FH University of Applied Sciences TECHNIKUM WIEN

MASTER THESIS

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program Computer Science

Comparison between Microservice and Monolithic API in a Cloud Environment

By: Florian Feka

Student Number: 1910257104

Supervisor: Aichbauer Lukas, MSc.

Vienna, October 22, 2022

Declaration

“As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (see Urheberrechtsgesetz /Austrian copyright law as amended as well as the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I hereby declare that I completed the present work independently and that any ideas, whether written by others or by myself, have been fully sourced and referenced. I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool.“

Vienna, October 22, 2022 Signature

Kurzfassung

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Schlagworte: Schlagwort1, Schlagwort2, Schlagwort3, Schlagwort4

Abstract

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Keywords: Keyword1, Keyword2, Keyword3, Keyword4

Contents

1 Introduction

Companies nowadays have many options on how to provide their software product to their customers and with this many things to consider when choosing hosting platform. Corporations can choose to deploy on premise handling everything from the software down to maintaining servers or delegate some of that work to cloud providers. Applications can be deployed through Infrastracture as a Service (IaaS)[3] or Platform as a Service (PaaS)[4] and have a chance to take advantage of other services provided by Cloud Providers, like auto scaling, high availability, continuous delivery and more. Most companies that move an application to the Cloud predominantly move a monolithic application.

Companies move to the cloud in hope of utilizing features of IaaS/PaaS solutions to improve efficiency in their processes and to improve scaling to brace for peaks in requests. Most software application developed by companies are three tiered web applications, commonly using Java, .Net, PHP. Many of those applications face problems when migrating to the cloud since many architecture styles commonly used when developing web applications do not consider the ability to add/remove servers on demand and do not consider the option for multiple instances of the server to run.

Monoliths are applications build in, often one, big code base. There are some variations since some monoliths are split according to their overarching responsibility. For example a monolith that has one code base including the back-end and front-end(e.g. Spring in combination with Thymeleaf) could be split into Spring being the back-end and Angular being the front-end.

Microservices are separate, small, modular services that can be independently developed, updated, deployed and managed. Each service runs in its process and communicates over the network using a well-defined communication protocol. These services are self-describing and can be discovered and used by other processes without human intervention. Microservices are loosely coupled and can be easily scaled up and down based on the current business needs. They can also be used to develop applications much faster than a monolithic approach because each service can be developed independently of one another.[6]

1.1 Goals and Scope

The main goal of this thesis is to compare the monolithic and microservice architectural style in a cloud environment. Cloud provider offer many options on how to build an application, whether by build the infrastructure and application or just the application and let the cloud provider manage the infrastructure. This thesis revolves around the following question:

- Which of the two architectual styles profits more from the landscape of services in the cloud environment and which is more cost and resource effective?

This question can be splitted in the following sub-questions:

- Which of the two architectual styles is cheaper while performing the same?
- Consequently which performs better while allocated the same amount of resources?
- Does one of the two architectual styles scale more efficiently?

The thesis does not try to answer which types of applications would benefit from any architectual style but just tries to outline the differences between monoliths and microservices deployed in the cloud.

1.2 Approach

To answer the above stated questions, an imaginary software product is created written in a monolithic architecture and once as microservices. With this they will be deployed and benchmarked in Microsoft Azure. With the data collected through the benchmarks and Azures Cost Manager the results can be compiled and should provide answers for this thesis.

1.3 Structure of the Thesis

This thesis is split into three chapters. First chapter being "Basics" which defines all components used in the implementation or relevant to the subject. In this chapter architecture patterns, software tools and general cloud provider products are explained. This thesis does not hone in on to Microsoft Azure since what is being used for this thesis is also available at other cloud providers. The second chapter "Applied Methods" will go into detail how the imaginary product is structured, which tools were used and how. It will also explain how the application was benchmarked. The last chapter "Results" will inculde the findings of this thesis.

2 Basics

2.1 Monolith

Monolithic software is often written with components and functions being tightly coupled and designed to be self-contained meaning it builds to be software with no external dependencies after being built.
Enterprise applications are often built in three pieces:

- A client-side user interface consisting of HTML, CSS and Javascript that run in a Web Browser
- A server-side application which will handle HTTP Requests and business logic
- A database which consists of tables usually using a relational model

