DESIGNING A HIGHLY AVAILABLE AND SCALABLE CLOUD ARCHITECTURE FOR A WEB APPLICATION

Lappeenranta–Lahti University of Technology LUT
Master's Programme in Software Product Management and Business
Master's thesis 2022
Janetta Huoponen

Examiners: Associate Professor Sami Hyrynsalmi
University Lecturer Erno Vanhala

ABSTRACT

Lappeenranta–Lahti University of Technology LUT

LUT School of Engineering Science

Software Engineering

Janetta Huoponen

Designing a highly available and scalable cloud architecture for a web application

Master's thesis 2022

67 pages, 17 figures, 2 tables, and 1 appendix

Examiners: Associate Professor Sami Hyrynsalmi

University Lecturer Erno Vanhala

Keywords: cloud computing, cloud architecture, web application, scalability, availability

Today, the highly competitive software industry sets high expectations for modern web applications, including scalability and availability. Being able to scale allows a web application to adapt to an increasing number of users on the application simultaneously, whereas availability allows the application to continue functioning after a component failure. Successful cloud architecture is designed with these principles in mind. The purpose of this thesis is to study how to design a highly available and scalable cloud architecture for a web application. The architecture is designed, demonstrated, and analyzed obeying a design science research method. The architecture is done by following the well-architected framework and interviewing experts in architectural cloud design. The most significant finding of this thesis is that the designed cloud architecture and its components enabled high availability and scalability capabilities for the web application used for demonstration purposes.

TIIVISTELMÄ

Lappeenrannan–Lahden teknillinen yliopisto LUT

LUT School of Engineering Science

Tietotekniikan koulutusohjelma

Janetta Huoponen

Korkean saatavuuden ja skaalautuvuuden pilviarkkitehtuurin suunnitteleminen web-sovellukselle

Diplomityö 2022

67 sivua, 17 kuvaa, 2 taulukkoa ja 1 liite

Tarkastajat: Apulaisprofessori Sami Hyrynsalmi

Yliopisto-opettaja Erno Vanhala

Avainsanat: pilvipalvelu, pilviarkkitehtuuri, web-sovellus, skaalautuvuus, saatavuus

Tänä päivänä ohjelmistotuotanto ja alalla vallitseva kilpailu asettavat web-sovelluksille korkeat vaatimukset. Skaalautuvuutta voidaan pitää yhtenä tärkeimpänä avaintekijänä web-sovelluksen menestymisessä. Sen avulla voidaan varautua ja vastata merkittävien samanaikaisten käyttäjämäärien kasvuun automaattisesti. Lisäksi web-sovellus on suunniteltava saatavuusnäkökulma huomioiden. Tällä varmistetaan, että yhden tai useamman palvelun häiriöllä ei ole vaikutusta web-sovelluksen saatavuuteen. Tämän diplomityön tavoitteena oli tutkia, kuinka suunnitellaan korkeasti saatavan ja skaalautuvan web-sovelluksen pilviarkkitehtuuri. Pilviarkkitehtuuri suunniteltiin, havainnollistettiin ja analysoitiin noudattaen suunnittelutieteellisen tutkimuksen menetelmää. Pilviarkkitehtuuri toteutettiin hyödyntämällä teoreettista viitekehystä ja pilviarkkitehtuuriin erikoistuvien asiantuntijoiden näkemyksiä. Työn merkittävimpänä havaintona ja tuloksena voidaan pitää suunnitellun pilviarkkitehtuurin ja sen sisältämien komponenttien mahdollistavan korkean saatavuuden ja skaalautuvuuden esimerkkisovellukselle.

ACKNOWLEDGEMENTS

I would like to thank my supervisors Sami Hyrynsalmi and Erno Vanhala for their guidance and valuable feedback.

February 2022
Helsinki, Finland

Janetta Huoponen

SYMBOLS AND ABBREVIATIONS

AD	Active Directory
ALB	Application Load Balancer
AMI	Amazon Machine Image
ASR	Architecturally Significant Requirement
AWS	Amazon Web Services
AZ	Availability Zone
CDN	Content Delivery Network
CIDR	Classless Inter-Domain Routing
CPU	Central Processing Unit
DNS	Domain Name System
EC2	Elastic Compute Cloud
ELB	Elastic Load Balancing
GCP	Google Cloud Platform
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
ICT	Information and Communications Technology
LOR	Least Outstanding Requests
NAT	Network Address Translation
NLB	Network Load Balancer
PaaS	Platform as a Service
POP	Point of Presence
RAM	Random Access Memory
RR	Round Robin
S3	Simple Storage Service
SaaS	Software as a Service
SPOE	Single Point of Entry
SPOF	Single Point of Failure
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

VM	Virtual Machine
VPC	Virtual Private Cloud

Table of contents

1. Introduction

The software industry is one of the most rapidly developing sectors in the world. Today, end users require web applications which will not crash and are available on multiple platforms and scales. These two quality attributes—availability and scalability—are the key characteristics of high-quality web applications (Qasim 2015). In order to create resilient and scalable web applications, care needs to be taken in its design. Necessary technologies and components need to be chosen and linked together properly. Hassan (2011) states that cloud computing is a response to the on-demand access to web application's remote resources and enables users to face market volatility in an agile manner. Thus, using cloud computing provides the opportunity to design and implement highly available and scalable web applications based on services that have their functionalities and each of these committed services can be managed automatically.

The term *cloud computing* was permanently added to information technology vocabulary in early 2006, though the idea of cloud computing had been created by Professor John McCarthy several decades ago. McCarthy invented a time-sharing computer system where computing resources such as time, processing cycles, and storage space could be shared simultaneously with many users (Haigh & Ceruzzi 2021, 113). This kind of computing allows resources to be allocated according to demand (Liu et al. 2011, 13).

As web application development is switching to cloud-based servers, the need for and importance of cloud architecture is emphasized. Salam et al. (2015, 79) state that cloud architecture refers to describing the components and services required for cloud computing. Additionally, the relationships and how different components and services of a web application are tied together can be illustrated by modeling cloud architecture.

The objective of this thesis is to solve the research problem: how to design a highly available and scalable cloud architecture for a web application. This is due to the desire to set a well-defined and justified scope for the work so that the research results can be presented clearly and comprehensively.

In order to reach the deliverable mentioned above and create a methodologically sound research project, there are a number of steps to be taken. First, a literature review of cloud computing and related themes needs to be done. The review forms a theoretical framework for the thesis, assesses the current state of the research topic, and assures that the appropriate data sources and best practices are used. It is also used to assure that the appropriate data sources and best practices are adopted in the thesis. Second, requirements need to be defined for the designed architecture. Third, a proper research method and process need to be chosen. Fourth, a suitable cloud platform needs to be selected because it defines what kind of services can be chosen when designing a cloud architecture. Finally, the cloud architecture can be modeled and evaluated.

Based on the previous chapters, the actual cloud architecture is designed and each chosen service is explained in more detail. Reasons and justifications for the selected components, services, and technologies are provided. Next, the functionality of the designed cloud architecture is demonstrated using an example web application. Finally, discussion and conclusions are provided in the last two chapters.

2. Literature review

This chapter gives a brief introduction to cloud computing, its essential characteristics, and cloud service models. Additionally, different cloud platform providers are introduced.

2.1 Cloud computing

According to Liu et al. (2011), cloud computing refers to a model which enables easy access to shared computing resources such as services, applications, and storage over the internet. Mell & Grance (2011) presents five essential characteristics of cloud computing:

1. Automatic on-demand utilization of computing resources,
2. Accessibility of resources through standardized mechanisms on a variety of devices such as laptops, tablets, and mobile phones,
3. Orchestrating resources such as memory, processing power, and storage into a single large pool to which all consumers have simultaneous access,
4. Computing resources are elastically and automatically provisioned and released according to consumer demand, and
5. The usage of resources is measured automatically and continuously.

The goal of cloud computing is to provide a simple way for end-users to meet the changing business requirements through different types of service models. Mell & Grance (2011) present the three most popular cloud service models. These are presented as follows and illustrated in Figure 1.

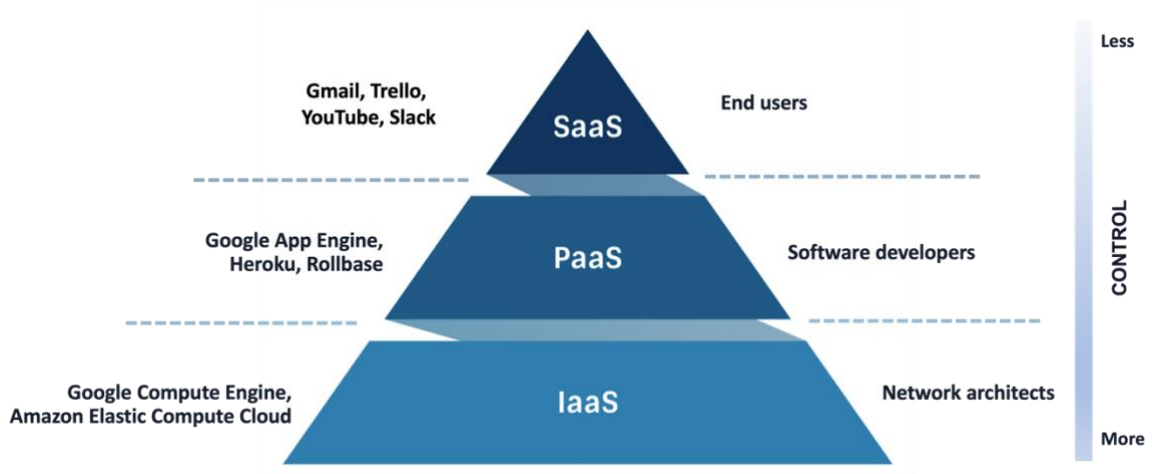

Figure 1. Cloud service models (based on Salas-Zárate & Colombo-Mendoza 2011).

The first model is Software as a Service (SaaS). In this model, the application is running on a provider's cloud infrastructure and consumers access the application's resources over the internet (Buxmann et al. 2013, 169). These kinds of applications are provided to end-users through a client interface such as a web browser. The consumer does not manage or control the underlying cloud infrastructure e.g. services, storage, or operating systems (Savchenko 2019).

The second most popular cloud service model is Platform as a Service (PaaS) where consumers are provided with the capabilities to create their applications using tools and services supported by the cloud service provider. Just like in the Software as a Service model, the consumer does not manage or control the underlying cloud infrastructure. The difference in this model is that the consumer can control deployed applications and configure the application-hosting environment. (Mell & Grance 2011)

The last model is Infrastructure as a Service (IaaS). This model forms the basis for the higher-level abstractions of the Software as a Service and Software as a Platform models. The consumer has control over the computing resources, storage, and services. Nevertheless, the underlying cloud infrastructure is still controlled by the cloud provider. (Mell & Grance 2011)

To summarize, cloud computing covers both the delivery of the web application through the Internet and the hardware and software that are required to provide these services (Ojala 2013).

2.2 Software architecture

In this thesis, the concept of cloud architecture is examined from three different perspectives. A general definition of software architecture can be used to explain the concept of cloud architecture. Brown (2016, 3-4) draws a distinction between architecture as a noun and as a verb. As a noun, *architecture* is perceived as a structure – software which can be broken down into smaller parts. The structure may consist of components, elements, and building blocks, and the interactions and relationships of these items are designed within the architecture.

Similarly to Brown, Bass et al. (2015, 6) defines architecture as "an abstraction of an application that illustrates selected and non-selected details". The simplest way to define an architecture is to see it as a set of software structures. The structure is composed of many elements and modules and the relations among them, using the term *architecture* as a noun.

Looking at *architecture* as a verb means using architecture as an ideological vision or project roadmap. Using *architecture* in this way means creating a roadmap which takes into account any necessary requirements, known as architectural drivers (Brown 2016, 3-4). These drivers can include quality and functional requirements, as well as business and technical constraints. The goal is to create a technical roadmap based on the above-mentioned requirements that can be used while communicating this vision to various stakeholders.

