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

The microservices architecture provides various advantages such as agility, independent scalability etc., as compared to the monolithic architecture and thus has gained a lot of attention. However, implementing microservices architecture is still a challenge as many concepts within the microservices architecture including granularity and modeling process are not yet clearly defined and documented. This research attempts to provide a clear understanding of these concepts and finally create a comprehensive guidelines for implementing microservices.
Various keywords from definitions provided by different authors are taken and cateogorized into various conceptual areas. These concepts along with the keywords are researched thoroughly to understand the microservices architecture. Additionally, the three important drivers: quality attributes, constraints and principles, are also focussed for creating the guidelines.
The findings of this research indicate that eventhough microservices emphasize on the concept of creating small services, the notion of appropriate granularity is more important and depends upon four basic concepts which are : single responsibility, autonomy, infrastructure capability and business value. Additionally, the quality attributes such as coupling, cohesion etc should also be considered for identification of microservices. Furthermore, in order to identify microservices, either the domain driven design approach or the use case refactoring approach can be used. Both these approaches can be effective in identifying microservices, but the concept of bounded context in domain driven design approach identifies autonomous services with single responsibility. Apart from literature, a detail study of the architectural approach used in industry named SAP Hybris is conducted. The interviews conducted with their key personnels has given important insight into the process of modeling as well as operating microservices. Moreover, the challenges for implementing microservices as well as the approach to tackle them are done based on the literature review and the interviews done at SAP Hybris.
Finally, the findings are used to create a detail guidelines for implementing microservices. The guidelines captures how to approach modeling microservices architecture and how to tackle its operational complexities.

Contents

1 Introduction

The software architecture is the set of principles assisting software architect and developers for system or application design [DMT09]. It defines a process to decompose a system into modules, components and specifies their interactions [Bro15]. In this chapter, two different architectural approaches namely monolithic architectural approach and microservices achitectural approach are discussed. Firstly, a conceptual understanding of the monolithic architecture approach is presented, which is followed by its various advantages and disadvantages. Secondly, an overview of the microservices architecture approach is explained. In Section 1.3, the motivation for the current research is discussed which is then followed by a list of research questions. Finally, in Section 1.4 and Section 1.5, the approach that is used to conduct the current research is presented. The purpose of this chapter is to provide a basic background context for the following chapters.

1.1 Monolithic Architectural Approach

A mononlithic architectural approach is one in which even a modular application is deployed as a single artifact. Figure 1.1 shows the architecture of an Online-Store application that has a clear separation of components such as Catalog, Order etc., as well as respective models such as Product, Order etc. Despite the modular decomposition of the application, all the components are deployed as a single application artifact on the server.[Ric14a][Ric14c]

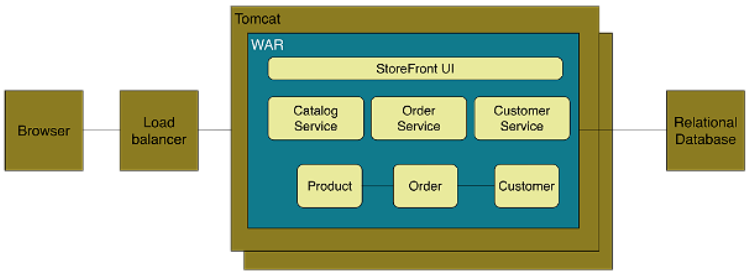

Figure 1.1: Monolith Example from [Ric14a]

1.1.1 Types of the Monolithic Architectural Approach

According to [Ann14], a monolith can be of several types depending upon the viewpoint, as shown below:

1. **Module Monolith**: If all the code to realize an application share the same codebase and need to be compiled together to create a single artifact for the whole application then the architecture is called Module Monolithic Architecture. An example of this architecture is shown in Figure 1.2. The application on the left side of the figure, has all the code in the same codebase in the form of packages and classes without clear definition of modules and gets compiled to a single artifact. However, the application on the right side is developed as a number of modular codebase, each has separate codebase and can be compiled to different artifact. In the application shown in the left side of the figure, the various parts of the codebase can reference each other directly.

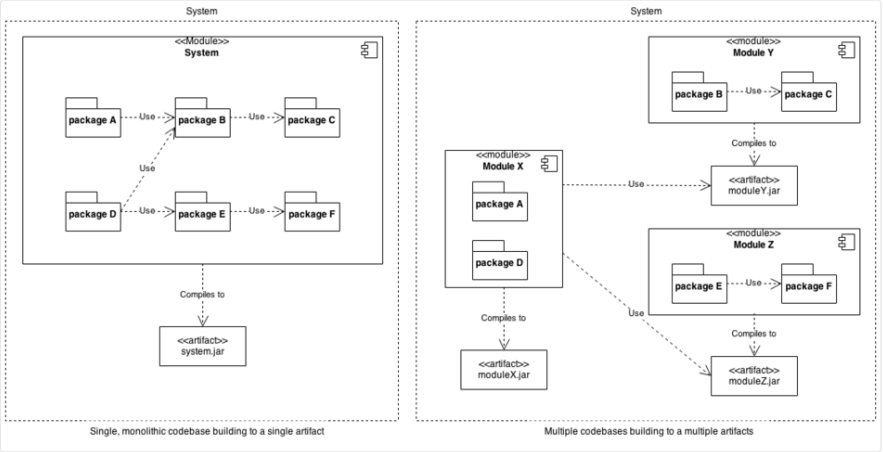

Figure 1.2: Module Monolith-Example from [Ann14]