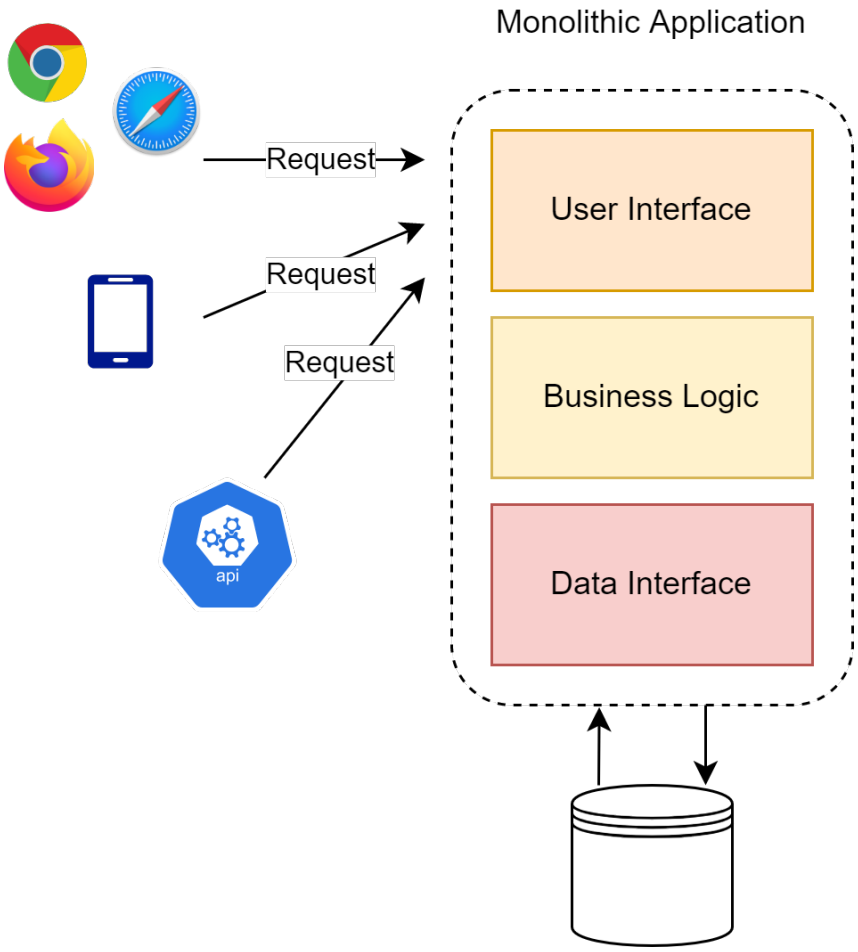

Figure 1: Monolithic Architecture

A monolithic application has all services and business logic in one code base which is being developed on. The development team need to ensure when modifying services that other parts of the application do not break. Monolithic architectures are common in many applications due to their simplicity and ability to meet system requirements quickly, while limiting the number of dependencies that must be satisfied at deployment time. For example, e-commerce applications tend to use monolithic architectures because they are relatively simple, and can be deployed quickly to achieve product/market fit. However, as the system scales, the system becomes increasingly complex to maintain and troubleshoot. Dependency management becomes increasingly difficult, and managing releases becomes challenging as changes have to be made across multiple layers in the system. A monolithic architecture also introduces a single point of failure that can adversely impact the entire system if it fails.[16]