2.3 Cloud service providers

Today, service providers are increasingly offering cloud-based services for their customers. Thus, cloud-based solutions have grown into a sizable market for the IT industry. Three cloud service providers play a leading role in driving this change – Google Cloud, Microsoft

Azure, and Amazon Web Services. In this section, the players and their services will be explained in more detail.

2.3.1 Google Cloud Platform

Bala et al. (2021) state that Google Cloud Platform (GCP) is the world's third-largest cloud provider. GCP offers public cloud services in over 20 geographical regions, and is built on Google's self-designed and assembled infrastructure (Google 2021, Krishnan & Gonzalez 2015). GCP provides a highly scalable and reliable basis for software developers to build, test, deploy, and monitor software applications. GCP stands out among other service providers by its open-source mindset, constantly evolving internal research and expertise. For example, Google has been a major player in building Kubernetes for automating and orchestrating containerized applications (Kubernetes 2021).

2.3.2 Microsoft Azure

The world's second-largest cloud provider is Microsoft Azure which offers previously introduced PaaS and IaaS capabilities. This also explains the success of the platform. Due to the fact that it supports many different use cases, enterprises can deploy simple virtual machines (VM) or build a web application from scratch on Microsoft Azure's cloud environment (Gartner 2021). In addition, Microsoft benefits from an optimized product portfolio – enterprises can buy a combination of Teams, Office 365, and Azure with a single click. According to Bala et al. (2021), one of the disadvantages of Microsoft services is poor resiliency with critical services such as Azure Active Directory (Azure AD).

2.3.3 Amazon Web Services

According to Bala et al. (2021), Amazon Web Services (AWS) is the leading cloud service provider. The reason for this is the comprehensiveness and diversity of its services. Along with its breadth of services, they are well-tested and reliable – making it an attractive and safe choice for customers. In addition, AWS' comprehensive documentation enables the implementation of best practices in the development of sustainable applications.

The AWS Well-Architected Framework provides a good example of essential and fundamental documentation. The idea of the framework is to provide a "self-service portal" where their customers can ensure that their designed architecture is following AWS' best practices. The Well-Architected Framework consists of five pillars, as shown in Figure 2.

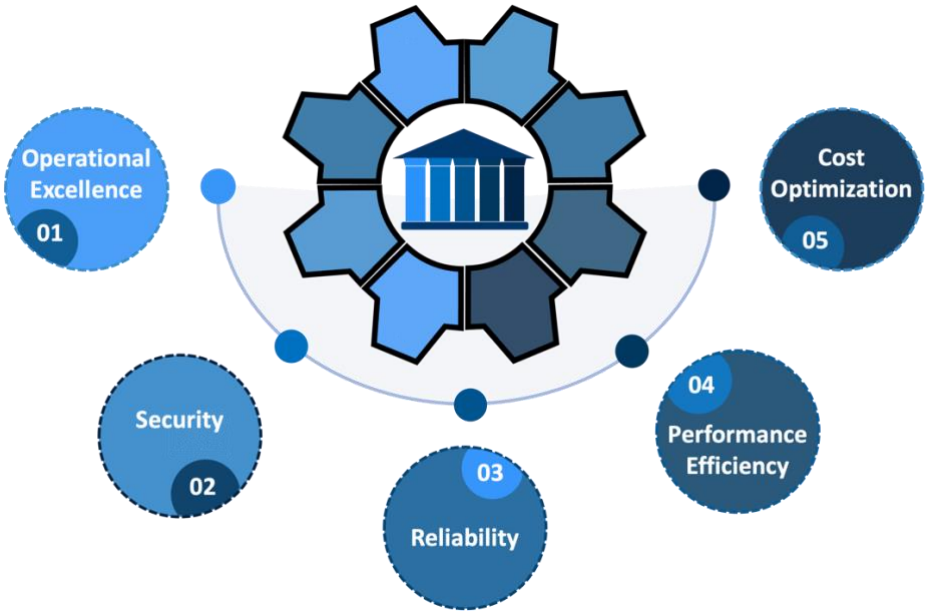

Figure 2. AWS Well-Architected Framework (based on Amazon Web Services 2021b).

The first pillar deals with operational excellence, which encourages performing different operations as a code such as deploying a web application. Doing things programmatically and having operations as a code has multiple benefits: the documentation will be within the code, and it allows for reversible operations. (Amazon Web Services 2021b)

The second pillar is security, where the focus is on identifying the foundation – by asking questions such as who has access, who needs to be authenticated, who needs to be granted permissions, and what permissions want to be granted (Amazon Web Services 2021b). By enabling traceability, customers are able to investigate and monitor different events that have happened in their environment. Additionally, based on AWS' architectural best practices, security configurations need to be implemented into all application's layers.

The third pillar, reliability, focuses on making the environment highly available and fault-tolerant. This means that computing resources are dynamically acquired to meet demand. Also, recovering from service or infrastructure failures is fast and efficient.

The fourth pillar is about software performance. Choosing resources of the right size and type is crucial – in order for the application to perform optimally, there needs to be enough capacity available. Additionally, the application must be built in a way that it scales as demand changes. The Well-Architected framework encourages users to delegate more complex tasks to a cloud service provider. Therefore, instead of developing services from scratch, it would be more efficient to consume technology as a service. Also, mechanical sympathy is emphasized – meaning that the chosen services are based on application requirements. (Amazon Web Services 2021b)

The final pillar is about optimizing costs by eliminating unneeded expenses and using the built-in managing services. The Well-Architected framework provides information and examples on how users can build cost-aware workloads while meeting business requirements and maximizing return on investment. (Fitzsimons et al. 2020)

2.4 Availability

According to Sommerville (2011), availability is one of the principal properties of dependability. He states that the availability of a web application is the probability of an application to keep functioning and serving end-users as requested. A commonly used criterion of highly available web applications is that resources are deployed to more than one data center (Sommerville 2011). In order to ensure proper resource deployment, Ronzon (2016) states that designing a highly available web application architecture usually requires executing fallback scenarios. These scenarios help to identify, understand, and mitigate the impacts of a failed resource on the performance of the entire application. Availability builds on the concept of asset recovery: when a resource fails, an alternative implementation of the resource is automatically available.

2.5 Scalability

Scalability is one of the key quality attributes of a high-quality web application as it allows the application to accommodate growth without changing the architecture. In its simplicity, it refers to the ability of a web application to provide adequate resources as requests for the application increase. Additionally, it encompasses the ability to decrease the resources when they are not in use—making the application more cost-effective (Khare et al. 2012). Kriushanth et al. (2013) present two common scaling types in a cloud environment:

- Vertical scaling
- Horizontal scaling

Vertical scaling, also known as scaling up, refers to the ability to increase the capacity of a resource to meet the growing demands of the web application. This can be done by adding more power, such as random access memory (RAM) and central processing units (CPU) to an existing resource, as shown in Figure 3.

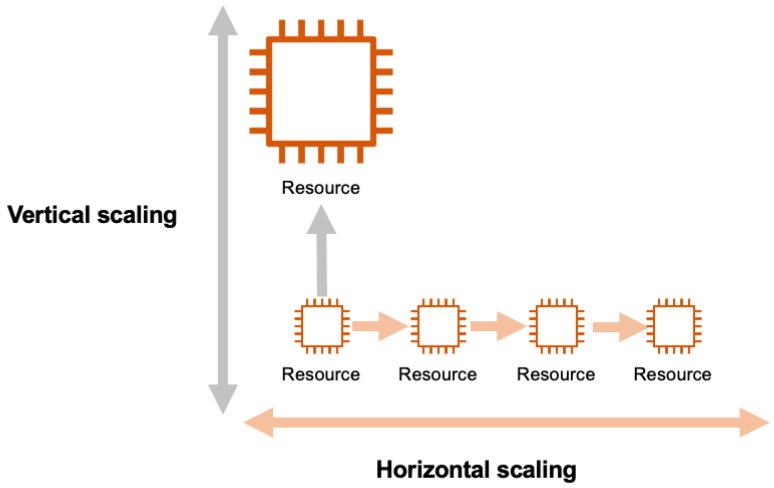

Figure 3. Types of cloud scalability (based on Kriushanth et al. 2013).

A challenge here, however, is that the resource is not redundant due to the fact that the architecture includes only a single resource which is resized as required. To address this challenge, it is necessary to divide the requests coming to the application into different

resources and scale them independently. This can be done by utilizing horizontal scaling (also known as scaling out) within the architecture (Roy et al. 2019). Horizontal scaling refers to a process where the number of resources is increased within a single tier and resources work together to cope with new demands. However, horizontal scaling has its own challenges. Because it requires definitions of how resources work together and their individual responsibilities, horizontal scaling often increases the complexity of a software architecture.

3. Research problem, methodology, and process

In this chapter, the research problem and motivation are established. After that, the proper research method and process are chosen. These choices were made by studying existing research methods and applying them to the objective of the thesis.

3.1 Research problem and motivation

A single failure in any component causes an error in a web application, leaving the service unavailable to end-users. As the volume of data and number of users are constantly increasing, efficient performance and high availability attributes are essential for web applications (Mahmudova 2019). At the same time, cloud service providers are shaping and diversifying global information and communication technology (ICT) markets: companies are migrating existing applications to cloud service platforms, such as Google Cloud Platform, Microsoft Azure, and Amazon Web Services. Obviously, the current trend is that web applications which are developed from scratch are more likely to be deployed on a cloud infrastructure rather than keeping them on-premises. Therefore, the need for proper planning regarding the design of web application architecture and development is essential. These conditions provide motivation to solve the research problem of this thesis: how to design a highly available and scalable cloud architecture for a web application.

3.2 Research process

To solve the research problem, design science was selected as the proper research method. According to Peffers et al. (2008), design science strives to create and evaluate IT artifacts to solve the identified organizational problems. Gasevic et al. (2009) state that IT artifacts can include models, diagrams, structures, frameworks, documents, components, and mock-ups that enable or support solving the problem and/or achieving the objective of the research. Peffers et al. (2008) implied that a rigorous process must be followed to design an IT artifact, thus they have created a process model (Figure 4) for design science research that includes six steps: problem identification and motivation, objectives of a solution, design and

development, demonstration, evaluation, and communication. Additionally, they have identified four possible entry points for research: problem-centered initiation, objective-centered initiation, design and development centered initiation, and client/context initiation.

