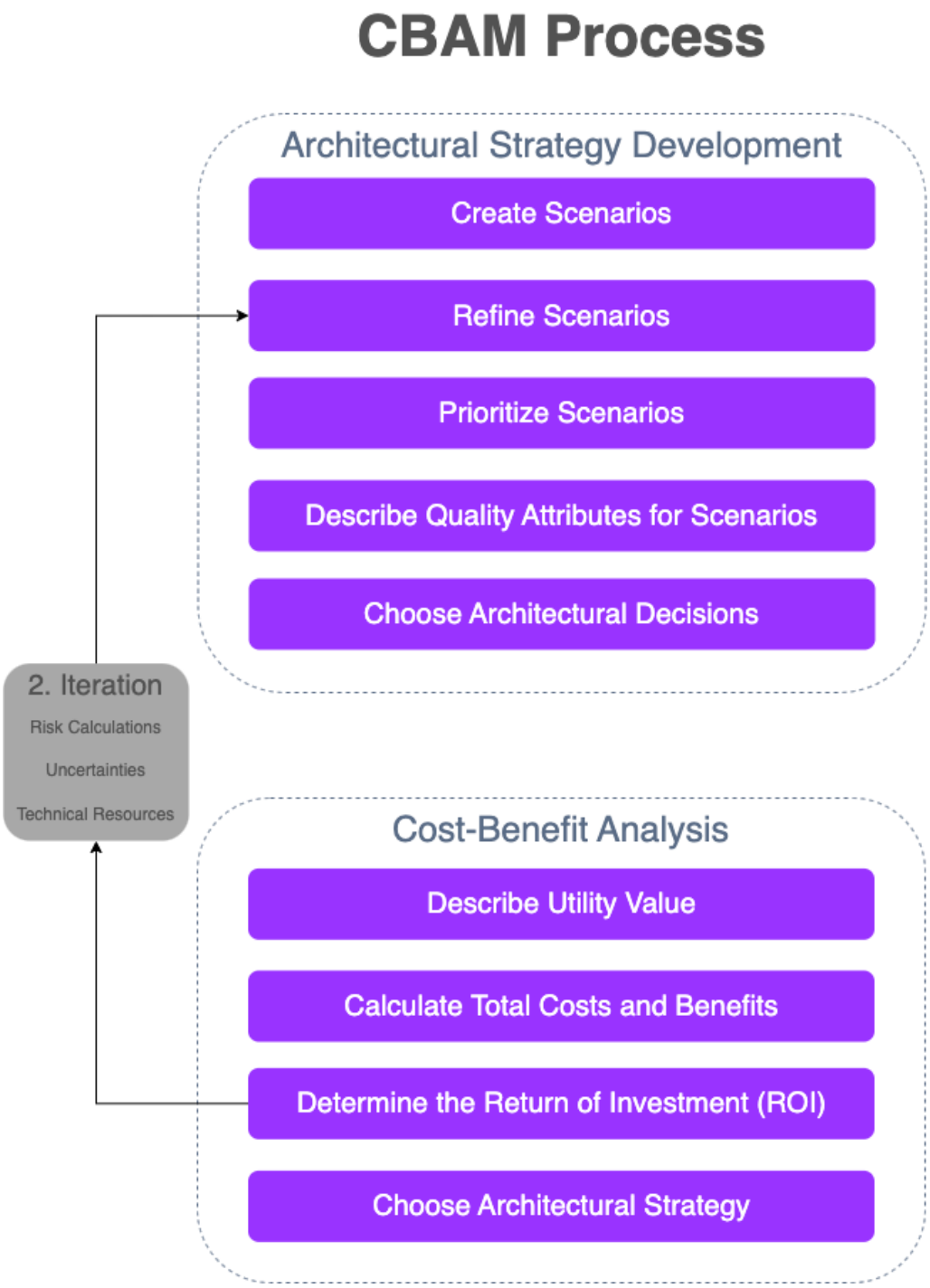

Figure 4.2: CBAM process

fits and uncertainties. The costs and benefits of architectural decisions are precisely determined which leaves less room for intuition or preconception. In CBAM, the ROI is calculated for the architectural decisions and, thus, the selected methods are not only chosen by relying on costs or benefits. [32] Also, CBAM produces clarity since it requires active communication between different stakeholders. Similar to ATAM, the CBAM demands serious effort from stakeholders to gain effective results since it is a human-centric method. Thus, the human-centricity can be seen as uncertainty. Another issue for CBAM is its incapability of effectively defining the added value of architectural strategies. This makes CBAM less efficient method for uncertain environments such as internet of things (IoT). [28]

4.3 AWS Well-Architected

The AWS Well-Architected framework is a cloud-specific evaluation tool to review and improve the system cloud architecture. The framework improves the knowledge of business impacts of the architectural decisions. It provides a way to measure the system architecture against best practices. The framework includes general design principles, best practices, and guidance. Well-Architected consists of six operational pillars - operational excellence, security, reliability, performance efficiency, cost optimization, and sustainability. AWS provides Well-Architected Tool which is a cloud service that follows the user through the architecture evaluation. The framework consists of questions about all the five pillars. Well-Architected Tool extends the evaluation to consider specific architectural style models such as serverless and microservices. These extensions are called lenses. The evaluation can be done by using only the general Well-Architected framework or by extending it with the architectural specific extensions. [33]

Pillar 1: Operational excellence

Well-Architecture documentation defines the operational excellence pillar as "The ability to support development and run workloads effectively, gain insight into their operations, and to continuously improve supporting processes and procedures to deliver business value." [34]. Workload means software components that produce business value as a group. In addition to evaluating the processes and procedures of the development, operational excellence also refers to the business goals of the system and how the organizational aspects support them. Operational excellence can be seen as a general overview of what the business goals are, understanding the software, operating the system to achieve business and customer outcomes, and constantly improving the procedures to evolve as a whole. [35] Operational excellence is a quality specification that might not have a counterpart, for example, in ATAM.