2.2 Microservice

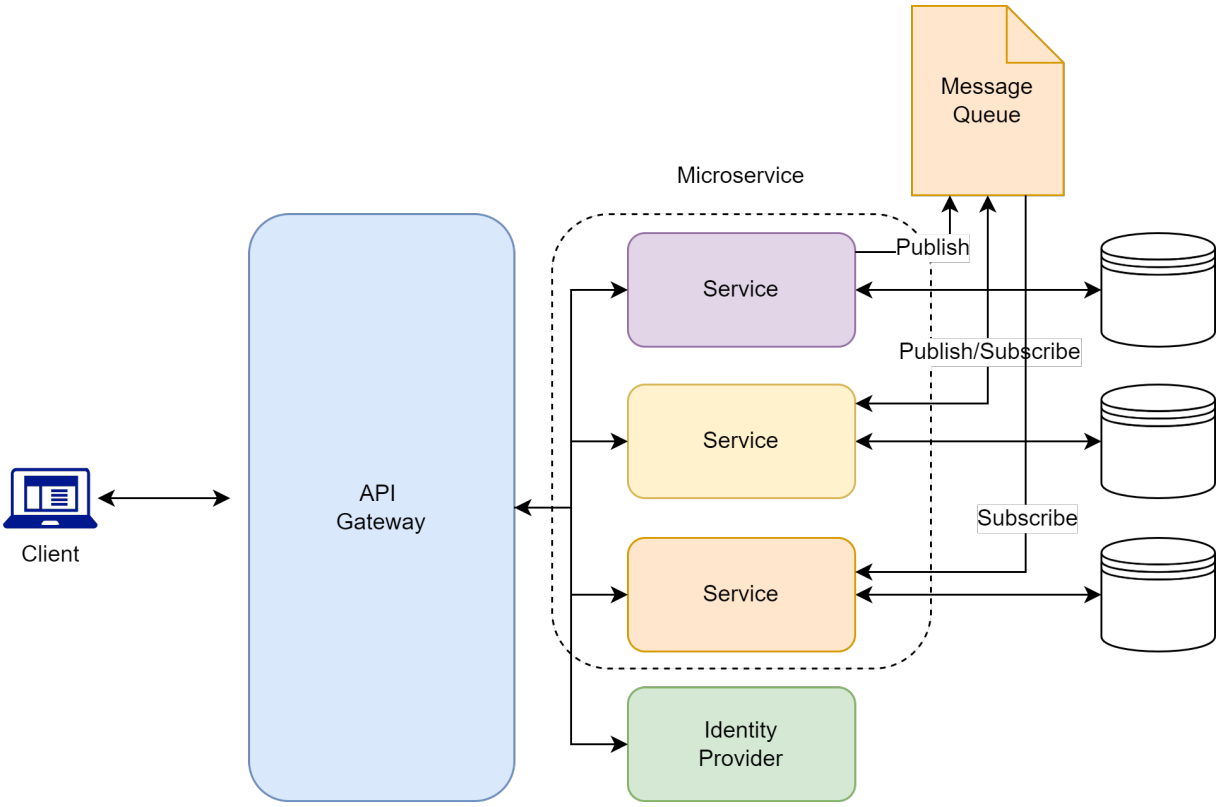

Figure 2: Microservice Architecture

Microservices are separate, small, modular services that can be independently developed, updated, deployed and managed. Each service runs in its process and communicates over the network using a well-defined communication protocol.[11] These services are self-describing and can be discovered and used by other processes without human intervention. They are

loosely coupled and can be easily scaled up and down based on the current business needs.

Microservices are a popular architectural style for building applications, where each service does one thing well. Each service typically only exposes a small set of its functionality via a RESTful API and communicates with other services using asynchronous messaging.[2] This style of architecture allows services to be scaled independently; if one service is overloaded, it can simply stop accepting new requests for a period of time without affecting the performance of other services.

Microservices are often built using API-first approaches, where the API is designed and developed first, followed by the other services that use it. This approach enables developers to prototype and test the microservices in isolation before integrating them into the main application. This makes development and testing faster and more cost-effective. It also means that once a service is successfully deployed, it can be used in other contexts where it may be used to achieve additional benefits.

Microservices are a recent trend where each service can be developed independently and scaled independently, making it much faster to develop and deploy new versions or features with minimal downtime for the application users. On the other hand, a monolithic API approach is easier to maintain since it consists of a single process across the server instead of multiple processes that have to communicate over the network with each other. [13]

API management platforms[6] are tools which allow microservices to be managed easily. These tools allow the developer to work with different services and easily integrate them in the application by providing a uniform interface for the developers to communicate with each of the services. They provide tools for deploying and monitoring the services so they can easily be scaled up or down depending on the needs of the business.

2.3 The Cloud

By implementing DevOps practices in their organizations, companies can align IT operations with business goals and increase operational efficiency and agility. This is achieved by adopting new technologies like cloud computing, containerization, microservices, and automation tools to speed up the development lifecycle and deliver high-quality products and services to customers faster. Companies can also reduce costs by optimizing compute and storage consumption across different workloads and infrastructures.[14]

2.3.1 Infrastracture as a Service (IaaS)

Cloud providers such as AWS, Microsoft Azure, Google Cloud Platform, IBM Cloud, or Oracle Cloud provide cloud infrastructure-as-a-service (IaaS) that allows businesses to set up virtual machines on a shared infrastructure to manage their applications and workloads. IaaS is a type of cloud service that provides virtualized computing resources (e.g. CPU, memory, storage and networking resources) on demand to users over the Internet. It is managed by the service provider using a cloud computing stack that consists of underlying hardware, virtualization software and the cloud management platform.[7]

IaaS solutions enable companies to build and deploy their own custom IT systems on the cloud without having to purchase and maintain any hardware or software. It enables them to focus on building their applications and business processes without worrying about the technical details related to the IT infrastructure. IaaS also offers a lot of flexibility in terms of scaling up or down the resources required for their workloads as and when needed. It is still required to manage the OS and the packages whole deployment process.

2.3.2 Platform as a Service (PaaS)

Platform as a Service (PaaS) is a cloud computing service that provides a platform on which developers can build and deploy their applications without having to worry about the underlying infrastructure or hardware resources.

The main benefits of using PaaS include the following:

- Quick Setup and Deployment: With PaaS, developers no longer need to concern themselves with setting up and maintaining an IT infrastructure for running their applications as all of this is done for them by the service provider. They can simply sign up for the service and start developing their application immediately and deploy it to the production environment by simply pushing a button without any further configuration or maintenance required. This can save a lot of time and effort on the part of the developer so that they can focus on developing the application itself rather than on troubleshooting technical issues.[10]
- Cost Effective: Eliminates the need to purchase and maintain expensive hardware and software resources for running applications and reduces the costs associated with managing such resources. Developers can use these services to develop their applications without having to invest in costly infrastructure components and equipment.[9]
- High Availability: Helps achieve high availability in a cost-effective manner by allowing users to scale their applications up or down depending on the requirements at any given time. With this service, they can easily scale down their applications when they are not

required in order to increase efficiency and reduce costs. Similarly, they can scale up their applications when needed to provide additional capacity to handle increased workloads without incurring any additional costs. In addition, the service provides them with a variety of options to support different business requirements such as number of users, data storage requirements and bandwidth availability. This allows them to optimize the performance and functionality of their applications for different types of users.