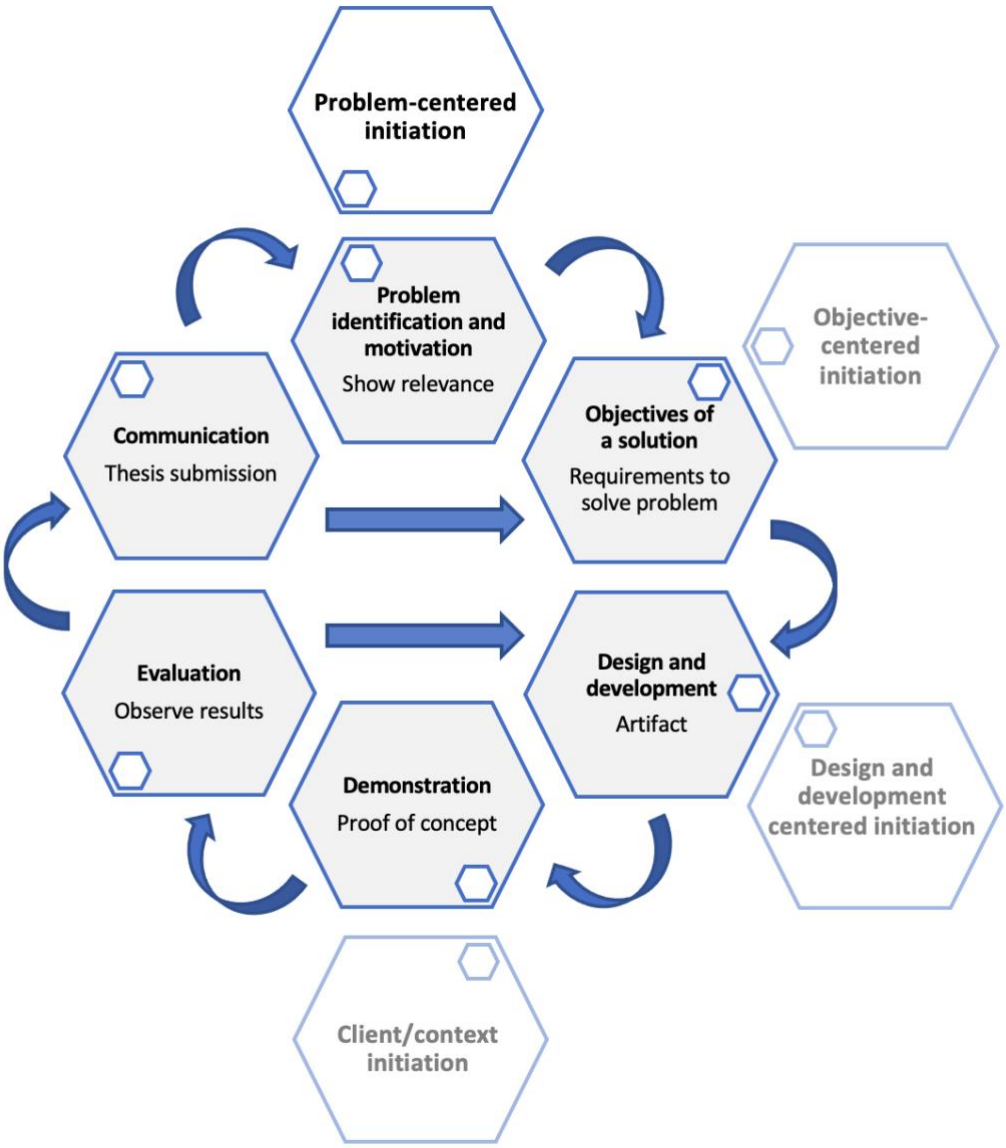

Figure 4. Design science research method process model (based on Peffers et al. 2008).

In this thesis, the chosen entry point was problem-centered. Existing software architectures already exist in the industry, but do not currently satisfy company needs. Companies are looking to migrate their existing applications to a simple structure or build greenfield applications to cloud-based infrastructure. At the same time, they also desire to have

computing power, storage, and other resources as their business demands. Thus there is a need for the architecture to be highly available and scalable.

In this thesis, the design process started by defining the research problem and justifying the value of the artifactual solution in Section 3.1. The next step was to define the project objective, to design a highly available and scalable cloud architecture for a web application based on certain requirements, such as quality and functionality. These requirements were determined based on a literature review and data gathered from expert interviews with software architects. Four software development experts specialized in software architecture design were interviewed for this thesis. The goal of the interviews was to obtain concrete examples, design patterns, and requirements for a highly available and scalable architecture based on professionals' previous software projects. The requirements of this project are listed in Table 1 below.

Table 1: Requirements and constraints of the cloud architecture

Requirements and constraints:		
ID	**Requirement**	**Source**
01	The application should scale horizontally to manage the increasing simultaneous use of users.	Expert interviews 2021, Wolf & Henley 2017
02	The application should utilize a client-server software architecture pattern including web-tier, app-tier, and data-tier.	Barabanova et al. 2019
03	The application should follow a loose coupling approach regarding web-tier and app-tier.	Sharma 2015
04	The application's web-tier and app-tier should be able to adapt to increased demand.	Expert interviews 2021
05	The application's web-tier should be stateless.	Wolf & Henley 2017

06	The application must be able to ensure a highly available environment during any component failure.	Expert interviews 2021, Chatzakis 2016

In the design and development step, the artifactual solution is created. After this, the efficacy of the design is demonstrated using an example web application. The final steps of the design process are evaluation and communication. The architecture is evaluated using the scenario-based method and a weak market test. Finally, the research findings are presented and summarized.

4. Cloud architecture planning and design

This chapter is devoted to the third phase of the research process – design and development. Before starting the actual design, the cloud platform provider was selected. Next, the components of the cloud architecture were introduced and justified.

4.1 Select the cloud service provider

Before designing the cloud architecture, it is essential to select a suitable service provider, as the architecture will utilize their services. In Section 2.3., the three most popular cloud service providers and different characteristics of each cloud service were introduced. In this thesis, the cloud service provider was selected based on user-friendliness and versatility of services rather than focusing on pricing models and costs. As discovered in Subsection 2.3.3, Amazon Web Services has the most comprehensive and diverse service offering and therefore was chosen for this thesis. Furthermore, Amazon Web Services provides AWS' Well-Architected Framework (Figure 2) for cloud architecture design. This framework was used to ensure that the cloud architecture followed AWS' architectural best practices, especially in terms of reliability and performance. In addition, the comprehensive documentation of Amazon Web Services was utilized when justifying service choices and evaluating the success of the cloud architecture.

4.2 Amazon Web Services global infrastructure

In order to design an executable cloud architecture in AWS, it is important to understand its global infrastructure.

4.2.1 Data centers

AWS has massive data centers to provide a centralized, reliable, and certified operating environment for IT equipment and operations, such as data storing and application dissemination. A data center consists of physical hardware components, including servers,

routers, switches, operating systems, and network equipment (Sawehli & Xiang 2018). Typically, one data center houses tens of thousands of servers (Amazon Web Services 2021c).

4.2.2 Availability Zones

One of the key concepts in AWS infrastructure is an availability zone (AZ). The AZ consists of one or more data centers that provide redundant power and networking. Additionally, each AZ has individual power, cooling, and physical security systems. Availability zones are designed to be scalable and fault-tolerant, where customers can operate applications and databases. Fault isolation is implemented so that the AZs are physically separated from each other but still within 100km of each other. This way, AWS can isolate and protect its customers from natural disasters, such as lightning strikes, power outages, and tornadoes. AZs are interconnected using AWS' private high-bandwidth and low-latency networking (Beach 2014, 3-4). Customers are allowed to choose from which AZ they want to run their resources, the resources should run across multiple AZs to ensure resiliency, according to AWS' best practices.

4.2.3 Regions

A region is a geographical location that contains two or more isolated and physically separated AZs. As the previous chapter states, each AZ consists of one or more data centers, thus the concept of a region is used to cluster AWS' data centers (Amazon Web Services 2021c). According to AWS' architectural best practices, the following questions should be addressed before choosing a region:

1. What kind of data privacy laws are there across the region?
2. Should customer data be stored within the country, or is it more beneficial to store it outside the country?
3. Does the company comply with the local governance when choosing a particular region?

Along with data residency and regulatory compliance, a region should be selected based on where the customers are located to ensure lower latency. In addition, the availability of services must be taken into consideration, such as whether the desired services are available in that particular region.

4.3 Networking

In order to run resources in AWS, different networking layer components need to be configured and created, such as Virtual Private Cloud (VPC), subnet, and internet gateway.

4.3.1 Virtual Private Cloud

AWS provides private network space for their customers in the AWS cloud. This space is called Virtual Private Cloud (VPC). In the VPC, customers have complete control over the virtual networking environment, including security, connectivity, and resource placement. Thus, customers can define how the VPCs are communicating with each other across regions, availability zones, and accounts. VPCs can be deployed into one AWS region and the VPC will span across the AZs within the region. Thus, a VPC can host resources from any availability zone within the same region. Moreover, in order to create a VPC, customers have to specify a range of private IPv4 addresses by using Classless Inter-Domain Routing (CIDR) notation. Private IP addresses enable deployed resources to communicate with each other within the VPC. How can communication be enabled in practice? One possibility is to use route tables that can be used to direct the traffic between VPC resources. (Amazon Web Services 2021c)

4.3.2 Subnets

As the VPC is a large pool of shared resources, there is a need to divide it into smaller subsets. Hence, AWS provides a possibility to add one or more subnets in each AZ. A subnet is a segment within a VPC that contains a range of IP addresses. Such an approach allows a user to be able to isolate a group of resources within a particular VPC. (Amazon Web

Services 2021c) A VPC can be configured to use two types of subnets, public and private.

The main difference between these subnets is how they route the traffic. As the public term indicates, public subnets have a routing table entry to an internet gateway. This means that if the traffic comes from outside of the VPC, it will be transferred to an internet gateway. After that, the internet gateway is responsible for routing the traffic to the correct and available resources. In contrast, private subnet resources cannot be directly accessed from the public internet. Thus, the private subnet does not have a routing table entry to an internet gateway, making the resources within it more secure (Amazon Web Services 2021c). However, it is possible to use a Network Address Translation (NAT) gateway to allow private instances to connect with the services that are not within the VPC, though the NAT gateway will block incoming initiations from external services.

To conclude, architecture built on the AWS' Well-Architected Framework includes both private and public subnets. Resources that require communication to and from the internet are placed on the public subnet, and resources that should not be accessible from the internet are placed in the private subnet.

4.3.3 Internet Gateway

In order to allow traffic from outside of the VPC, an internet gateway needs to be created and attached to the VPC. The internet gateway is a VPC component that allows communication between resources inside of the VPC and the internet. This requires that the subnet route table has a route to direct internet-bound traffic to the internet gateway. (Fortinet 2021)

4.4 Security

As Subsection 4.3.1 states, AWS' customers will have full control over VPC security, allowing them to determine security settings for their resources. One way of doing this is to define security groups for resources. These groups act as virtual firewalls that enable control of both inbound and outbound traffic. According to Fortinet (2021), security groups operate

at the instance level and thus are attached to the instance's network interface. Hence, different security groups can be assigned to instances on the particular subnet. Furthermore, security groups are stateful, automatically allowing return traffic regardless of the inbound rules.

4.5 Caching

Based on expert interviews and literature review, caching can be identified as one of the basic characteristics of a highly available web application. The software architects who were interviewed for this thesis emphasized three key benefits of caching:

1. Improved web application performance
2. Reduced database load by optimizing and minimizing time-consuming database queries
3. Reduced response latency

The use cases of caching needed to be identified and determined in order to achieve the benefits presented above. Experts highlighted three main cases when data caching is highly recommended: when the data is static and frequently accessed, when the data queries are expensive and slow, and when the most recent version of the data is not urgent.

4.5.1 CloudFront

It is possible to utilize caching in different parts of the data journey, but the most common cache practices are at the edge or server-side. CloudFront is a global content delivery network (CDN) service of AWS which allows users to cache the data in different locations. AWS' global infrastructure provides multiple edge locations that are known as points of presence (POPs). POPs can be used for quickly serving content as they are closer to the end-users; thus the amount of time that it takes to deliver the content is reduced. Once the content is not available in the edge location it can be stored into a regional edge cache and provided from there. CloudFront's edge and regional caches fetch the content from the configured origin, such as Elastic Compute Cloud (EC2) instance or Simple Storage Service (S3) from

which the content is hosted. Moreover, the architecture of CloudFront service requires defining different settings, including expiration rules for the content and the locations in which the content needs to be cached. (Atkinson et al. 2020)

4.5.2 ElastiCache

According to the experts interviewed, the session information of a web application needs to be cached in order to provide a seamless user experience. Session management enables users to have a transparent authorization for each HTTP request to the web application, meaning that users don't have to log in repeatedly (Wedman et al. 2013). This kind of design can be considered as a part of stateless architecture, where the instance stores session information using an external caching service instead of its local memory. The stateless nature of the web allows for a convenient way to scale a web application horizontally by changing the number of instances within a single tier. (Wolf & Henley 2017, 101)

Amazon Web Services provides various ways to decouple the web application's session information from the instances. The best practice for storing session information is to create a distributed cache out of its instances. Wiger & Timalsina (2019) present an in-memory key-value data store called ElastiCache. They highlight that the architecture of an ElastiCache requires deployment of one or more cache clusters in order to store session information of a web application. A cache cluster consists of nodes that hold the session information which can be used by instances to perform various HTTP requests. In addition, if an instance goes down or a new version of a particular service needs to be released, this design principle makes it possible as it mitigates the impacts on end-users (Figure 5).