Pillar 2: Security

The security aspect of the framework considers how well the system protects its data, software components, and assets. It analyzes how efficiently the system takes advantage of the cloud specific security technologies. The security part covers methods such as identity management, traceability and security incidents. Identity management can be centralized to make it easier. Traceability is an efficient method to detect actions and changes in the system in real-time. The framework encourages users to have specific process in case of security incidents. As well as in operational excellence, automation is a key part of the security. Cost-efficiency and scalability improve when security mechanisms are automated. [36]

Pillar 3: Reliability

The reliability part of the framework evaluates the system's ability to perform efficiently and consistently the operations it is supposed to. Reliability includes failure

management as well as the ability to scale rapidly. A key point of reliability is the automatic recovery from failure states. This can be achieved through active monitoring and event triggers. By efficient scaling, the system keeps reachable to the users when the demand increases quickly. The scaling needs to be implemented also downwards to adapt to lower demand. Another key part of reliability is to develop system to consist of small individual resources to lower the impact of a single resource failure. [37]

Pillar 4: Performance efficiency

The performance efficiency pillar evaluates how efficiently the system utilizes the cloud computing resources. This pillar encourages the users to search and select suitable technologies for different use cases. Cloud technologies makes applying new technologies easier since many of them are easily deployable. The pillar suggests to constantly review the technologies that are in use in terms of performance and thus, benefit from the continuous innovation of cloud technologies. This part of the framework also considers how the cloud resource usage is monitored. [38]

Pillar 5: Cost optimization

The Well-Architecture documentation defines the cost optimization pillar as "The ability to run systems to deliver business value at the lowest price point". [34] Cost optimization consists of active measurement of the business value output and the costs associated with it. Cost optimization is key part of cloud computing since they are often charged with pay-as-you-go model and, thus, can be optimized based on the needs. [39]

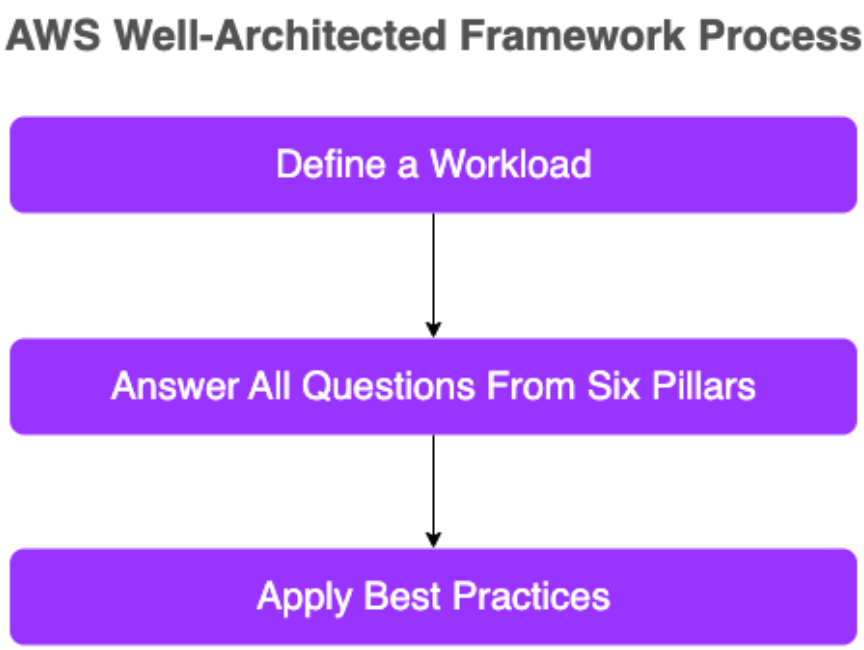

Figure 4.3: AWS Well-Architected Framework process

Pillar 6: Sustainability

The sustainability part reflects on the environmental impacts of different architecture decisions. It encourages the architecture designers to consider energy consumption and efficiency as part of the system development process. To achieve sustainability, it is important to understand the impact of users cloud resources, establish sustainability goals and maximize the utilization. [40]

Review process

The Well-Architected framework suggests the users to constantly re-examine the system architecture. A Well-Architected review can be done in any phase of the software development process. Simple illustration of the review process can be seen in figure 4.3 The review process should include the whole development team to increase the knowledge of the architecture and its attributes. The review is implemented using the questions about different pillars and it is suggested to be

more of a conversation between team members rather than an audit. The output of the review process can be seen as actions that improve the system as a whole. [41]

5 Case Study

5.1 Lixani Oy

Lixani Oy is part of the Lemonsoft Oyj concern. Lemonsoft Oyj is a Finnish publicly listed company that offers ERP-solutions for small and medium sized companies. Lemonsoft operates in different industries including manufacturing, logistics, accounting, and construction. The concern consists of a parent company and multiple subsidiaries including Lixani Oy. Lixani joined Lemonsoft in 2020.

Lixani Oy is a company that offers multiple products in construction industry. Its main product is an ERP-system called Lixani. The system is targeted for small and medium sized companies specialized in renovation construction and new construction. The system has been around for over eight years and during the years it has evolved from a text-message based system to a modern web application. This case study focuses on Lixani 5, a new version of the system released in 2022.

5.1.1 Lixani 5 Application

Lixani 5 is an ERP-system built for mobile and desktop use while focusing primarily on the mobile environment. Construction workers add their work records and report with mobile devices. Similarly, the construction supervisors monitor the sites using mobile devices. The system provides a wide selection of methods for monitoring the construction site accurately. Cost control allows users to monitor costs on a

denomination level. The workflow feature provides a way to wrap up records of all project stakeholders. The project bank is a centralized document store for all project related information. The information reporting feature allows companies to report project related employee listings and costs to Finnish Tax Administration, which is mandatory in the construction industry.