- Scalability: Allows developers to easily scale their applications to accommodate growing workload requirements without having to make any additional investments.[10] They can do this by simply adding additional resources to their applications and scaling them up accordingly to meet the increasing demands. They can also easily scale their apps down when they are no longer required in order to reduce operational costs and minimize their storage requirements.
- Flexibility: Provides users with a high degree of flexibility in terms of choosing their services and platform components to meet their specific needs. This gives them the ability to choose each component individually based on their specific requirements and only pay for the ones that they need. It also allows them to implement new features and functionality quickly and easily without requiring them to make any changes to the existing applications.

2.4 Docker

Docker is an open-source platform for automating application containers on Linux based operating system. A Docker Container is lightweight, portable, and isolated application packaging. It allows developers to package an application and all its dependencies in a single file, known as a "container", that can run on any Linux server. Different containers can be combined into a single cluster, and each container runs as an independent process with its namespace. Docker containers are lightweight and can easily be moved between development and production environments. They can also easily be linked together to create powerful applications.[15]

The key benefits of using Docker Containers include the following:

- Lightweight: Docker containers are generally smaller compared to a virtual machine. This means multiple applications can run on a single server without compromising performance.[5]
- Portable: Docker containers are highly portable, which facilitate moving applications from development to production or even between different servers.[15] This is especially useful for DevOps teams as they can use the same containers in different environments, such as development and staging or production and testing.

- Isolated: Each Docker container is completely isolated from other containers on the system, so there is less risk of a vulnerability affecting multiple containers at the same time.[8] This also help to have multiple container running different versions of runtimes or compilers which would be much harder to maintain in a virtual machine.
- Easy to Manage: Docker containers are designed to minimize management overhead so that developers can focus on building great apps instead of managing infrastructure. This is achieved by giving the developer fine-grained control over container resources and providing simple tools for managing them.
- Secure and Reliable: Docker containers are more secure and reliable than traditional applications because they are self-contained and easy to manage. With Docker containers, a developer can use a different image for each environment and easily migrate their application between environments without running into any downtime or other issues.
- Efficient: Because Docker containers are easy to create and manage, they provide a cost-effective alternative to traditional applications that typically require complex and expensive infrastructure and management solutions. As a result, organizations can save a significant amount of time and money by moving to Docker containers for their development and deployment needs.
- High Performance: Containers provide a significant boost to the performance of any application by isolating it from the host machine. This means that each container can run independently of the others so that there are no resource constraints that can negatively affect its performance.[15]
- Easily Scaled: Because containers run on the Linux kernel, they are easily scalable and highly customizable, which means that they can be configured to meet a wide range of performance and resource requirements.[15] In addition, they are compatible with most standard networking and storage technologies and can therefore be deployed in many environment.

Due to these advantages, Docker containers have become the de facto standard for deploying and running applications in the public cloud and have been widely adopted by both individual developers and large enterprises for all of their development and deployment needs.

2.5 Kubernetes

Kubernetes is an open source software for automating deployment, scaling and management of containerized applications. It provides a scalable way to run containerized applications on any public cloud or on-premise infrastructure. Kubernetes is workload agnostic, meaning that it can support multiple types of services without requiring the developers to re-architect the application for Kubernetes.[12]

Kubernetes features include resource provisioning, scalability, load balancing, service discovery, multi-cluster management, auto-scaling, etc. It also provides a REST API for managing resources and a web-based dashboard for end users to monitor the status of the cluster and the resources in it. These features enable Kubernetes to control and monitor many containers simultaneously in a dynamic environment. This enables it to be used as an Infrastructure as a Service (IaaS) platform for controlling and running applications in the cloud. It can also be used for development and testing environments as it can dynamically scale resources to meet user demand without deploying new servers each time additional resources are required.[1]

2.6 .NET

//TODO

2.7 API

//TODO Quick intro: https://dev.to/andreidascalu/soap-vs-rest-vs-grpc-vs-graphql-1ib6

2.7.1 Restful API

2.7.2 SOAP API

2.7.3 gRPC

2.7.4 graphQL

3 Applied Methods

The project created for this thesis called "Examich" is for students to create, manage and share practice exams. This service also offers the option to generate a PDF Version of the practice exams. Since this thesis focuses on the API the frontend for this project will be skipped.