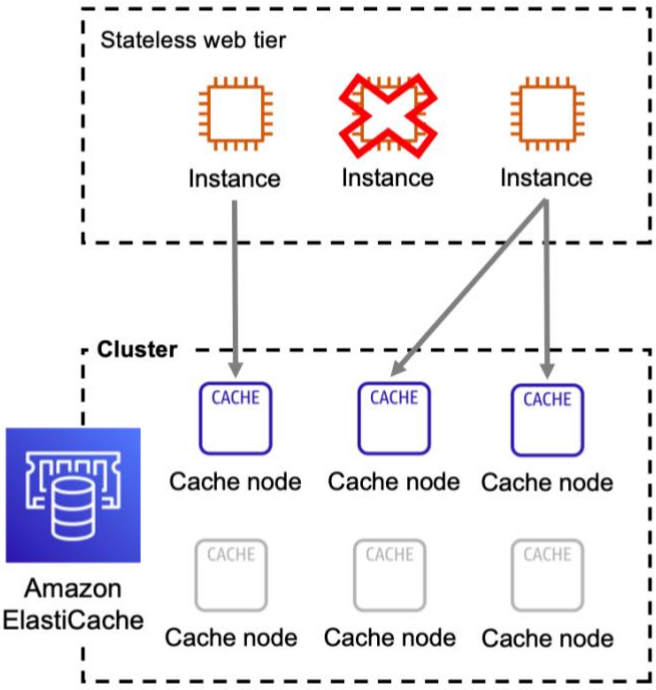

Figure 5. Session management with Amazon ElastiCache (based on Wiger & Timalsina 2019).

According to Wiger & Timalsina (2019), ElastiCache supports two open-source key-value engines, Memcached and Redis. Users can choose between these two options when launching an ElastiCache cluster. Memcached is designed for simple caching that has the ability to scale the cache horizontally as required. Memcached is the preferred choice when the primary goal is object caching. Redis, on the other hand, is a more comprehensive in-memory key-value store as it supports advanced data structures such as strings, lists, sets, hashed, and bit arrays. Another advantage of Redis over Memcached is that it supports replication, allowing Redis to be highly available due to the fact that users can achieve Multi-AZ redundancy. Because Redis has Multi-AZ redundancy, it allows feature allows automatic failover from the primary node to a replica during any event failure. (Wiger & Timalsina 2019)

4.6 Designing for high availability

After creating a base for the cloud architecture, the planning and designing process can start. Malkawi (2013) states that in order to achieve a highly available environment, software downtime needs to be minimized as much as possible. Secondarily, the elimination of a single point of failure (SPOF) plays a key action towards this resiliency. A highly available environment is able to restore the failed service and limit the impacts so that they do not cause visible inconvenience for end-users. Fault recovery and switching to a secondary source will cause downtime for a web application, therefore, it is important to select the appropriate components for the application and ensure adequate resources.

The design of the cloud architecture follows four design principles for high availability and scalability defined by Sharma (2015) and Amazon Web Services (2021d):

- Design for failure consideration
- Utilize multiple availability zones
- Isolate components
- Utilize auto-scaling

These design principles are explained and illustrated in the following sections.

4.6.1 Considering failures in architecture design

First, Sharma (2015) and Amazon Web Services (2021d) emphasize that when an organization moves to a cloud infrastructure solution, it has to plan for failure and design backward. Such as ensuring that any individual failure would have a fallback within the architecture. This kind of redundancy is necessary in order to avoid SPOFs (Figure 6).

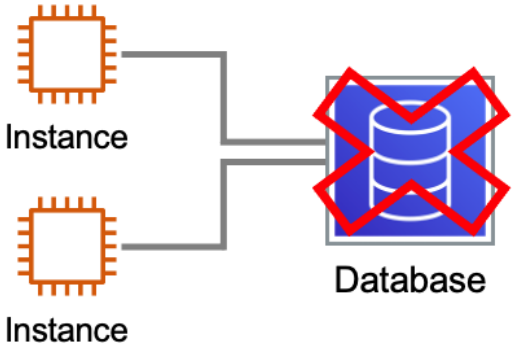

Figure 6. Anti-pattern - Single point of failure (based on Dutta et al. 2014).

Based on the architect interviews, the following hypothetical scenarios can be used to help design redundancy in cloud architecture:

- What happens if the X component fails?
- What happens if X service is unavailable?
- What are the consequences of a failure in the X component?
- How long does it take to recover from X component failure?

By implementing redundancy, the failure of an entire web application can be prevented since the SPOFs have been taken into account and can be avoided, as in Figure 7.

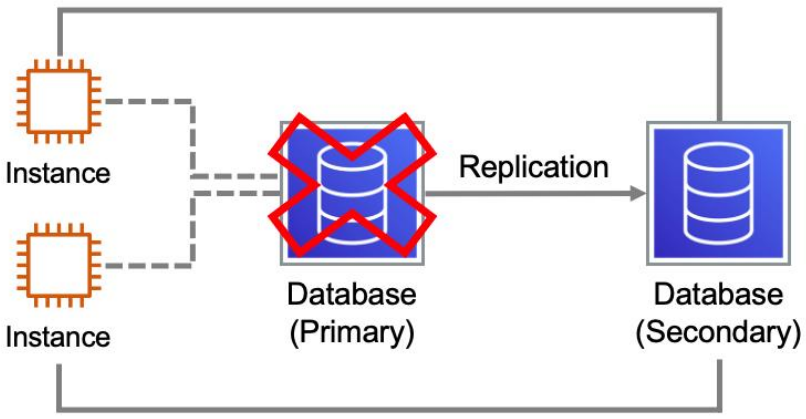

Figure 7. Best practice – Redundant implementation (based on Dutta et al. 2014).

4.6.2 Utilizing multiple availability zones

According to Weiss and Furr (2019), the second most significant design principle while designing a highly available cloud architecture is the use of multiple AZs. AZs are designed independently from each other and therefore they have separate networking and power connectivities. This raises the question as to how many AZs should organizations utilize. According to Eliot and Livingstone (2021), it is highly recommended to deploy at least two AZs to run application resources in an AWS region. In this scenario, the web application will continue to run in one AZ if the other becomes unavailable. According to Tomsen Bukovec (2018), the entire AZ or AWS region has never gone down completely. Thus by using two availability zones, organizations can mitigate the risks of technical and natural disasters. In conclusion, in order to achieve a highly available environment, resources must be allocated across multiple AZs within the region.

4.6.3 Taking loose-coupling into account in architecture design

The third design principle regards isolating the components. Sharma (2015) states that all the components should be decoupled and isolated from each other. He also highlights that tightly coupled solutions increase the complexity of an architecture due to the fact that each component needs to be aware of other components. This principle is highlighted in Figure 8. The increase of complexity can be seen especially when resources are added to the design.

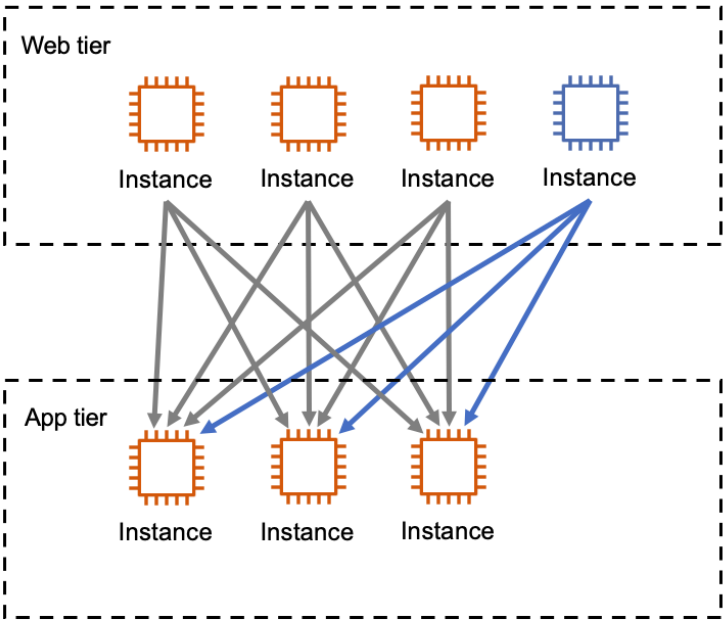

Figure 8. Tightly coupled design (based on Sharma 2015).

Connection to the final design principle of auto-scaling, components need to be loosely coupled in order to scale up or down without any dependencies. Decoupled cloud architecture can be designed by utilizing a load balancer between tiers as shown in Figure 9. This figure also illustrates that when instances are added, it does not require adding any new connections from the app tier perspective. Additionally, change or failure impacts of one instance are reduced since that instance does not affect other instances.

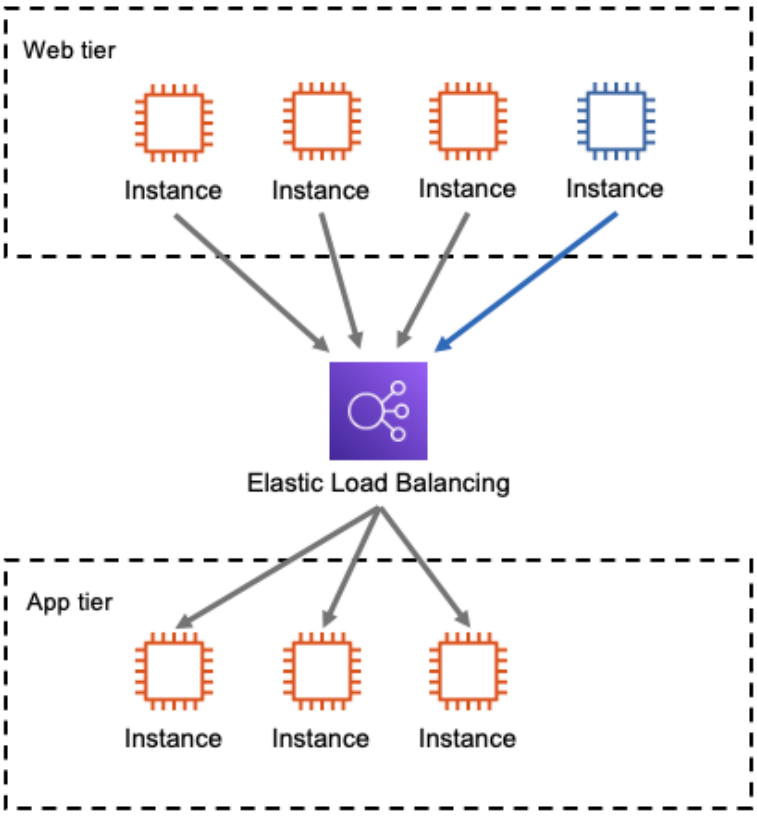

Figure 9. Loosely coupled design (based on Sharma 2015).

Amazon Web Services' platform provides a service called Elastic Load Balancing (ELB) for distributing incoming traffic across multiple resources and AZs. ELB enables a single point of entry (SPOE) where user traffic is transferred to a certain access point. Afterwards, these requests are load balanced across the resources if defined security requirements are met. ELB can be configured to distribute the traffic using round robin (RR) or least outstanding requests (LOR) scheduling algorithms. Concerning the features, ELB supports HTTP, HTTPS, TCP, and SSL protocols. ELB can also be external-facing or internal-facing. The main difference between the two is that the external-facing load balancer can be accessed from the public network, whereas the internal-facing load balancer is meant for private use only. Additionally, ELB performs health checks on all deployed instances to check their status. Once an instance appears unavailable, the ELB recognizes it and stops routing the requests for those unhealthy instances. Only those instances which return a status code of '200 OK' will receive requests from ELB. By default, the health check interval is set to 30 seconds, but it is possible to reduce or increase it from five seconds up to 300 seconds. (Amazon Web Services 2021c)

In terms of ELB options, Amazon Web Services (2021c) recommends using either an Application Load Balancer or a Network Load Balancer. The Application Load Balancer (ALB) functions at the application level and supports load balancing of HTTP and HTTPS traffic, whereas the Network Load Balancer (NLB) functions at the connection level and can load balance TCP and UDP traffic. ALB enables advanced load balancing, supporting path conditions by configuring rules for a listener to catch the URL of the request. This function enables a web application to be divided into smaller services and direct the traffic to desired services based on the content of the URL. In comparison to ALB, NLB is optimized to handle even non-standard traffic. This means the NLB is able to handle tens of millions of requests per second. (Amazon Web Services 2021c)

4.7 Considering scalability in architecture design

As a load balancer distributes the traffic between different tiers, there is a need to scale the number of instances within the tiers according to demand. This can be done by utilizing an auto-scaling service within the cloud architecture, as shown in Figure 10. The following figure illustrates a case where the minimum size is one instance, the number of desired instances is two, and the maximum size is four instances (Amazon Web Services 2021d). According to Liao et al. (2015), auto-scaling service has a significant role in cloud computing, especially when ensuring elasticity. They highlight that these services are mainly used to respond to growing workload demand and/or terminate unused instances. In other words, auto-scaling services strive to automate the provisioning of the instances without changing the design. Auto-scaling has an additional benefit of cost reduction, as it can be optimized to scale resources up and down intelligently so that the right amount of capacity is always available.