Lixani's vision is to improve transparency within construction industry. Often, construction sites consist of long subcontracting chains that provides an opportunity for cheating in different parts of the chain. Lixani aims to improve transparency by providing precise monitoring of the project's current state. A centralized store for all project data from day-to-day work of all stakeholders improves communication inside the project and provides cost savings. With real-time data, the accuracy of decision making becomes easier. Lixani also encourages the construction business to be more self-organized by allocating more responsibilities for construction workers. Anticipation and budgeting are also things that improve by active monitoring of projects' states.

5.1.2 Lixani 5 System Architecture

A simplified description of Lixani 5 system architecture can be seen in Figure 5.1. The architecture is a versatile entity that consists of many different AWS cloud services. The architecture follows the serverless architecture model. The architecture relies heavily on the Serverless Framework. It is a framework that aims to ease the development process of serverless architecture with AWS Lambdas and other AWS resources that are in relation with the Lambdas. [42] The main parts of the architecture will be introduced in the following sections.

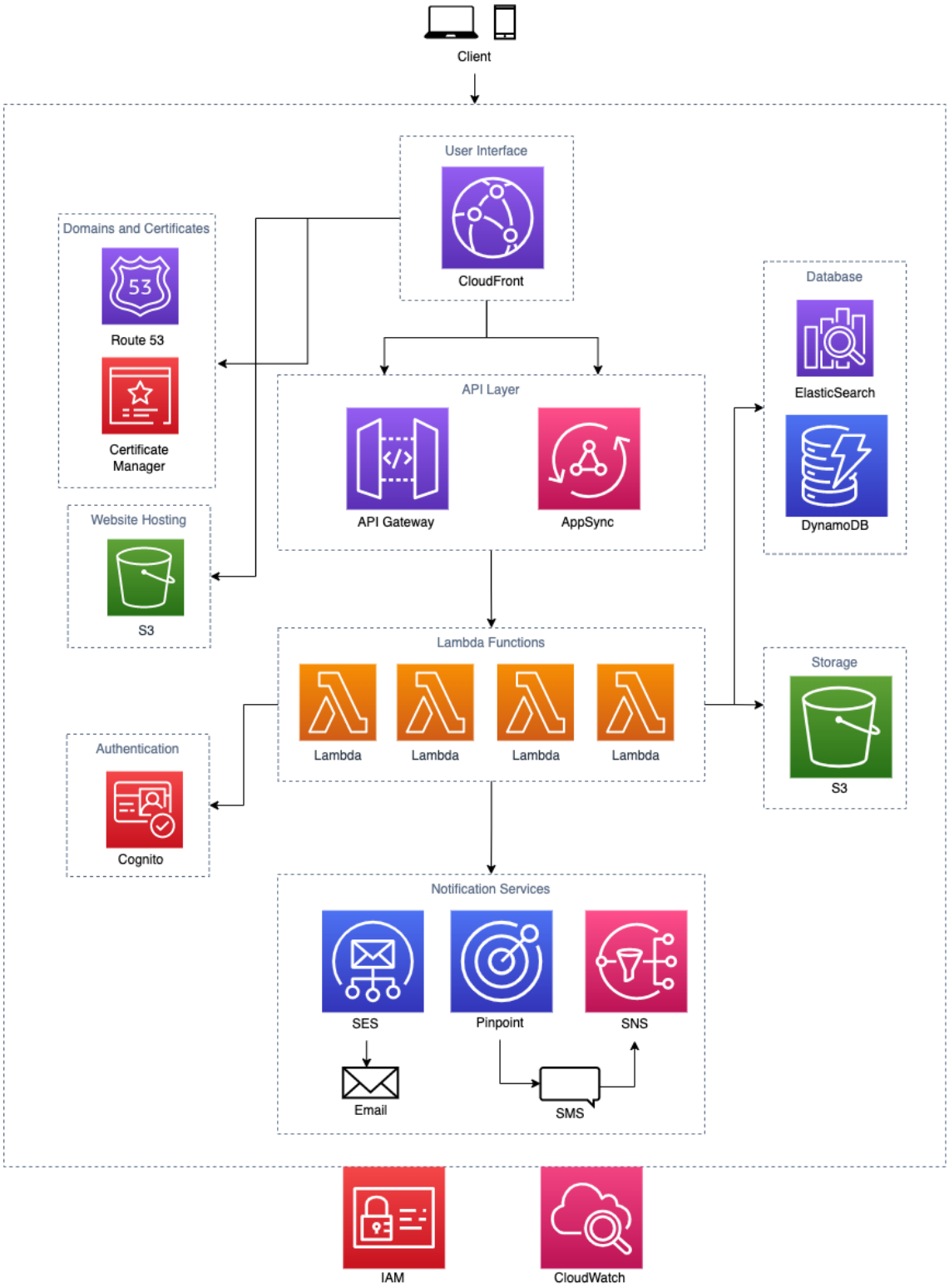

Figure 5.1: Lixani 5 architecture

CloudFormation Stacks

AWS resources that are used in the architecture are divided into different CloudFormation stacks. CloudFormation is a service that provides a way to create, provision, and manage multiple AWS resources as a single unit. CloudFormation handles the infrastructure deployments of the resources. A CloudFormation Stack is a collection of AWS resources. CloudFormation stacks are defined in template documents that improve repeatability and predictability. [43] Most of the CloudFormation stacks in use are divided into logical entities that have single responsibility in terms of business logic. Entities are, for example, Lixani-core, Lixani-files, Lixani-reports, and Lixani-SMS.