3.1 Monolith

3.1.1 Architecture & Implementation

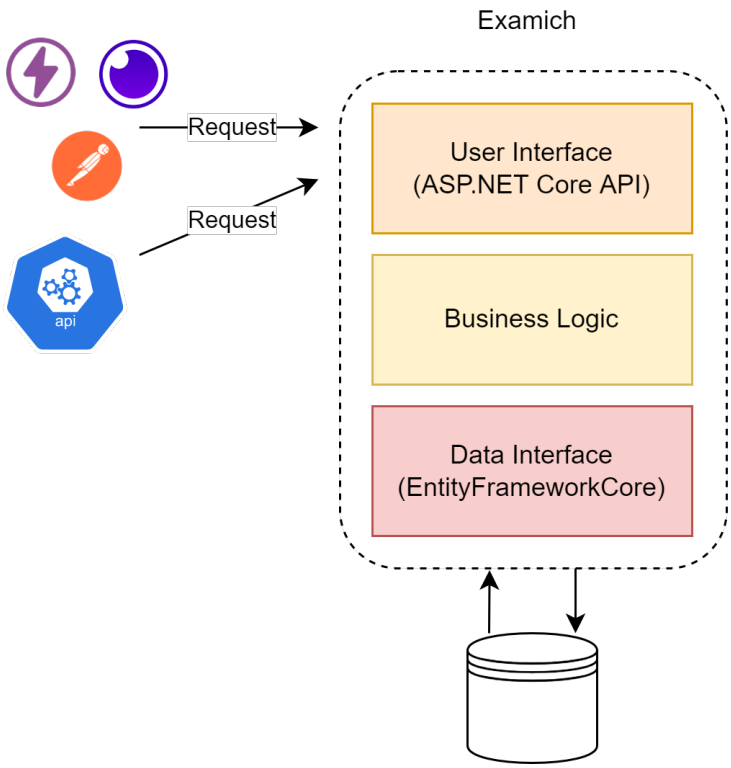

Figure 3: Examich Monolith Architecture

The monolith consists of tree main parts. The API that exposes the functionality to its users. The business logic which does for now only handle the PDF Generation and the data layer which handles the communication with the data base.
The endpoints it exposes:

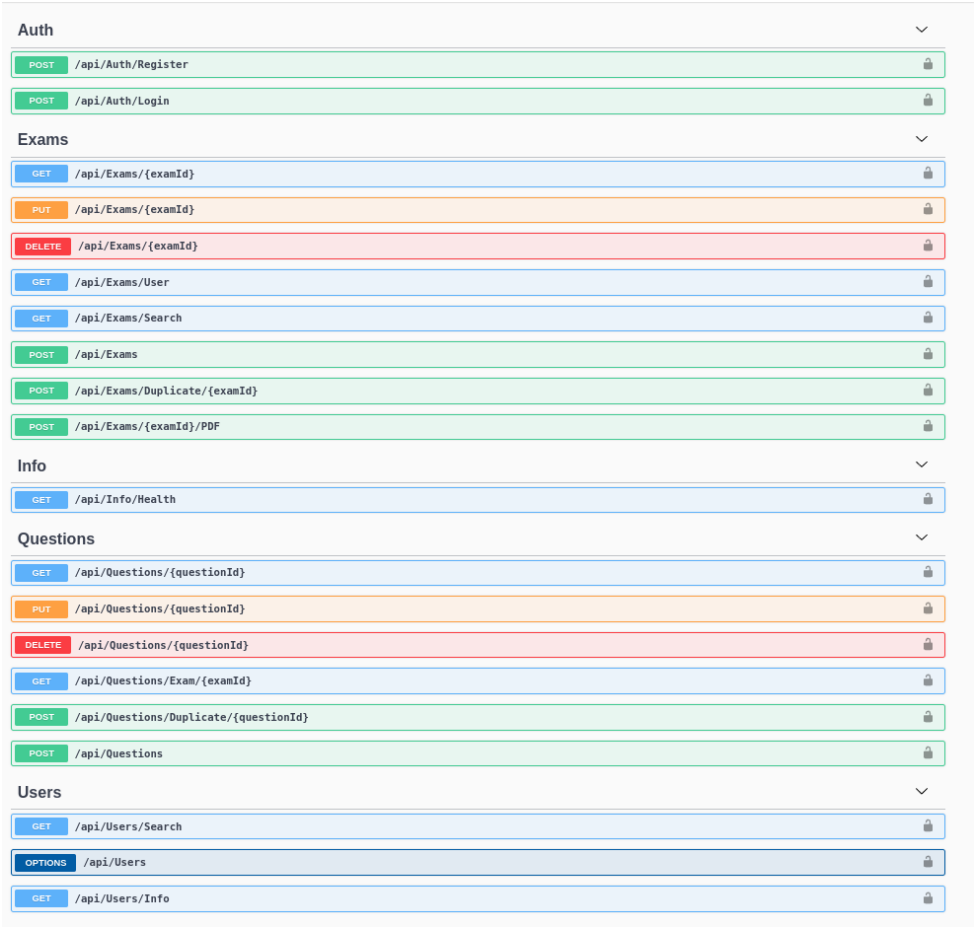

Figure 4: Examich Monolith Swagger

The most resource intense endpoint would be "/api/Exams/examId/PDF", since with this the PDF generation is triggered. It will render the PDF and send it as response back to the user.