2. **Allocation Monolith**: An Allocation Monolith is created when code is deployed to all the servers as a single version. This means that all the components running on the servers have the same version at any time. The Figure 1.3 gives an example of the allocation monolith. The system on the left side of Figure 1.3 has the same version of the artifact for all the components on all the servers. It does not make any difference whether or not the system has a single codebase and artifact. However, the system on the right as shown in the figure is realized with multiple versions of the artifacts in different servers at any time.

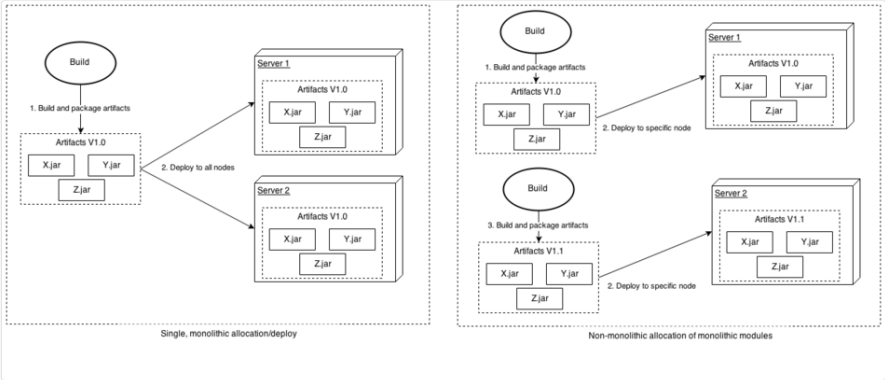

Figure 1.3: Allocation Monolith-Example from [Ann14]

3. **Runtime Monolith**: In the runtime monolith, the whole application is run under a single process. The application on the left side of Figure 1.4 shows an example of runtime monolith where a single server process is responsible for the whole application. Whereas the application on the right has allocated multiple server processes to run distinct set of component artifacts of the application.

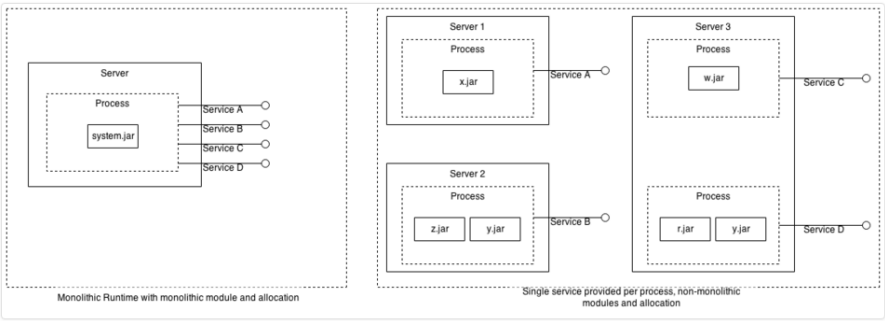

Figure 1.4: Runtime Monolith-Example from [Ann14]

1.1.2 Advantages of the Monolithic Architectural Approach

The monolithic architectural approach is appropriate for small applications and has the following benefits [Ric14c][FL14][Gup15][Abr14] :

- It is easy to develop a monolith application since various development tools including IDEs are created around the single application concept. Furthermore,

it is also easy to test the application by creating appropriate environment on the developer's machine.

- The deployment can be simply achieved by moving the single artifact for the application to an appropriate directory in the server.
- The scaling can be clearly and easily done by replicating the application horizontally across multiple servers behind a load balancer as shown in Figure 1.1
- The different teams are working on the same codebase so sharing the functionality can be easier.

1.1.3 Disadvantages of the Monolithic Architectural Approach

As the requirements of the system grows over time, the corresponding system's codebase becomes large and the size of team increases. Under such circumstances, the monolithic architecture faces many challenges as explained below [NS14] [New15][Abr14] [Ric14a] [Ric14c] [Gup15] :

- **Limited Agility**: As the whole application has a single codebase, even changing a small feature to release it in the production takes time. Firstly, a small change can trigger changes to other dependent code. In huge monolithic applications, it is very difficult to manage modularity especially when all the team members are working on the same codebase. Secondly, to deploy a small change in the production, the whole application has to be deployed. Thus continuous delivery gets slower. This is more problematic when multiple changes have to be released on a daily basis. The slow pace and low frequency of release will highly affect agility.