User Interface

Lixani 5 is a React application that is hosted on AWS S3 file system and delivered through AWS CloudFront service. CloudFront is a service that distribute content with low latency through worldwide edge locations. Domains and certificates are managed in AWS Route 53 service and AWS Certificate Manager service.

API Layer

Architectures API layer consists of AWS AppSync service and numerous API Gateway services. The application web interface communicates with AppSync through GraphQL requests and with API Gateway through HTTP requests. GraphQL is a query language for APIs. AppSync is a centralized GraphQL API layer provided by AWS. It parses the GraphQL requests and navigate them to corresponding resources – AWS Lambda functions. Lixani includes numerous API Gateways, such as HTTP API and REST API services. As well as AppSync, also the API Gateway services act as a middleware between Lambda functions and the web application interface.

Function as a Service

AWS Lambda functions are the core of the architecture. Lixani 5 consists of over 200 different Lambda functions. They handle all the logical tasks in the application. Lambdas are mostly invoked by the API:s - API Gateway and AppSync. Other ways to invoke lambdas are different events such as file addition to S3 storage system, and message triggers from Simple Queue Service (SQS) or Simple Notification Service (SNS). Some lambdas are also scheduled to trigger at a specific time.

Lambda functions interact with other AWS services to implement the business logic. The user authentication and user management are handled by Lambdas that interact with Cognito service. File handling is implemented through functions that interact with S3 Storage. Data operations are implemented with Lambdas that communicate with DynamoDB database and ElasticSearch indexes. Communication between users and the system through email and SMS messages are implemented with Lambdas that interact with three AWS services: Simple Email Service (SES), Pinpoint, and Simple Notification Service (SNS).

Security and Monitoring

Security between cloud resources is implemented with the AWS Identity and Access Management (IAM) service. IAM provides a method to manage access to different AWS resources. For example, every Lambda function has a role that includes a definition of every permission that they have. Permissions are provided with least-privilege method: provide only permissions that are required for the resource to work.

Monitoring is implemented in the whole system through AWS CloudWatch. It is a monitoring service that provides logging, allows creating and tracking different metrics, and allows setting alarms. Mostly, CloudWatch is used for its logging services. For example, every Lambda function has its own log file in the CloudWatch

service.

5.2 AWS Well-Architected Review

The Well-Architected review for the Lixani 5 software was conducted as a meeting between the Lixani development team and CEO. In the meeting, all the 58 questions in the review framework were considered in a conversational manner. In the following sections, the results are analyzed for each of the Well-Architected pillars.

Pillar 1: Operational excellence The first pillar emphasizes the operations inside the organization and the team. The pillar was an effective way to go through the general practices of the organization. Regarding the development team, positive things that rose up were active and transparent communication, and clearly defined responsibility areas. What comes to revising the current practices, the team thought that code review should be applied to the development process in the near future. Also, the documentation practices were considered to be enhanced. Organizational matters that came up were the need for improvement in feature definition and to conduct more research before the start of developing new components to the software. In addition, the governance requirements should gone through to ensure our system takes into account the Finnish legislation.

Pillar 2: Security The second pillar included ten questions regarding the security. This part stood out with positive and negative aspects concerning Lixani. General practices regarding AWS infrastructure usage such as multi-factor authentication, using separate accounts for different environment etc. were in good shape. Regarding the security in the system, there were some aspects that stood out. The external programming libraries that are on use, for example, React and Apollo should be upgraded to newer versions to prevent security threats regarding outdated

versions. The team noted that a specific process should be created for applying permissions to AWS resources in order to maintain the AWS least-privilege policy. Regarding the technical services that AWS offers, team noted that the analyzing of application logic should be enhanced. Automated response for exception handling and metrics to analyze AWS Lambda usage should be applied to reduce the time to notice unwanted events. In addition, the team pointed out that documentation should be enhanced related to the security aspects.

Pillar 3: Reliability The reliability pillar included questions related to the design of architecture to ensure it is working in a reliable manner. The architecture model selection – Serverless architecture – is a practical solution that works efficiently for the system's purposes. Considering that the Lixani 5 is still mainly in a development phase and used by only a small amount of customers, the elasticity of the software should be tested before the customer base grows. This is to ensure our logical functions are functional when the demand increases and to implement possible changes in early phase. In addition, the documentation between different logical functions inside the serverless architecture should be enhanced to improve the knowledge of correlations between different functions. What comes to monitoring the architecture and its state, previously mentioned automated response for exception events and implementing metrics for monitoring the logical functions are features that should be applied in the future. One question of the pillar was dedicated to backing up data and storing it securely. The team decided that the organization's back up practices should be revised for all AWS resources in use to ensure that back up operations are done according to best practices.

Pillar 4: Performance efficiency Pillar four consisted of eight questions related to the practices implementing performance. Questions handled the applying of new technologies, data storage solutions, database services, networking technologies, and

performance of general architectural decisions. The team evaluated current process for applying new technologies as an efficient one since the decisions are iterated multiple times between the team and managing persons before the solution is approved. The team highlighted that transparent communication should be maintained inside the organization.

The questions related to the database services stood out as an interesting conversation. The team pondered that the database solution of the system should be standardized. Currently, the external part of the Lixani 5 – tax reporting – uses both AWS (Relational Database Service) and AWS DynamoDB as database services while all other components use the DynamoDB service. In the future, the RDS usage should be replaced with the DynamoDB. By doing the replacament all of the data would be in a same location and thus, the handling of the data becomes better manageable.

Regarding the general performance efficiency of the software, one aspect stood out. Currently, the development team and the subcontractors are doing a transition from the GraphQl and AppSync API solution towards the use of an HTTP REST API model. Recently, it was noted that the GraphQl and AppSync solution became a bit of a bottleneck for the software efficiency and the decision was made to slowly refactor the software to use HTTP REST API solution. This transformation is already showing positive effects in the performance of the software.