3.1.2 Data Model

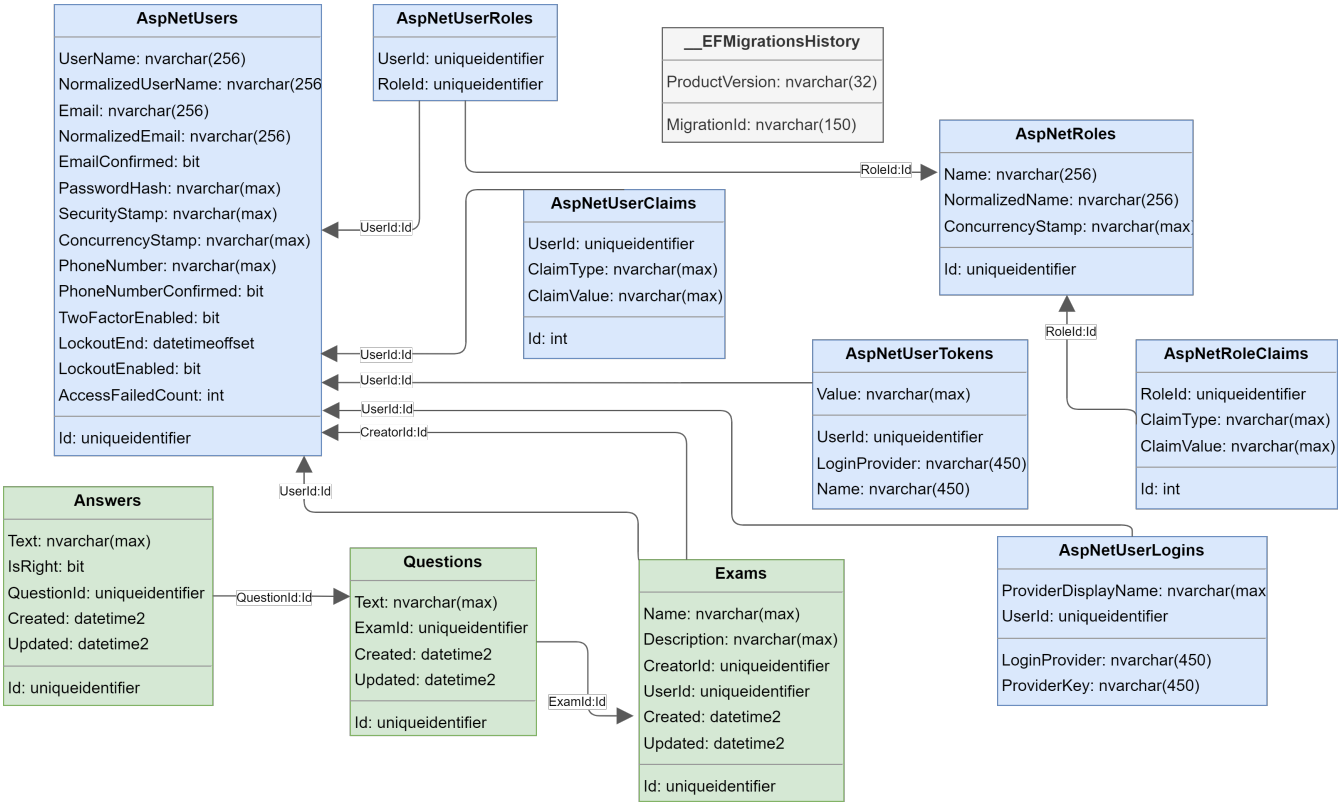

Figure 5: Examich Monolith Data Model

This is the data model for the monolithic application. The blue tables are tables generated by the Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityDbContext, the grey one is generated be EntitiyFrameworkCore to keep track of the migrations and the green ones were created for the project. The important tables are the green ones. Every User had zero or more Exams, every Exam had zero or more Questions and every Question had zero or more Answers. A pretty simple data structure.