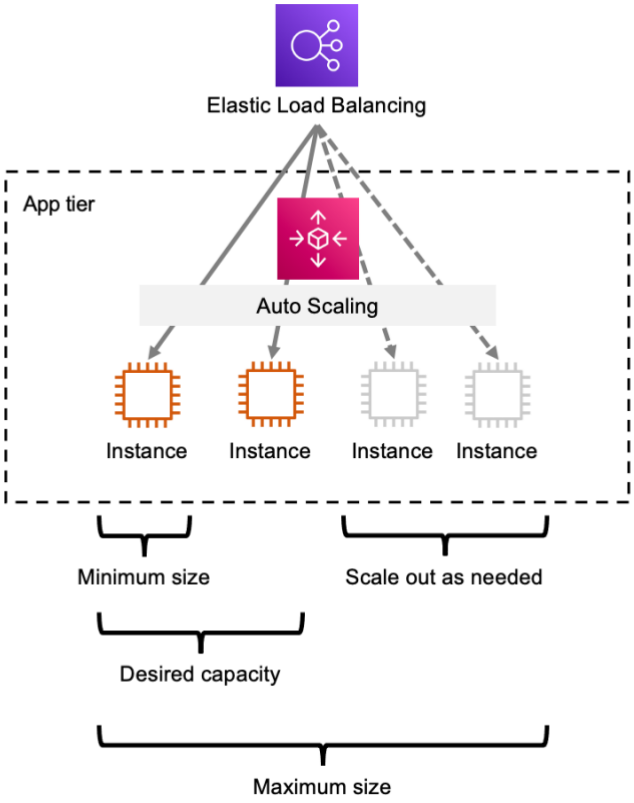

Figure 10. Maintaining the number of instances with auto-scaling (based on Amazon Web Services 2021d).

According to Amazon Web Services (2021d), there are four types of scaling: scheduled, dynamic, predictive, and manual. Scheduled scaling means turning on resources when there is an anticipated need for their use. For example, development and testing environments can be optimized so that they are up and running only during business hours. Dynamic scaling can be used in unpredictable workload cases. The purpose of dynamic scaling is to ensure enough computing power is available to match unexpected demand. Predictive scaling works by using machine learning algorithms to detect changes and anomalies in temporal trends. These algorithms allow future traffic to be predicted, such as regularly occurring spikes. Predictive scaling allows companies to provide more accurate capacity provisioning to their customers. Finally, manual scaling allows the user to adjust the number of instances. This type of scaling is suitable for cases where the capacity needs to be held in a fixed number of instances. Depending on user needs, it is possible to combine the scaling methods presented above. (Amazon Web Services 2021d)

5. Highly available and scalable cloud architecture

This chapter complements and summarizes the third phase of the research process – design and development. The components and design principles presented in Chapter 4 are combined to design the cloud architecture, which is enriched with the data gathered from interviews with software architects and existing literature. Additionally, the goal is to design a cloud architecture, which covers the requirements and complies with the constraints defined in Table 1.

5.1 Visualization of the cloud architecture

Figure 11 was created by following the design principles described in Section 4.6. It illustrates a highly available and scalable software architecture for a web application based on the requirements specified in Section 3.2.

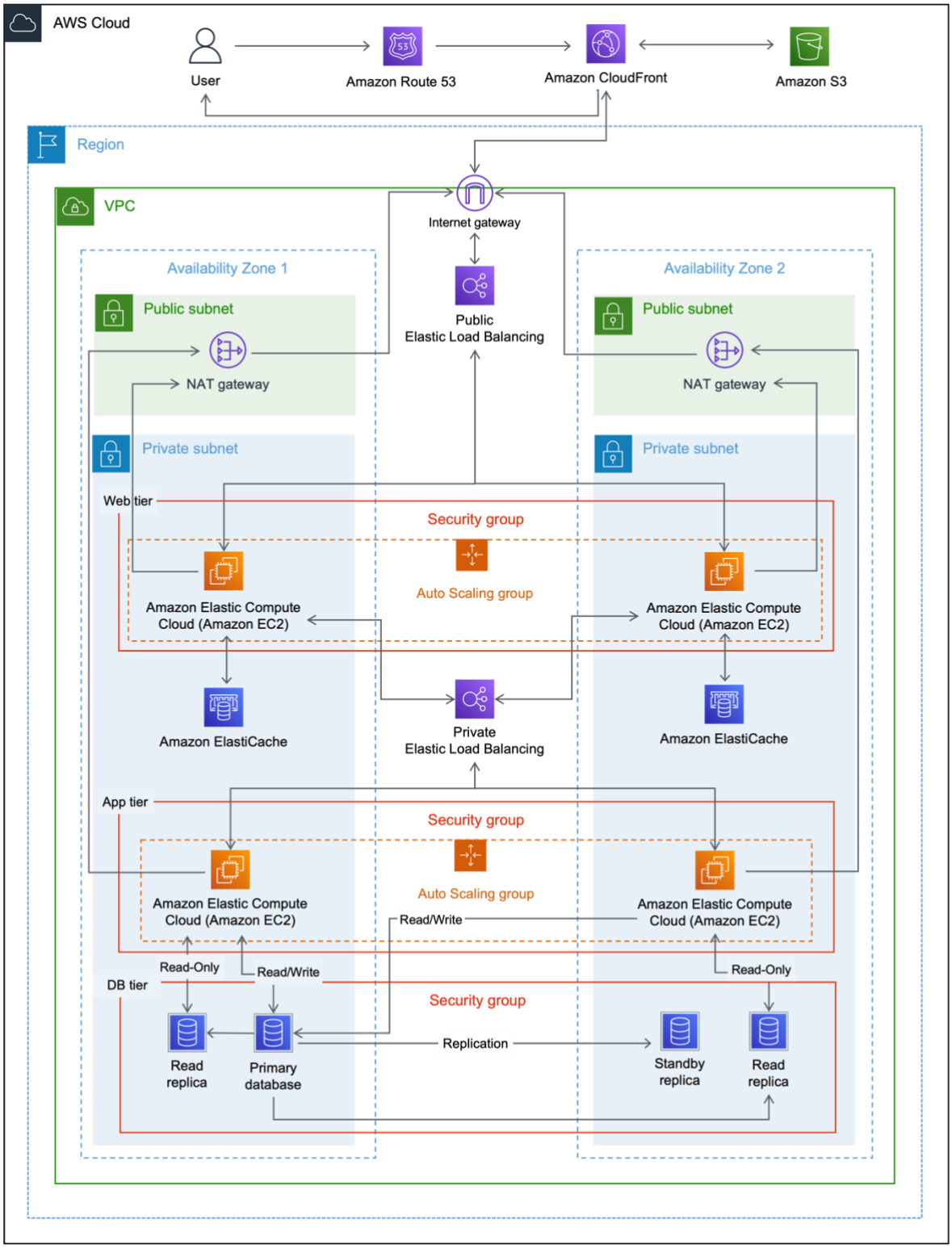

Figure 11. Highly available and scalable cloud architecture

First, the region is defined. Within the region, a VPC is created with Domain Name System (DNS) support. Second, the internet gateway is created and attached to the VPC to allow

communication between the instances in the VPC and the internet. Integral to the highly available nature of this architecture is two availability zones, which deploy and run resources. Each availability zone consists of a public and private subnet. The private subnet follows a standard three-tier architecture with security at each level, including web, app, and DB tiers. The web tier provides the user interface and handles the communication, the app tier houses the business logic of the web application, and the DB tier manages and stores the data. In contrast, the public subnet does not include any resources except managed services and NAT gateways. Managed services such as load balancers are used for distributing the traffic to the private subnet across multiple AZs. NAT gateways are deployed to allow instances in a private subnet to connect services that are outside of the VPC. In the third step, caching is implemented in the web tier and outside of the region. Static content is cached outside of the region and provided to the end-users from there. Web tier caching is used for storing and providing user session information.

5.2 Data flow

When hypothesizing user data flow, there are a number of events the data will pass through. As the user tries to access the web application by typing a domain name into the web browser, this request is forwarded to Amazon Route 53. Route 53 is a DNS service that resolves the given domain into its corresponding IP address. It routes the traffic to a content delivery network service called Amazon CloudFront. The user gets all the static content from Amazon CloudFront, but dynamic content requests are forwarded to the public Elastic Load Balancer. The ELB distributes user requests to the web tier across multiple availability zones and resources. The session information related to the user is decoupled from the instance by using a web tier’s caching service called ElastiCache. The auto-scaling group ensures the application has allocated the right amount of compute resources based on the number of user requests. Then the request is sent to the private ELB which distributes the traffic to the resources within the app tier. Depending on the type of request, it is either forwarded to the primary database or read replicas.

6. Demonstration of the cloud architecture

This chapter demonstrates the highly available and scalable aspects of the designed cloud architecture. As presented in Figure 11, in order to demonstrate the architecture in practice, the following services need to be configured:

- VPC
- NAT gateway
- Load balancer
- Auto-scaling group

To increase transparency, these services are implemented through a management console instead of using automated templates.

6.1 Creating a VPC

As stated in Subsection 4.3.1, VPC is a virtual network that allows customers to take over the networking environment such as security settings, subnets, network gateways, and resource launching. The basis of the VPC is created by deploying it with the following:

- An internet gateway
- A public subnet in two AZs
- A private subnet in two AZs
- A NAT gateway in two AZs

The VPC creation process starts by opening the VPC console, naming it, and choosing the desired range of IPv4 addresses in the form of a Classless Inter-Domain Routing (CIDR) block. The values used for the VPC creation can be found in Appendix 1 of this thesis.

After the VPC is successfully created, the next step is to create subnets within the VPC. In the subnets dashboard, a user selects the newly created VPC. After that, the subnet settings

such as name, AZ, and IPv4 CIDR block need to be configured. The values used for the subnet can be found in Appendix 1.

The architecture consists of two types of subnets, private and public, thus this process needs to be done twice to both AZs. Private and public subnet pairs should be located in the same availability zone. In addition, the public subnet's "auto-assign public IPv4 address" setting needs to be enabled so that an EC2 instance provisioned in this subnet will be assigned a public IP address. The private subnet's IPv4 CIDR block can be configured as 10.0.2.0/23 to make it larger, due to the fact that most of the resources are placed in the private subnet.

After the subnets are deployed successfully within the VPC, the next step is to connect two of the subnets to the internet via an internet gateway. This can be done by creating an internet gateway and attaching it to the created VPC. In order to route network traffic to its desired destination, each subnet in a VPC must be associated with a route table. An automatically generated route table is the main route table that is generated when the VPC is created. This route table can be named "private" due to the fact that the route (10.0.0.0/16) allows subnets within the VPC to communicate with each other. To set up a public route table, see Appendix 1. Additionally, a route to direct internet-bound traffic to the internet gateway needs to be configured as shown in Appendix 1. The route table is connected to the public subnets by editing the subnet associations to make them officially public.