Pillar 5: Cost optimization Cloud providers usually charge the customers based on the amount of usage per service. AWS offers different possibilities to monitor the monthly costs. The fifth pillar of the review consisted of questions regarding the customer's processes to monitor costs. The questions raised awareness on how the costs can be monitored and they extended the knowledge on what kind of possibilities AWS provides for cost management. Based on the questions, the team decided that all AWS resources that are on use should be periodically gone through and

inspected whether there are costs provided by resources that are unnecessary. AWS offers budgeting tools that are suggested to be in use for monitoring purposes. The budgeting tools could be in use to monitor whether there are some services providing unexpected costs. While cloud providers usually charge users based on the usage, AWS offers also other ways to handle billing. Reserved instances are AWS resources that are charged with fixed monthly pricing. The review questions invoked a need to evaluate different billing models for specific AWS resources that are offering the fixed pricing model.

Pillar 6: Sustainability Using cloud services there are numerous ways to think about the sustainability aspect. The pillar was a good addition to the framework since the sustainability issues were not considered earlier by the team members. Based on the questions in this pillar, the team considered that it should be noted that unused cloud resources should not be created or maintained.

5.3 Interviews With Experts

Interviews were conducted as separate meetings between an interviewer and interviewee. Sessions were productive meetings that included a broad conversation about the subject in general. The interviewees were the following persons:

- Janne Tammi, CTO, Lemonsoft Oyj
- Janne Annila, Architect, Lemonsoft Oyj
- Jari Ikävalko, Cloud Architect, Skillwell Oy

5.3.1 How system architecture is being evaluated in your company?

System architecture evaluation is an important part of software company's routines. When Lemonsoft is considering buying new companies to be part of the organization, the architecture evaluation becomes critical. Tammi has created his own technology audit template that is conducted to the target companies. The audit gathers different business and technology related stakeholders together to discuss the system architecture from different perspectives. Tammi's audit template is a pack of questions and topics that have been formed as a result from long career on the software development field.

Regarding the software development in Lemonsoft's own products, the architecture evaluation process differs. The evaluation is emphasized in the early phase of new product and during the adoption of new technologies, as Tammi explains. The evaluation of how new technologies fit in the current architectural environment is important. Also, Tammi states that the iterative evaluation of architecture design between the architect team and CTO before the start of development reduces the possibility for major turnarounds in the development phase.

Lemonsoft's main product includes business logic and implementations from over 15 years. Annila thinks that the architecture evaluation is important to make sure new entities of the software align with the architectural decisions that are already in use. Annila highlights the company's POC (Proof Of Concept) culture. When new major feature is going to be implemented to the software, the process starts by creating a POC. The POC is designed, previewed, and iterated inside the architect team to find possible architectural problems in the early phase.

Skillwell also does not rely on any specific evaluation model but has its own process for it. Ikävalko explains that through the organizational culture, they try to grow their consultants to approach the architecture in a similar way. The process

has been shaped from the broad expertise that the consultants have been acquired during their career. Ikävalko considers that the AWS Well-Architected framework have similar main points than their practices.

5.3.2 In which part of the architecture development process do you aim to evaluate the architecture?

In Lemonsoft, the architecture evaluation is emphasized in the early phase of the architecture development according to Tammi and Annila. The architecture model and required technologies are analyzed between multiple stakeholders including the architecture team and the CTO. Architecture evaluation becomes evident when new technologies are going to be applied in the current architecture. The technologies and architectural decisions are analyzed and iterated before start of their development. Problems related to the architectural decisions are tried to be found before the start of the implementation phase but it is not possible always. Thus, also the implementation phase requires architectural evaluation on how well the added feature align with the current architecture, ponders Annila.

According to Ikävalko, in Skillwell, the architecture is being evaluated in the very early phases of the system development process and re-evaluated constantly during the lifespan of the system. The re-evaluation is often done when there is need for greater revisions in the system. Cloud environment offers a wide set of different tools and constantly introduce new ones. The re-evaluation of architectural decisions is also a good way to mirror the architecture to the existing best practices of the cloud provider.

5.3.3 What are the advantages in architecture evaluation?

Acquiring subsidiaries requires a deep evaluation of target company's system architecture. According to Tammi, it is important to know what kind of system archi-

tecture is going to be part of the concern in the future. During the development of company's own products, the deep evaluation before bringing in new technologies and architectural decisions can save resources in a long run since it is in known what kind of advantages and problems can lie ahead when specific methods have taken in use, thinks both Tammi and Annila. Through deep evaluation, the development phase usually contains less surprises. Annila encourages his team to do even more research on architecture designing process to prevent future problems in a development phase.

Thorough architecture evaluation in a pre-development phase and constant re-evaluation of the software architecture during its lifespan maintain the architect's knowledge of the system's architectural decisions. "Constant re-evaluation of system architecture and good knowledge of it helps me to sleep well", explains Ikävalko.

5.3.4 If you have used both the traditional methods (ATAM, CBAM) and cloud specific methods which do you prefer and why?

None of the interviewees were used the traditional architecture evaluation methods. They all rely on the best practices that have been formed over their long careers and have created their processes for evaluating the system architecture. Ikävalko, who has conducted AWS Well-Architected reviews for their customers thinks that the review framework has good main points, but it usually requires much more context and deeper knowledge of the target architecture to form a comprehensive picture about it. Also, the design smells can be harder to find when only the Well-Architected review is used since it goes through the architectural decisions in a rather general manner.

6 Case Study Results

This chapter focuses on the results of the case study part of the thesis. The architecture review process results are discussed and the interviews with experts part is analyzed.