3.2 Microservice

3.2.1 Architecture & Implementation

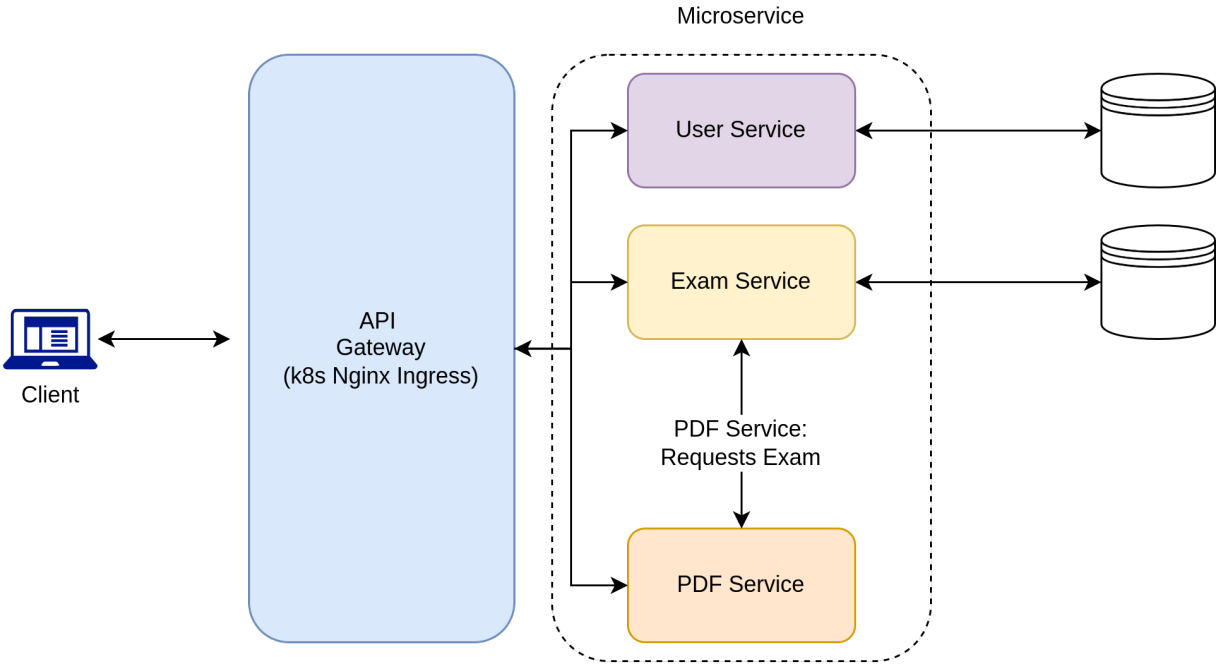

Figure 6: Examich Microservice Architecture

3.2.2 Data Model

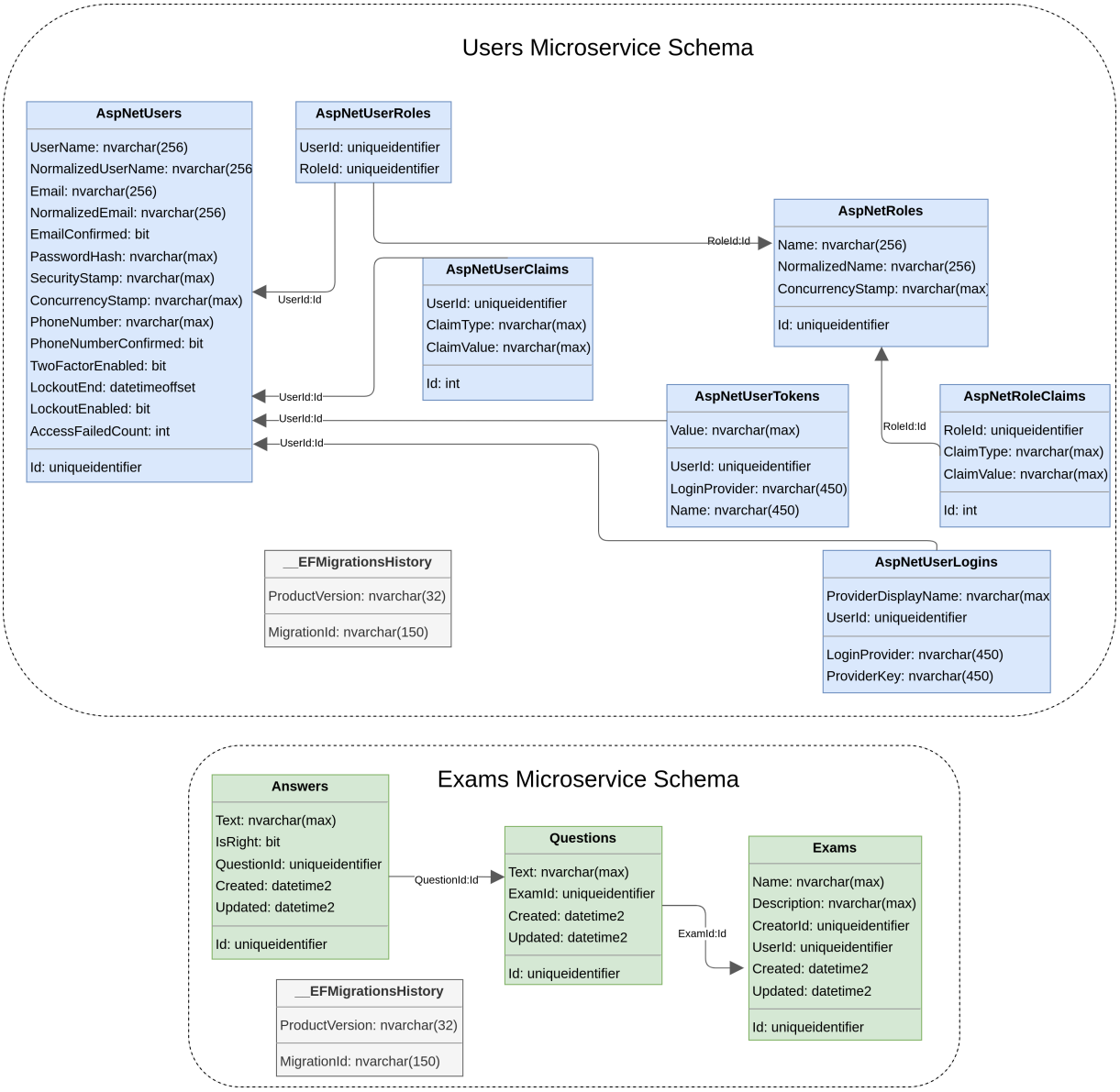

Figure 7: Examich Microservice Data Model

The

3.3 Performance Benchmark

This chapter will cover how the performance for the monolith and microservice were tested.

3.3.1 k6 - Load Testing Tools

4 Results

4.1 Erste Überschrift Tiefe 2 (section)

4.1.1 Erste Überschrift Tiefe 3 (subsection)

Erste Überschrift Tiefe 4 (subsubsection)

5 Conclusion

5.1 Erste Überschrift Tiefe 2 (section)

5.1.1 Erste Überschrift Tiefe 3 (subsection)

Erste Überschrift Tiefe 4 (subsubsection)

Zweite Überschrift Tiefe 4 (subsubsection)

Querverweise werden in LaTeX automatisch erzeugt und verwaltet, damit sie leicht aktualisiert werden können. Hier wird zum Beispiel auf Abbildung 8 verwiesen.

Figure 8: Beispiel für die Beschriftung eines Buchrückens.

```
1 #include <iostream>
2
3 void SayHello(void)
4 {
5     // Kommentar
6     cout << "Hello World!" << endl;
7 }
8
```

```
 9 int main(int argc, char **argv)
10 {
11     SayHello();
12     return 0;
13 }
```

Quellcode 1: 1. Beispiel

Bibliography

[1] *Production-grade container orchestration*.

[2] *Service meshes in a microservices architecture*.

[3] *What is IaaS?*.

[4] *What is PaaS?*.

[5] *Why use containers vs. VMS?: Vmware glossary*.

[6] ADERALDO, C. M., N. C. MENDONCA, C. PAHL and P. JAMSHIDI: *Benchmark requirements for Microservices Architecture Research*. 2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE), 2017.