The next step is to deploy a NAT gateway on each public subnet. As the NAT gateways are placed in each public subnet, an elastic IP address needs to be associated with the NAT gateway when they are created. Because a NAT gateway is deployed to two public subnets, two elastic IP addresses need to be allocated. The process of creating a NAT gateway is straightforward – defining a name, selecting a subnet, setting the connectivity type to be public, and selecting an elastic IP address that was previously created. In order to use the NAT gateway, it is necessary to edit the private route table. To ensure that instances in the private subnets have access to the internet, a new route needs to be configured with a target of the created NAT gateway and destination of 0.0.0.0/0.

In order to control the inbound and outbound traffic, the security groups need to be configured for services, as shown in Figure 11. This can be accomplished via the security

groups dashboard by defining security group name, description, VPC name, and rules for inbound and outbound traffic. For demonstration purposes, the created security groups are available in Appendix 1. These security groups are used for services such as the load balancer and EC2 instances which will be integrated in upcoming sections.

6.2 Creating a load balancer

Figure 11 shows, the resources are launched in two availability zones and each tier includes multiple instances. A load balancer needs to be created in order to distribute the traffic across these components. Additionally, a load balancer can be configured to do health checks so that traffic is distributed only for healthy instances. For this project, only one load balancer is needed, and it was created and linked to the web tier that is placed in the private subnet.

The load balancer creation process starts with defining a load balancer type and configuring its basic information, such as network mappings, security groups, listeners, and routings. In this project, the load balancer is placed in the public subnet with the settings that are available in Appendix 1.

After the load balancer is created, it can be attached to an auto-scaling group. Setting an auto-scaling group first requires the creation of a launch configuration or launch template. In this project, the launch configuration is selected for defining the qualities of EC2 instances used in the auto-scaling group.

6.3 Creating a launch configuration

A launch configuration is used by an auto-scaling group for defining an EC2 instance-level setting, such as Amazon Machine Image (AMI), instance type, and security group. The launch configuration set up can be found in Appendix 1.

To verify the high availability of the architecture, the user data script was employed in the launch configuration. This script allowed the EC2 instance changes to be visible to the user

via the web browser. In order to execute the user data described above, it is necessary to allow EC2 instances to connect services that are outside of the VPC.

6.4 Creating an auto-scaling group

After creating the load balancer and launch configuration, an auto-scaling group can be created. As stated in Section 4.7, an auto-scaling group is used to automatically launch or terminate EC2 instances horizontally according to demand and health checks. Furthermore, it is used to automate the distribution of the EC2 instances across availability zones. The auto-scaling group detailed creation process is described in Appendix 1.

After the auto-scaling group is created, two EC2 instances are running–one in each availability zone. This configuration ensures the application is highly available and resilient.

6.5 Testing the demo web application

This chapter is dedicated to proving that the designed architecture satisfies the desired objectives. To confirm that the demo application is highly available, the first step is to check that there are two registered targets under target groups as shown in Figure 12.

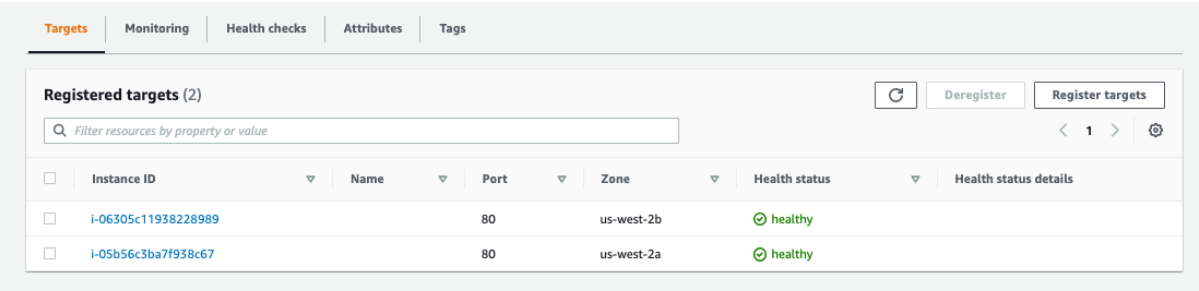

Figure 12. Target groups

The second step is to make sure that the instances pass the load balancer health checks, ensuring the launched instances are running correctly.

The demo web application is tested through the load balancer which distributes the user traffic to one of the EC2 instances. As mentioned above, the high availability can be illustrated in visual format by copying and pasting the load balancer's DNS name to the web browser. As shown in Figure 13, the load balancer forwards the user request to the instance i-06305c11938228989 which operates in the availability zone us-west-2b.

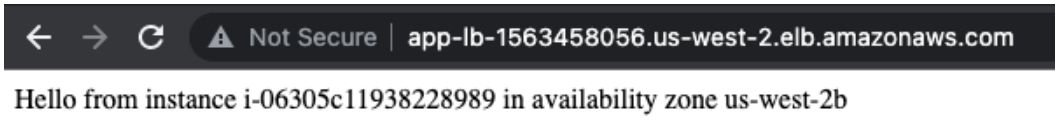

Figure 13. User request forwarded to i-06305c11938228989 instance

If the web browser is reloaded, it can be observed that the target instance and the availability zone change between the two instances, as shown in Figure 14. This observation highlights the web application remains highly available even if one of the EC2 instances fails.

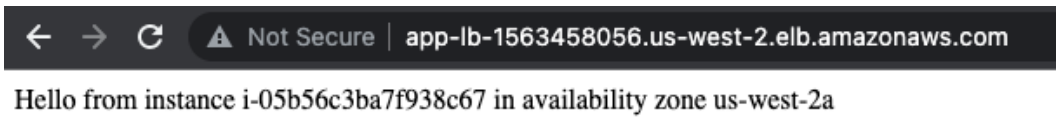

Figure 14. User request forwarded to i-05b56c3ba7f938c67 instance

It can be proven that the scaling is working as intended by terminating one of the running EC2 instances. As the instance is terminated, (Figure 15) the load balancer activates due to the fact that the instance is not passing the health checks. Thus, the load balancer stops sending requests to the unhealthy instance and continues to route requests to the healthy one instead. Furthermore, the demo web application continues to remain even if one of the instances fail.

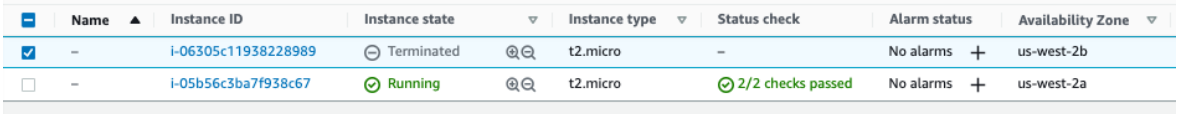

Figure 15. Terminated instance

After the auto-scaling group registers that one of the instances has failed, it automatically launches a new instance, as shown in Figure 16. This is due to the fact that the launch configuration is set to maintain two instances running at all times. The automatic launch of a new instance can be proven by reloading the EC2 dashboard. The user will observe a new instance with a new instance ID was created to the same availability zone where the failed instance existed previously.

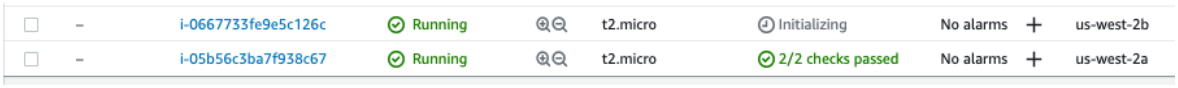

Figure 16. Auto-scaling initializing a new instance automatically after a failure

After the new instance passes the health checks, the load balancer continues to route the requests between the two availability zones. This can be proven by reloading the web browser and observing that the instance and the availability zone are changing. Thus, this demonstrates that the demo web application is highly available and scalable.

The figure below summarizes the flow of the traffic and illustrates the used services:

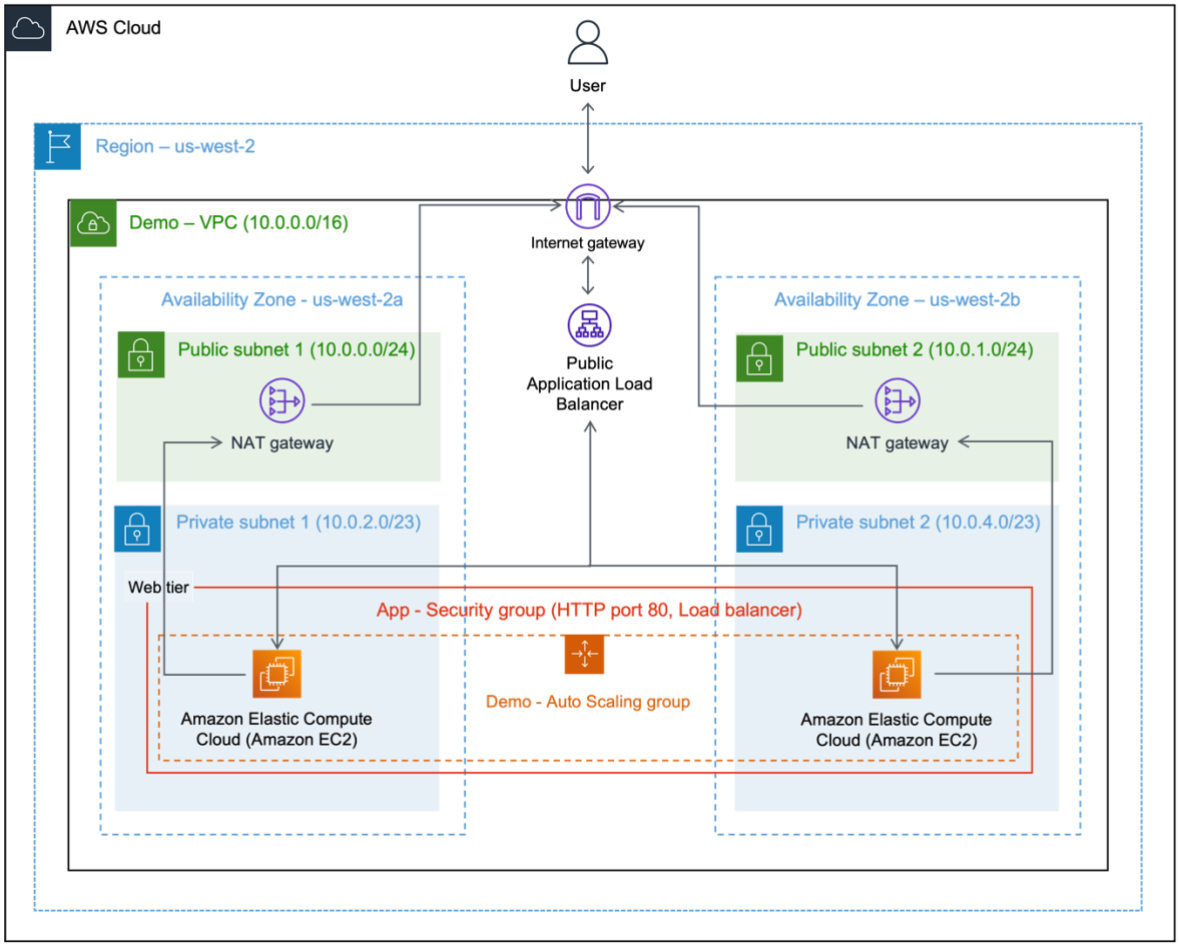

Figure 17. The cloud architecture used in the demonstration

The flow starts when a user sends a request to the public load balancer that is linked to the internet. Next, the load balancer selects one of the private EC2 instances based on defined scheduling algorithms and health check results. After this, the EC2 instance returns the demo web application to the load balancer, which then returns it to the user's web browser.

7. Evaluation of the cloud architecture

This chapter evaluates the created cloud architecture design. The first part of the evaluation is done by using the scenario-based method. The second part of the evaluation is done by adopting a constructive research testing method — a weak market test.