6.1 Architecture Review with AWS Well-Architected

Architecture review session with AWS Well-Architected framework was overall a successful event. The knowledge about the architecture was increased inside the team, numerous positive aspects were noted from the current development processes, and several problems were highlighted and considered to be revised in the future.

Review session started with the introduction to the system architecture in a general level, and then continued as a presentation of all individual components of the architecture. Presenting the architecture at the start of the review invoked a lot of conversation of different architectural decisions. The presentation session already extended and clarified the team's current knowledge of the system architecture. Also, the presentation of the software architecture updated the documentation of the system architecture.

After the discussion of the current architecture, the Well-Architected review questions were gone through for every individual pillar. The review raised a large number of matters that should be revised and improved on the organization level, in the development team, and in the system. The matters were collected and based

on the notes, tickets were created to the company's project management system. In addition to things that should be revised and improved, the team noted that a lot of matters are handled properly. For these matters, the team got confirmation that the current way of working is approved and the team should continue practicing it.

Regarding the overall technical architecture, the AWS Well-Architected review did not go really deep. The architecture review questions consisted mainly from general subjects regarding the architecture such as the development and organizational habits, organizational culture, and processes. The architecture review might not give deep knowledge from the technical architecture and it relies more on the way of developing the system and company as a whole. This was noted also by one of the experts in the interview part. Thus, Well-Architected review might not be optimal solution for finding the technical issues in the implementation. It concentrates more on preventing the happening of these issues by highlighting the best practices of building the architecture with AWS resources. Regarding other architecture evaluation models, ATAM is a model that often goes a bit deeper discussing the technical decisions of the architecture in the investigation and analysis part of the ATAM process. This can be seen from the Figure 4.1.

While the Well-Architected review might not be optimal choice for spotting technical issues related to the code itself, it could be beneficial in many ways. The review process forces the stakeholders to think about the company and the software from different perspectives. If the company does not yet practice constant reflecting on their processes, the Well-Architected review could be viable solution for doing it periodically. As noted previously, the review process also either extend or at least strengthen the system's architectural knowledge of all stakeholders.

6.2 Interviews with Experts

All the interviews were different types of sessions. The time doing the interview was projected to be approximately half an hour per interviewee. One of the interviews extended to be over one hour and thus, invoked deep conversation. Two of the interviews were conducted remotely and one was conducted on the company's premise.

Interestingly, none of the interviewees were heard about the classic architecture evaluation models such as ATAM or CBAM, and only one of them had previous experience with Well-Architected review. While they do not use any architectural evaluation models, they all still actively evaluate the architecture in a rather systematic way. One of them has their own technology audit template, another approaches the evaluation from the organizational culture, and one relies on a specific software development practice (POC) as a way to evaluate architecure.

The practices used by the interviewees have similarities to all the architecture evaluation methodologies introduced in Chapter 4. Technology audit template gathers different stakeholders together to ponder the software itself, the architecture, and the company's processes. Similarly, the ATAM, CBAM, and Well-Architected reviews are all targeted to different stakeholders, not only technical persons. Some of the main points in all the evaluation methods that were handled in Chapter 4 are active participating of different stakeholders and open communication. These rise up also in the evaluation processes that are common among the interviewees.

Broad experience from the software development field has created practices that all the interviewees use in their architecture evaluation process in their daily work. The processes have iterated and formed during their careers. The methods also have similarities to academic architecture evaluation models and have similar main points than the Well-Architected pillars.

7 Conclusions

This chapter provides the conclusions of the thesis, answers to the research questions and provides possibilities for future studies. The goal of the chapter is to compress research outcomes into conclusions.

The thesis included a literature review for cloud computing, for cloud architecture models and for cloud architecture evaluation models. It included a case study section that introduced the case study company and the software. The case study also discussed the case study software's system architecture in a general overview. After the discussion, the cloud architecture evaluation was performed to the software using the AWS Well-Architected framework. Third research method was interviews with experts. The interviews included a discussion based on multiple questions about the architecture evaluation.

7.1 Answering the Research Questions

This section answers to the two research questions introduced at the beginning of the thesis. The answers are introduced based on the literature review, case study and expert interviews.

RQ1: How the cloud software architecture can be evaluated?

The cloud software architecture can be evaluated in many different ways. The architecture evaluation models can be divided into two categories: traditional models

and cloud-specific models. This thesis introduced two traditional evaluation models and one cloud-specific model. Traditional models were architecture trade-off analysis method (ATAM) and cost-benefit analysis method (CBAM). Both methods are viable solutions for executing architecture evaluation in the cloud environment. The cloud-specific model that was discussed in the thesis was AWS Well-Architected framework. As seen from the case study, this was also a proper solution to perform an architecture evaluation. The models are viable solutions but come also with negative sides. It was seen that the AWS Well-Architected framework does not go deep in the technical architecture decisions. Regarding the traditional methods, the human-centricity in the approaches and the required commitment can cause problems related to the costs and the quality of results.

One of the research methods of the thesis was interviews with experts. Based on the interviews, it was noticed that the architecture evaluation can also be done without a specific traditional or cloud-specific model. One of the interviewees followed his own technology audit template as a result of a long career in the industry. Another interviewee relied on the organizational culture where they were growing their consultants to approach the architecture in a similar way. Third interviewee introduced the organization's proof of concept (POC) culture as a way to analyze the architecture. These results show that there are numerous different ways to approach architecture evaluation.

RQ2: How efficient are the software architecture evaluation tools provided by cloud companies? Are they more beneficial than the traditional evaluation models?

This thesis discussed two traditional architecture evaluation methods: architecture trade-off analysis method (ATAM) and cost-benefit analysis method (CBAM). The architecture evaluation tool provided by cloud company discussed in thesis was

AWS Well-Architected framework. The case study was conducted using the AWS Well-Architected framework. The case study results showed that the AWS Well-Architected framework is a viable tool for reflecting the processes of the development team, reviewing the system from different perspectives such as security, reliability, and costs. It also provides further knowledge about different architectural decisions since it is common to start the review by going through the state of the architecture.