[7] BUYYA, R., S. PANDEY and C. VECCHIOLA: *Cloudbus Toolkit for Market-Oriented Cloud Computing*. In JAATUN, M. G., G. ZHAO and C. RONG (eds.): *Cloud Computing*, pp. 24–44, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[8] COMBE, T., A. MARTIN and R. DI PIETRO: *To Docker or Not to Docker: A Security Perspective*. IEEE Cloud Computing, 3(5):54–62, 2016.

[9] GAI, K. and A. L. STEENKAMP: *A Feasibility Study of Platform-as-a-Service Using Cloud Computing for a Global Service Organization*. Journal of Information Systems Applied Research, 7:2842, Aug 2014.

[10] LAWTON, G.: *Developing Software Online With Platform-as-a-Service Technology*. Computer, 41(6):13–15, 2008.

[11] LEWIS, J. and M. FOWLER: *Microservices a definition of this new architectural term*, March 2014.

[12] MARKO, L.: *Kubernetes in action*, p. 1624. Manning, 2018.

[13] PATEL, Y.: *Benefits of microservices: What they are, Business Value, examples*, Nov 2021.

[14] QIAN, L., Z. LUO, Y. DU and L. GUO: *Cloud Computing: An Overview*. In JAATUN, M. G., G. ZHAO and C. RONG (eds.): *Cloud Computing*, pp. 626–631, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[15] RAD, B. B., H. J. BHATTI and M. AHMADI: *An introduction to docker and analysis of its performance*. International Journal of Computer Science and Network Security (IJCSNS), 17(3):228, 2017.

[16] VILLAMIZAR, M., O. GARCES, H. CASTRO, M. VERANO, L. SALAMANCA, R. CASALLAS and S. GIL: *Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud*. 2015 10th Computing Colombian Conference (10CCC), 2015.

List of Figures

List of Tables

Quellcodeverzeichnis

Abkürzungsverzeichnis

ABC Alphabet

WWW world wide web

ROFL Rolling on floor laughing

A Anhang A

B Anhang B