7.1 Scenario-based evaluation

Scenario-based evaluation method is performed using steps that explain the inputs, outputs, or actions that occur during each scenario. Scenario-based methods are used to evaluate whether the number of highest-ranked architecturally significant requirements (ASRs) are addressed and satisfied through each scenario. According to Qin et al. (2008), the ASRs need to be written unambiguously as a stimulus-response pair. The stimulus can be seen as a condition that must be met to trigger a scenario. Additionally, the expected response that the web application should provide to it needs to be described. Next, the functional components need to be identified when certain stimulus conditions occur. Finally, the expected web application response needs to be provided.

The following items were identified as ASRs and were selected for the scenario-based evaluation:

- The application's web-tier and app-tier should be able to adapt to increased demand.
- The application's web-tier should be stateless.
- The application must be able to ensure a highly available environment during any component failure.

Table 2: Evaluation of the designed architecture using a scenario-based evaluation method

ID	Requirement
04	The application's web-tier and app-tier should be able to adapt to increased demand. **Stimulus:** As a result of a successful marketing campaign, hundreds of users want to access the web application on Friday from 9 a.m. to 5 p.m. **Response:** The developer creates a scheduled action within the auto-scaling group service, so it increases the capacity on Friday from 8 a.m. to 6 p.m. to handle the increased demand. **Scenario walkthrough:** 1. The company is marketing an upcoming campaign related to its products on its web application. 2. The company makes predictions about the upcoming traffic based on weekly usage data and traffic from previous campaigns. 3. The developer creates a scheduled action within the auto-scaling group based on the predictions by defining the following: - Name - Desired capacity, min, max - Recurrence - Time zone - Specific start time 4. The auto-scaling group adjusts the number of instances based on demand and health checks. 5. The web application serves all users, despite the high traffic surge of the campaign day.

05	The application's web-tier should be stateless. **Stimulus:** The user wants to continue using the web application after an EC2 instance shuts down without the user logging in again. **Response:** A replacement EC2 instance is created. This new instance gets the session information related to the user from an in-memory key-value data store called "ElastiCache" and continues to serve the user. **Scenario walkthrough:** 1. The user logs in to the web application. 2. The session information related to the user is immediately saved to the ElastiCache service. 3. The user suddenly receives an error message stating "the connection to the web application has been lost". 4. The auto-scaling group automatically launches a new instance to replace the failed one. 5. The load balancer registers an unhealthy instance and routes the user requests to a free and healthy instance. 6. The user decides to refresh the web application view and continues to use it without having to log in a second time.

06	The application must be able to ensure a highly available environment during any component failure. **Stimulus:** One of the EC2 instances' software needs to be updated, and therefore it needs to be stopped. **Response:** An auto-scaling group launches a new instance to meet the desired number of EC2 instances. **Scenario walkthrough:** 1. The software developer receives a notification from AWS that one of the EC2 instances' software is not up to date, and thus it needs to be updated. 2. The software developer recognizes that the software update requires terminating the EC2 instance. 3. The software developer terminates the EC2 instance and makes the necessary software updates to it. 4. The auto-scaling group performs health checks of the instances and identifies an unhealthy EC2 instance, due to the fact that it was terminated. 5. The auto-scaling group automatically replaces the terminated EC2 instance with a new one. 6. The load balancer recognizes the terminated instance and thus forwards the requests only to healthy instances. 7. The web application remains highly available.

As Table 2 shows, the architecture meets and covers the ASRs. Additionally, the architecture executes the required behavioral functions in each scenario, and thus meets the characteristics of proper cloud architecture. Furthermore, the scenario-based evaluation results prove that the services and their interdependence have been taken into account while designing the architecture.

7.2 Weak market test

The weak market test is the first level of a three-level market-based validation approach proposed by Kasanen et al. (1993). The test is conducted with software architects to determine the validity of the designed cloud architecture, and to understand if software architects would adopt it. To conduct the test, the following questions were asked:

1. Do the requirements presented in Subsection 3.2. meet the criteria of highly available and scalable cloud architecture?
2. Does the designed cloud architecture cover the requirements presented in Subsection 3.2.?
3. Could the designed cloud architecture be used as a basis for software projects?

Three experts participated in the weak market test. Two of them work as senior software architects in software consultancy and one works in-house as a cloud engineer. In general, the experts provided similar and positive answers to the above questions. They agreed that the presented architecture is highly available and scalable and its design met the requirements in Subsection 3.2. A key finding of the interviews was that experts were willing to adapt the designed cloud architecture and its design patterns in their work. They stated that the cloud architecture includes the necessary components to achieve a highly available and scalable architecture. Moreover, they highlighted that the architecture works as a basis for the design process and gives space needed for project specific design decisions.

One of the considerations raised by the experts was that in some cases it can be beneficial to use two-tier architecture instead of three-tier architecture, due to the fact that the two-tier architecture often reduces the complexity and costs of design. According to the expert, “having fewer moving parts makes it easier to figure out how the software behaves and which components are linked to each other”. He added that additional layers can also increase the risk of component failure, though he noted that three-tier architecture also has its advantages. For example, in cases where a company has a legacy application from which data is fetched, a user interface can be deployed to the app tier that allows caching operations. This approach makes the application more efficient, increasing its usability and performance. Three-tier architecture also allows for more optimal scaling capabilities as it can scale

horizontally at each tier to handle incoming requests. In summary, the weak market test was successful. The designed cloud architecture met all requirements, and the experts highlighted they will consider the characteristics of availability and scalability in future projects. It is possible the experts may employ the architecture presented in this thesis as the basis in their work moving forward.

8. Discussion

This chapter is devoted to the last phase of the research process: communication. The results are presented and discussed with reference to the objective of the thesis, which was to design a highly available and scalable cloud architecture for a web application.

As the demonstration chapter proves, by using the created cloud architecture design patterns, it is possible to achieve a highly available and scalable web application environment. The patterns presented in the thesis are versatile. They can be used for situations such as greenfield web application architecture or when migrating an existing application to a cloud service platform. This is due to the fact that the designed cloud architecture and its building blocks succeed in providing a solid foundation for the web application in terms of availability and scalability. It should be noted that further research in this area is possible, for example, considering aspects such as cost optimization and sustainability of the architecture.

The designed cloud architecture can be expanded with services necessary for a web application, making it a good candidate to serve as the basis of any web application. This allows companies to focus more on other functional requirements of the application.

The design principles and best practices regarding the availability and scalability of the web application environment have been investigated and presented in a comprehensive manner. Precise steps were provided to reproduce the creation process of each individual service. By utilizing the research and steps provided in this thesis, companies can increase the efficiency and reliability of their web application architecture design process. This thesis also helps to identify and fill the knowledge gaps regarding the different services, their use cases, and their interdependencies.

The requirements were selected on the basis that they are independent of the core nature and features of a web application. The reason for this is that the requirements were intended to be universal and reusable, allowing companies to utilize them to achieve a highly available

and scalable cloud environment for their web application. Finally, the reliability of the architecture was assessed to ensure quality and minimize the risks posed by the ASRs.

Although the designed cloud architecture covers the requirements in terms of availability and scalability, it still has limitations. Limitations include:

- Primary use cases are greenfield projects or migrating an existing application to a cloud platform
- Best practices being primarily related to AWS' Well-Architected Framework
- Deployment configurations are AWS specific

First, the primary use cases of the designed architecture are limited to greenfield projects or when migrating an existing application to a cloud platform. This is because the architecture provides basic building blocks in terms of scalability and availability - it would be more difficult in the case of trying to scale an existing software. This is because when extending an architecture, the structure, relations, and dependencies on existing components must be taken into account. Moreover, it must be ensured that the specified services are available in the environment in which the software is running. Second, the next limitation is because the design principles and best practices used in the architecture are primarily related to AWS' Well-Architected Framework, making them less universal. For example, some of the design principles, such as using ElastiCache, are highly AWS related and would not be replicated using a different cloud service. To address this limitation, however, the thesis provides general guidelines and design principles that can be used for any software architecture. Finally, due to the choices made in the study, the architecture is implemented in the AWS' cloud environment, and thus the deployment configurations are AWS specific.

9. Conclusions

This thesis provided insight into cloud computing, cloud service models, cloud architecture, and the most known cloud service providers. Throughout the thesis, the design principles and best practices were evaluated to solve the research problem. The role of the software architecture was emphasized as it informs high-level design choices, and it is responsible for meeting the requirements.

The thesis provided examples of general resilience and scalability issues and guidelines on aspects that need to be considered and avoided when designing a cloud architecture. These examples allow companies to understand the nature of a highly available and scalable cloud environment. In addressing a knowledge gap in the sector, the guidelines presented create a roadmap for software architects, ensuring a level of consistency across different types of architecture projects. This thesis increases sector knowledge by providing detailed explanations about services, their possible use cases, and justifications regarding design choices. These guidelines are useful for greenfield projects or when migrating an existing application to a cloud platform, because they cover the key requirements regarding availability and scalability.

The main goal of this thesis was to design a highly available and scalable cloud architecture for a web application. The process of designing the cloud architecture involved three essential steps: design, demonstration, and evaluation. The design phase provided a comprehensive overview of the selected cloud service provider's global infrastructure and essential services. Additionally, the selection of services was justified and how these services were interrelated with availability and scalability design were explained. The designed cloud architecture was demonstrated by using a demo web application to illustrate the availability and scalability features. The demonstration phase was used to prove that the selected design patterns allowed the web application to achieve the goals of the thesis. Finally, the designed cloud architecture was evaluated by using the scenario-based evaluation method and the weak market test. The main goal of the scenario-based evaluation process was to prove that the architecture meets the ASRs and executes the required behavioral functions in each scenario. The weak market test was conducted to find out whether software architects are

willing to adopt the designed cloud architecture in their projects. The evaluation was also used to validate the architecture decisions, discover good architecture practices, and identify potential risks.

Existing software architectures already exist in the industry, but they do not currently satisfy company needs. This is due to the fact that every company developing a web application has application-specific requirements. Designed architecture is not universal, even though it may contain commonly used design choices and elements. Furthermore, ensuring and analyzing resilience and scalability has not been a key priority for existing architectures. The cloud architecture designed in this thesis can be applied as the basis of any kind of high-quality web application, as it focuses on their two key characteristics: scalability and availability.

Cloud computing is still in its infancy. Nevertheless, it can already provide significant opportunities for companies to run their business in a more user-friendly and efficient way than ever before. This thesis enhances the existing cloud computing knowledge base, enabling the design and implementation of high-quality cloud architecture. However, cloud architecture can be further developed by considering other important aspects such as sustainability, and cost optimization.

References

Amazon Web Services, 2021a. Amazon Virtual Private Cloud. [WWW Document]. URL https://docs.aws.amazon.com/vpc/latest/userguide/vpc-ug.pdf#what-is-amazon-vpc (accessed 12.11.2021).

Amazon Web Services, 2021b. AWS WellArchitected Framework . [WWW Document]. URL https://docs.aws.amazon.com/wellarchitected/latest/framework/wellarchitected-framework.pdf#welcome (accessed 14.10.2021).

Amazon Web Services, 2021c. Overview of Amazon Web Services. [WWW Document]. URL https://docs.aws.amazon.com/whitepapers/latest/aws-overview/aws-overview.pdf#global-infrastructure (accessed 25.11.2021).

Amazon Web Services, 2021d. What is Amazon EC2 Auto Scaling? [WWW Document]. URL https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html (accessed 20.10.2021).

Anandamurugan, S., Priyaa, T., 2014. Service Oriented Architecture, Computer Science, Technology and Applications. Nova Science Publishers, Inc, Hauppauge, New York.

Appavoo, J., Uhlig, V., Scalability : The Software Problem 4. [WWW Document], URL https://cs-web.bu.edu/~jappavoo/Resources/Papers/stmcs07-scal.pdf (accessed 20.09.2021).

Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M. 2009. Above the Clouds: A Berkeley View of Cloud Computing 25. [WWW Document], URL https://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf (accessed 17.09.2021).

Atkinson, L., Bogacz, K., Coleshill, I., Morrow, A. 2020. Amazon CloudFront for Media Best Practices for Streaming Media Delivery. [WWW Document], . URL https://d1.awsstatic.com/whitepapers/amazon-cloudfront-for-media.pdf (accessed 12.11.2021).

