FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Designing a business platform using microservices.

Rajendra Kharbuja

FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Designing a business platform using microservices.

Entwerfen einer Business-Plattform mit microservices.

Author:	Rajendra Kharbuja
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Manoj Mahabaleshwar
Submission Date:	

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, Rajendra Kharbuja

Acknowledgments

Abstract

Contents

1 Challanges of Microservices Architecture

1.1 Introduction

The section **??** lists some drawbacks of monolithic architecture and these have become the motivation for adopting microservices architecture. Microservices offer opportunities in various aspects however it can also be tricky to utilize them properly. It do come with few challanges. The response from interview **??** also highlights some prominent challanges. In this chapter, the challanges of microservices alongside its advantages will be discussed. [Fow15]

Advantages

Strong Modular Boundaries
It is not totally true that monolith have weaker modular structure than microservices but it is also not false to say that as the system gets bigger, it is very easy for monolith to turn in to a big ball of mud. However, it is very difficult to do the same with microservices. Each microservice is a cohesive unit with full control upon its business entities. The only way to access its data is through its API.

Independent Deployment
Due the nature of microservices being autonomous components, each microservice can be deployed independently. Deployment of microservices is thus easy compared to monolith application where a small change needs the whole system to be deployed.[New15]

Challanges

Distributed System
The infrastructure of microservices is distributed, which brings many complications alongside, as listed by 8 fallacies.[Fac14] The calls are remote which are imminent to accomplish business goals. The remote calls are slower than local and affect performance in a great deal. Additionally, network is not reliable which makes it challanging to accept and handle the failures.

Integration
It is challanging to prevent breaking other microservices when deploying a service. Similarly, as each microservice has its own data, the collaboration among microservices and sharing of data can be complex.

Agile
Each microservice is focused to single responsibility, changes are easy to implement. At the same time, as the microservices are autonomous, they can be deployed independently, making the release cycle time short.

Operational Complexity
As the number of microservices increases, it becomes difficult to deploy in an acceptable speed and becomes more complicated as the frequency of changes increase. Similary, as the granularity of microservices decreases, the number of microservices increases which shifts the complexity towards the interconnections. Ultimately, it becomes complex to monitor and debug microservices.

1.2 Integration

The collaboration among various microservices whilst maintaining deployment autonomy is challenging. In this section, various challanges associated with integration and their potential remedies are discussed.

1.2.1 Shared Database

An easiest way to collaborate, q is to allow services to access and update a common data source. However, using this kind of integration creates various problems.[New15]

1. The shared database acts as a point of coupling among the collaborating microservices. If a microservice make any changes to its data schema, there is high probability that other services need to be changed as well. Loose Coupling is compromised.
2. The business logic related to the shared data may be spread across multiple services. Changing the business logic is difficult. Cohesion is compromised.
3. Multiple services are tied to a single database technology. Migrating to a different technology at any point is hard.

Alternatives There are three distinct alternatives.[Ric15]

1. **private tables per service** - multiple services share same database underneath but each service owns a set of tables.
2. **schema per service** - multiple services share same database however each service owns its own database schema.

3. **database server per service** - each service has a dedicated database server underneath.

Consequences The logical separation of data among services increases autonomy but also present some downsides.[Ric16] [Ric15]

1. The implementation of business transaction which spans multiple services is difficult and not recommended because of theorem 2.1. The solution is to apply eventual consistency 2.2 focussing more on availability.
2. The implementation of queries to join data from multiple databases can be complicated. There are two alternatives to achieve this.
   a) A separate mashup service can be used to handle the logic to join data from multiple services by accessing respective APIs.
   b) CQRS pattern 2.3 can be used by maintaining separate views as well as logic for updating and querying data.
3. The need for sharing data among various autonomous services cannot be avoided completely. It can be achieved by one of the following approaches.
   a) The data can be directly accessed by using the resource owner′s API.
   b) The required data can be duplicated into another service and made consistent with the owner′s data using event driven approach.

2 Appendices

2.1 CAP Theorem

2.2 Eventual Consistency

2.3 Command Query Responsibility Segregation(CQRS)

Acronyms

API Application Programming Interface.

CQRS Command Query Responsibility Segregation.

CRUD create, read, update, delete.

DEP Dependency.

HCP Hana Cloud Platform.

IDE Integrated Development Environment.

IFBS International Financial and Brokerage Services.

ISCI Inter Service Coupling Index.

ODC Operation Data Granularity.

ODG Operation Data Granularity.

OFG Operation Functionality Granularity.

RCS Relative Coupling of Services.

RIS Relative Importance of Services.

SCG Service Capability Granularity.

SDG Service Data Granularity.

SDLC Software Development Life Cycle.

SFCI Service Functional Cohesion Index.

SIDC Service Interface Data Cohesion.

SIUC	Service Interface Usage Cohesion.
SIUC	Service Sequential Usage Cohesion.
SLC	Self Containment.
SMCI	Service Message Coupling Index.
SOAF	Service Oriented Architecture Framework.
SOCI	Service Operational Coupling Index.
SOG	Service Operations Granularity.
SRI	Service Reuse Index.
SWIFT	Society for Worldwide Interbank Financial Telecommunication.
UML	Unified Modeling Language.
YaaS	Hybris as a Service.

List of Figures

List of Tables

Bibliography

[Fac14] P. Factor. *The Eight Fallacies of Distributed Computing*. Dec. 2014.

[Fow15] M. Fowler. *Microservice Trade-Offs*. July 2015.

[New15] S. Newman. *Building Microservices*. O'Reilly Media, 2015.

[Ric15] C. Richardson. *Does each microservice really need its own database?* Sept. 2015.

[Ric16] C. Richardson. *Pattern: Database per service*. 2016.