Based on the results of the case study, the cloud-specific model is not seen as more beneficial comparing to the traditional models. The AWS Well-Architected framework is a good model for reviewing the development processes in general but regarding the architectural problems, it is not very efficient. As noted previously, for example, ATAM provides deeper discussion and understanding about different architectural decisions.

7.2 Further Studies

The thesis includes a literature review of the cloud computing definition and concepts in general, a literature review of the software architecture models, and a literature review of the architecture evaluation models. The architecture evaluation models chapter discussed three evaluation models that included two traditional models and one cloud-specific model. The subject could be improved by adding more models to the discussion. It could be specifically beneficial to introduce more cloud-specific models.

The case study part of the thesis included an architecture evaluation for the case application and interviews with experts. A further study suggestion would be to perform an architecture evaluation to the case application using a traditional architecture evaluation model such as architecture trade-off analysis method (ATAM) or cost-benefit analysis method (CBAM). This thesis performs an architecture evaluation only using the cloud-specific model AWS Well-Architected.

One interesting result from the interviews with experts was that the interviewees were not using constantly any specific architecture evaluation models. They were mostly leaning on other methods to analyze architectures that were molded during their careers. A further study suggestion would be to research this observation more. It could be possible to perform a broad survey research that could include a large number of software companies in Finland. The subjects of the survey could be to gather information on what kind of architecture evaluation models the companies are really using in the industry.

References

[1] D. C. Marinescu, *Cloud Computing: Theory and Practice*. San Francisco, UNITED STATES: Elsevier Science Technology, 2013, ISBN: 978-0-12-404641-2. [Online]. Available: http://ebookcentral.proquest.com/lib/kutu/detail.action?docID=1213925.

[2] P. Mell, T. Grance, *et al.*, "The NIST definition of cloud computing", 2011. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-145/final.

[3] C. Baun, M. Kunze, J. Nimis, and S. Tai, *Cloud Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, ISBN: 978-3-642-20916-1. DOI: 10.1007/978-3-642-20917-8. [Online]. Available: http://link.springer.com/10.1007/978-3-642-20917-8.

[4] N. K. Sehgal and P. C. P. Bhatt, *Cloud Computing: Concepts and Practices*. Cham: Springer International Publishing, 2018, ISBN: 978-3-319-77838-9. DOI: 10.1007/978-3-319-77839-6. [Online]. Available: https://link.springer.com/10.1007/978-3-319-77839-6.

[5] ISO/IEC/IEEE 42010, *Iso/iec/ieee 42010: Defining architecture*. [Online]. Available: http://www.iso-architecture.org/ieee-1471/defining-architecture.html (visited on 12/04/2022).

[6] Z. Qin, X. Zheng, and J. Xing, "Architectural styles and patterns", *Software Architecture*, pp. 34–88, 2008.

[7] T. Suzuki and L. Suzuki, "On the benefit of 3-tier SOA architecture promoting information sharing among TMS systems and Brazilian e-Government Web Services: A CT-e case study", *arXiv preprint arXiv:2005.13047*, 2020.

[8] M. Malawski, A. Gajek, A. Zima, B. Balis, and K. Figiela, "Serverless execution of scientific workflows: Experiments with hyperflow, aws lambda and google cloud functions", *Future Generation Computer Systems*, vol. 110, pp. 502–514, Sep. 2020, ISSN: 0167739X. DOI: 10.1016/j.future.2017.10.029.

[9] D. Ustiugov, "Data-centric serverless cloud architecture", 2022. [Online]. Available: https://era.ed.ac.uk/handle/1842/39060.

[10] Amazon Web Services, Inc., *Tutorial: Build a serverless web application with aws lambda, amazon api gateway, aws amplify, amazon dynamodb, and amazon cognito*. [Online]. Available: https://aws.amazon.com/getting-started/hands-on/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/ (visited on 10/26/2022).

[11] Google, *Google cloud documentation, write cloud functions*. [Online]. Available: https://cloud.google.com/functions/docs/writing (visited on 10/28/2022).

[12] Microsoft, *Microsoft azure documentation, supported languages in azure functions*. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-functions/supported-languages (visited on 10/28/2022).

[13] Amazon Web Services, Inc., *Aws lamda documentation, lambda runtimes*. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html (visited on 10/28/2022).

[14] J. Thönes, "Microservices", *IEEE Software*, vol. 32, no. 1, pp. 116–116, Jan. 2015, ISSN: 1937-4194. DOI: 10.1109/MS.2015.11.

[15] M. Söylemez, B. Tekinerdogan, and A. Kolukısa Tarhan, "Challenges and solution directions of microservice architectures: A systematic literature review", *Applied Sciences*, vol. 12, no. 1111, p. 5507, Jan. 2022, ISSN: 2076-3417. DOI: 10.3390/app12115507.

[16] Amazon Web Services, *Simple microservices architecture on aws*. [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/simple-microservices-architecture-on-aws.html (visited on 11/03/2022).

[17] Microsoft, *Compute options for microservices with azure*. [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/microservices/design/compute-options (visited on 11/03/2022).

[18] M. Gancarz, "8 - making unix do one thing well", in *Linux and the Unix Philosophy*, M. Gancarz, Ed. Woburn: Digital Press, 2003, pp. 127–136, ISBN: 978-1-55558-273-9. DOI: https://doi.org/10.1016/B978-155558273-9/50010-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781555582739500105.