AWS Well-Architected - Build secure, efficient cloud applications [WWW Document], Amazon Web Services, Inc. URL https://aws.amazon.com/architecture/well-architected/ (accessed 19.09.2021).

Bala, B., Gill, B. 2021. Magic Quadrant for Cloud Infrastructure and Platform Services. [WWW Document], . URL https://www.gartner.com/doc/reprints?id=1-271OE4VR&ct=210802&st=sb (accessed 14.10.2021).

Barabanova, I.A., Kravets, O.J., Tkalich, S.A., Mutin, D.I., 2019. Analysis of the intermediate layer work in the three-tier architecture “client-server” of automation engineering problems. IOP Conf. Ser.: Mater. Sci. Eng. 537, 032011. https://doi.org/10.1088/1757-899X/537/3/032011

Bass, L., Clements, P., Kazman, R., 2015. Software architecture in practice, 3rd ed. ed, SEI series in software engineering. Addison-Wesley, Upper Saddle River, NJ.

Beach, B., 2014. Pro PowerShell for Amazon Web Services: DevOps for the AWS Cloud. Apress.

Brown, S., 2016. Software Architecture for Developers: Volume 1 - Technical leadership and the balance with agility. Technical leadership and the balance with agility 133.

Buxmann, P., Diefenbach, H., Hess, T., 2013. Software as a Service: The Application Level of Cloud Computing, in: Buxmann, P., Diefenbach, H., Hess, T. (Eds.), The Software Industry: Economic Principles, Strategies, Perspectives. Springer, Berlin, Heidelberg, pp. 169–190. https://doi.org/10.1007/978-3-642-31510-7_6

Chatzakis, A., 2016. Architecting for the Cloud: AWS Best Practices. [WWW Document], URL https://aset.az.gov/sites/default/files/media/AWS_Cloud_Best_Practices.pdf (accessed 17.09.2021).

Dutta, S., Barua, S., Sen, J., 2014. Auto Default Gateway Settings for Virtual Machines in Servers using Default Gateway Weight Settings Protocol (DGW). International Journal of Wireless & Mobile Networks 6, 133–143. https://doi.org/10.5121/ijwmn.2014.6511 Eliminating Single Point of Failure and Data Loss in Cloud Computing, 2014. 3, 3.

Eliot, S., Livingstone, A., Disaster Recovery of Workloads on AWS: Recovery in the Cloud - AWS Whitepaper 32. [WWW Document], URL https://docs.aws.amazon.com/whitepapers/latest/disaster-recovery-workloads-on-aws/disaster-recovery-workloads-on-aws.pdf (accessed 18.09.2021).

Fitzsimons, P., Besh, N., Stepanian, L., Jarrett, K., Ng, PT., Basbaum, A., Hauser, J. Cost Optimization Pillar - AWS Well-Architected Framework. URL https://d0.awsstatic.com/whitepapers/architecture/AWS-Cost-Optimization-Pillar.pdf (accessed 16.11.2021).

Fortinet, Inc., 2021. Amazon Web Services (AWS) Reference Architecture. . [WWW Document]. Fortinet, Inc. URL https://www.fortinet.com/content/dam/fortinet/assets/white-papers/wp-aws-reference-architecture.pdf (accessed 07.11.2021).

Gasevic, D., Kaviani, N., Milanović, M., 2009. Handbook on Ontologies. Ontologies and Software Engineering. pp. 593–615. https://doi.org/10.1007/978-3-540-92673-3_27

Google Cloud Infrastructure [WWW Document], Google Cloud Infrastructure. URL https://cloud.withgoogle.com/infrastructure (accessed 19.09.2021).

Haigh, T., Ceruzzi, P.E., 2021. A New History of Modern Computing. MIT Press.

Hassan, Q.F., 2011. Demystifying Cloud Computing. CrossTalk: The Journal of Defense Software Engineering. 24. https://www.researchgate.net/publication/215795101_Demystifying_Cloud_Computing

Kasanen, E., Lukka, K., Siitonen, A., 1993. The Constructive Approach in Management Accounting Research. Journal of Management Accounting Research

Khare, A., Huang, Y., Doan, H., Kanwal, M.S., 2012. A Fresh Graduate's Guide to Software Development Tools and Technologies. [WWW Document], n.d. URL https://www.comp.nus.edu.sg/~seer/book/2e/ (accessed 07.02.2022).

Krishnan, S.P.T., Gonzalez, J.L.U., 2015, Building Your Next Big Thing with Google Cloud Platform: A Guide for Developers and Enterprise Architects. Apress, Berkeley, CA, pp. 3–12. https://doi.org/10.1007/978-1-4842-1004-8_1

Kriushanth, M., Arockiam, L., Mirobi, G.J., 2013. Auto Scaling in Cloud Computing: An Overview 2, 6.

Liao, W.-H., Kuai, S.-C., Leau, Y.-R., 2015. Auto-scaling Strategy for Amazon Web Services in Cloud Computing, in: 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity). Presented at the 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity), pp. 1059–1064. https://doi.org/10.1109/SmartCity.2015.209

Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., Leaf, D., 2011. NIST Cloud Computing Reference Architecture 35. https://doi.org/10.6028/NIST.SP.500-292

Mahmudova, S., 2019. Analysis of Software Performance Enhancement and Development of Algorithm 4, 219–229. https://www.researchgate.net/publication/330752244_Analysis_of_Software_Performance_Enhancement_and_Development_of_Algorithm

Malkawi, M., 2013. The art of software systems development: Reliability, Availability, Maintainability, Performance (RAMP). Human-centric Computing and Information Sciences 3, 22. https://doi.org/10.1186/2192-1962-3-22

Mell, P., Grance, T., 2011. The NIST Definition of Cloud Computing. [WWW Document], URL https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication500-292.pdf (accessed 17.09.2021)

Ojala, A., 2013. Software-as-a-Service Revenue Models. IT Professional 15, 54–59. https://doi.org/10.1109/MITP.2012.73

Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S., 2008. A Design Science Research Methodology for Information Systems Research. Journal of Management Information Systems 45–77. DOI 10.2753/MIS0742-1222240302

Production-Grade Container Orchestration [WWW Document], Kubernetes. URL https://kubernetes.io/ (accessed 19.09.2021).

Qasim, Z., 2015. Metrics for Quality Assurance of Web Based Applications. [WWW Document], URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2692506

Qin, Z., Xing, J.-K., Zheng, X., 2009. Software Architecture. Springer Science & Business Media.

Ronzon, T., 2016. Software Retrofit in High-Availability Systems: When Uptime Matters. IEEE Software 33, 11–17. https://doi.org/10.1109/MS.2016.49

Roy, C., Barua, K., Agarwal, S., Pandey, M., Rautaray, S., 2019. Horizontal Scaling Enhancement for Optimized Big Data Processing: Proceedings of IEMIS 2018, Volume 1. pp. 639–649. https://doi.org/10.1007/978-981-13-1951-8_58

Salam, A., Gilani, Z., Haq, S.U., 2015. Deploying and Managing a Cloud Infrastructure: Real-World Skills for the CompTIA Cloud+ Certification and Beyond: Exam CV0-001. John Wiley & Sons.

Sala-Zárate, M., Colombo-Mendoza, L., 2012. CLOUD COMPUTING: A REVIEW OF PAAS, IAAS, SAAS SERVICES AND PROVIDERS. Lámpsakos 47. https://doi.org/10.21501/21454086.844

Savchenko, D., 2019. Testing microservice applications. Lappeenranta-Lahti University of Technology LUT. https://urn.fi/URN:ISBN:978-952-335-415-9

Sawehli, A., Xiang, B., 2018. Data Center Infrastructure (DCI) - Assignment. https://doi.org/10.13140/RG.2.2.32032.43522

Sharma, R., 2015. Architecture design for high availability - Learning OpenStack High Availability. Packt Publishing.

Sommerville, I., 2011. Software engineering, 9th ed. ed. Pearson, Boston.

Tomsen Bukovec, M.-L., 2018. AWS re:Invent 2018 - Keynote with Dr. Werner Vogels. URL https://www.youtube.com/watch?v=femopq3JWJg (accessed 18.10.2021).

Wedman, S., Tetmeyer, A., Saiedian, H., 2013. An Analytical Study of Web Application Session Management Mechanisms and HTTP Session Hijacking Attacks. Information Security Journal: A Global Perspective 22, 55–67. https://doi.org/10.1080/19393555.2013.783952

Weiss, B., Furr, M., 2019. Static stability using Availability Zones. URL https://d1.awsstatic.com/builderslibrary/pdfs/static-stability-using-availability-zones.pdf?did=ba_card-body&trk=ba_card-body (accessed 17.10.2021).

Wiger, N., Timalsina, R., 2019. Performance at Scale with Amazon ElastiCache. [WWW Document], URL https://d0.awsstatic.com/whitepapers/performance-at-scale-with-amazon-elasticache.pdf (accessed 07.10.2021).

Wolf, D., Henley, A.J., 2017. Java EE Web Application Primer. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-3195-1

Appendix 1. Amazon Web Services' (AWS) deployment configurations

Creating a Virtual Private Cloud

For demonstration purposes, the VPC has been created with the following values:

- Name tag: Demo VPC
- IPv4 CIDR: 10.0.0.0/16

It is recommended to enable DNS hostnames to EC2 instances in the VPC for easier identification.

Creating a subnet

The following values are used for demonstration:

- VPC ID: Demo VPC
- Subnet name: Public subnet 1
- Availability Zone: us-west-2a
- IPv4 CIDR block: 10.0.0.0/24

Creating a public route table

To set up a public route table, the following settings were used:

- Name: Public route table
- VPC: Demo VPC

Setting a route to direct internet-bound traffic to the internet gateway

A route to direct internet-bound traffic to the internet gateway needs to be configured as follows:

- Destination: 0.0.0.0/0
- Target: Previously created internet gateway

Creating security groups

For demonstration purposes, the following security groups were created:

Load balancer security group:

- Security group name: Load balancer SG
- Description: Allow HTTP and HTTPS access to load balancer
- VPC: Demo VPC
- Inbound rules:
  - Type: HTTP, HTTPS
  - Source: Anywhere-IPv4

Application security group:

- Security group name: App SG
- Description: Allow traffic from load balancer
- VPC: Demo VPC
- Inbound rules:
  - Type: HTTP
  - Source: Load balancer SG

Creating a load balancer

In this project, the load balancer is placed in the public subnet with the following settings:

- Load balancer type: Application Load Balancer
- Load balancer name: App LB
- Scheme: Internet-facing
- IP address type: IPv4
- VPC: Demo VPC
- Mappings: us-west-2a, us-west-2b
- Security groups: Load balancer SG
- Listeners and routing:
  - Protocol: HTTP, HTTPS
  - Port: 80, 443

Creating a launch configuration

The launch configuration is set up as follows:

- Name: Demo LC
- AMI: amzn2-ami-kernel-5.10-hvm-2.0.20211201.0-x86_64-gp2
- Instance type: t2.micro
- User data:

```
#!/bin/bash
#install httpd(Linux2 version)
yum update -y
yum install -y httpd.x86_64
systemctl start httpd.service
systemctl enable httpd.service
EC2_AVAIL_ZONE=$(curl -s http://169.254.169.254/latest/meta-
data/placement/availability-zone)
```

EC2_INSTANCE_ID=$(curl -s http://169.254.169.254/latest/meta-data/instance-id)

echo "Hello from instance $EC2_INSTANCE_ID in availability zone $EC2_AVAIL_ZONE" > /var/www/html/index.html

- Security groups: App SG
- Key pair name: Demo key

Creating an auto-scaling group

The auto-scaling group is created with the following setup:

- Name: Demo ASG
- Launch configuration: Demo LC
- VPC: Demo VPC
- Availability zones and subnets:
  - us-west-2a | Private subnet 1
  - us-west-2b | Private subnet 2
- Load balancer: App LB
- Health check type: ELB
- Group size:
  - Desired capacity: 2
  - Minimum capacity: 1
  - Maximum capacity 2