[19] M. Villamizar, O. Garcés, L. Ochoa, *et al.*, "Cost comparison of running web applications in the cloud using monolithic, microservice, and aws lambda architectures", *Service Oriented Computing and Applications*, vol. 11, no. 2, pp. 233–247, Jun. 2017, ISSN: 1863-2386, 1863-2394. DOI: 10.1007/s11761-017-0208-y.

[20] EdPrice-MSFT, *Microservices architecture design - Azure Architecture Center*. [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/microservices/ (visited on 10/14/2022).

[21] G. K. Aroraa, L. Kale, and K. Manish, *Building Microservices with . NET Core: Architect Your . NET Applications by Breaking Them into Really Small*

Pieces—microservices—using This Practical, Example-Based Guide. Birmingham, UNITED KINGDOM: Packt Publishing, Limited, 2017, ISBN: 978-1-78588-496-2. [Online]. Available: http://ebookcentral.proquest.com/lib/kutu/detail.action?docID=4877937.

[22] G. Blinowski, A. Ojdowska, and A. Przybyłek, "Monolithic vs. microservice architecture: A performance and scalability evaluation", *IEEE Access*, vol. 10, pp. 20 357–20 374, 2022, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3152803.

[23] Amazon Web Services, *Amazon EC2*. [Online]. Available: https://aws.amazon.com/ec2/ (visited on 11/04/2022).

[24] Google, *Google cloud compute engine*. [Online]. Available: https://cloud.google.com/compute (visited on 11/04/2022).

[25] O. Al-Debagy and P. Martinek, "A comparative review of microservices and monolithic architectures", no. arXiv:1905.07997, May 2019, arXiv:1905.07997 [cs]. DOI: 10.48550/arXiv.1905.07997. [Online]. Available: http://arxiv.org/abs/1905.07997.

[26] N. C. Mendonça, C. Box, C. Manolache, and L. Ryan, "The monolith strikes back: Why istio migrated from microservices to a monolithic architecture", *IEEE Software*, vol. 38, no. 5, pp. 17–22, Sep. 2021, ISSN: 1937-4194. DOI: 10.1109/MS.2021.3080335.

[27] A. Patidar and U. Suman, "A survey on software architecture evaluation methods", in *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, IEEE, 2015, pp. 967–972.

[28] D. Sobhy, R. Bahsoon, L. Minku, and R. Kazman, "Evaluation of software architectures under uncertainty: A systematic literature review", en, *ACM Transactions on Software Engineering and Methodology*, vol. 30, no. 4, pp. 1–50, Oct. 2021, ISSN: 1049-331X, 1557-7392. DOI: 10.1145/3464305.

[29] F. Faniyi, R. Bahsoon, A. Evans, and R. Kazman, "Evaluating security properties of architectures in unpredictable environments: A case for cloud", in *2011 Ninth Working IEEE/IFIP Conference on Software Architecture*, Jun. 2011, pp. 127–136. DOI: 10.1109/WICSA.2011.25.

[30] R. Kazman, M. Klein, and P. Clements, *ATAM: Method for Architecture Evaluation:* Fort Belvoir, VA, Aug. 2000. DOI: 10.21236/ADA382629. [Online]. Available: http://www.dtic.mil/docs/citations/ADA382629.

[31] J. Lee, S. Kang, and C.-k. Kim, "Software architecture evaluation methods based on cost benefit analysis and quantitative decision making", *Empirical Software Engineering*, vol. 14, no. 4, pp. 453–475, Aug. 2009, ISSN: 13823256. DOI: 10.1007/s10664-008-9094-4.

[32] R. Kazman, J. Asundi, and M. Klien, *Making Architecture Design Decisions: An Economic Approach:* Fort Belvoir, VA, Sep. 2002. DOI: 10.21236/ADA408740. [Online]. Available: http://www.dtic.mil/docs/citations/ADA408740.

[33] Amazon Web Services, *AWS Well-Architected Framework, Introduction.* [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html (visited on 12/14/2022).

[34] Amazon Web Services, *AWS Well-Architected Framework, Definitions.* [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/framework/definitions.html (visited on 12/14/2022).

[35] Amazon Web Services, *AWS Well-Architected Framework, Operational Excellence.* [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/framework/oe-design-principles.html (visited on 12/14/2022).

[36] Amazon Web Services, *AWS Well-Architected Framework, Security.* [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/framework/security.html (visited on 12/14/2022).

[37] Amazon Web Services, *AWS Well-Architected Framework, Reliability*. [Online]. Available: `https://docs.aws.amazon.com/wellarchitected/latest/framework/reliability.html` (visited on 12/14/2022).

[38] Amazon Web Services, *AWS Well-Architected Framework, Performance Efficiency*. [Online]. Available: `https://docs.aws.amazon.com/wellarchitected/latest/framework/performance-efficiency.html` (visited on 12/14/2022).

[39] Amazon Web Services, *AWS Well-Architected Framework, Cost Optimization*. [Online]. Available: `https://docs.aws.amazon.com/wellarchitected/latest/framework/cost-optimization.html` (visited on 12/14/2022).

[40] Amazon Web Services, *AWS Well-Architected Framework, Sustainability*. [Online]. Available: `https://docs.aws.amazon.com/wellarchitected/latest/framework/sustainability.html` (visited on 12/14/2022).

[41] Amazon Web Services, *AWS Well-Architected Framework, Review Process*. [Online]. Available: `https://docs.aws.amazon.com/wellarchitected/latest/framework/the-review-process.html` (visited on 12/14/2022).

[42] Serverless, Inc, *Serverless Framework Concepts*. [Online]. Available: `https://www.serverless.com/framework/docs/providers/aws/guide/intro` (visited on 12/22/2022).

[43] Amazon Web Services, *AWS CloudFormation Documentation*. [Online]. Available: `https://docs.aws.amazon.com/cloudformation/index.html` (visited on 12/29/2022).