- **Decrease in Productivity**: It is difficult to understand the application especially for a new developer because of the size of codebase. Although it also depends upon the structure of the codebase, it will still be difficult to grasp the significance of the code when there is no hard modular boundary. Additionally, a developer can be intimidated due to need to see the whole application at once from outwards to inwards direction. Secondly, the development environment can be slow to load the whole application and at the same time the deployment will also be slow. Overall, it will slow down the speed of understandability, execution and testing.

- **Difficult Team Structure**: The division of a team as well as assigning tasks to the team members can be challenging. Most common ways to partition teams in monolith are by technology and by geography. However, partitioning either by technology or by geography may not be appropriate in all cases. In any case, the communication among the teams can be difficult and slow. Additionally, it is not easy to assign complete vertical ownership to a team for a particular feature from development to release. If something goes wrong in the deployment, there is always a question who should find the problem, is it either the operations team or the last person to commit the code. The approprate team structure and the ownership of code are very important for agility.

- **Longterm Commitment to Technology stack**: The technology to use is chosen before the development phase by analysing the requirements and the maturity of current technology at that time. All the teams in the project need to follow the same techonology stack throughout the lifecycle of an application. However, if the requirement changes then there can be situations when specific features can be best solved by different set of technology. Additionally, not all the features in the application are the same and hence cannot be solved using same technology. Nevertheless, the technology advances rapidly. So, the solution thought right at the time of planning can be outdated and there can be a better solution available at present. In monolithic applications, it is very difficult to migrate to a newer technology stack and it can rather be a painfull process.

- **Limited Scalability**: The scaling of monolith application can be performed in either of two ways. The first way is to replicate the application along many servers and dividing the incoming request using a load balancer in front of the servers. Another approach is using identical copies of the application in multiple servers as in previous case but partitioning the database access instead of user request. Both of these scaling approaches improves the capacity and availability of the application. However, the individual requirement regarding scaling for each component can be different but cannot be fulfilled with this approach. Also, the complexity of the monolith application remains the same because we are replicating the whole application. Additionally, if there is a problem in a component the same problem can affect all the servers running the copies of the application, this does not improve resilency.[Mac14][NS14]

1.2 Microservices Architectural Approach

With monolith, it is easy to start development. But as the system gets bigger and complicated over time, it becomes very difficult to be agile and productive. The disadvantages listed in Section 1.1.3 outweighs its advantages as the system gets old. The various qualities such as scalability and agility need to be maintained for the whole lifetime of the application. It becomes complicated due to the fact that the system needs to be updated continuously as the as the requirements keep changing and evolving over time. In order to tackle these disadvantages, microservices architecture style is followed.
The microservices architecture uses the approach of decomposing an application into smaller autonomous components. The following Section 1.2.1 provides details of the various decomposition techniques.

1.2.1 Decomposition of an Application

There are various ways to decompose an application. This section discuss two different ways of breaking down an application.

1.2.1.1 Scale Cube

The Section 1.1.3 captured various disadvantages related to the monolithic architectural approach. In [FA15], authors propose an approach called scale cube to address the challenges related to agility, scalability and productivity. The scale cube provides three dimensions of scalability as shown in Figure 1.5 which can be applied alone or simultaneously depending upon the situation and desired goals of the system.

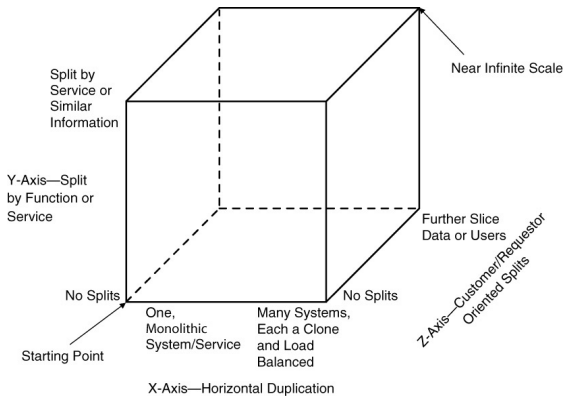

Figure 1.5: Scale Cube from [FA15]

The scaling along each dimensions are described below. [FA15][Mac14][Ric14a]

1. X-axis Scaling: It is achieved by cloning the application and data along multiple servers. A pool of requests are sent to a load balancer and the requests are deligated to the servers. Each of the server has the latest version of the application and access to all the data required. In this respect it does not make any difference which server fulfills the request. Rather, it is about how many requests are fulfilled at any time. It is easy to scale along X-axis as the number of requests increases. The solution is as simple as to add additional clones. However, with this type of scaling, it does not scale with the increase in data. Moreover, it also does not scale when there are large variation in the frequency of any type of requests because all the requests are handled in an unbiased way and allocated to servers in the same way.

2. Z-axis Scaling: The scaling is obtained by spliting the request based on certain criteria or information regarding the requestor or customer affected by the request. It is different than X-axis scaling in the way that the servers are responsible for different kinds of requests. Normally, the servers have the same copy of the application but some can have additional functionalies depending upon the requests expected. The Z-axis scaling helps in fault isolation and transaction scalability. Using this scaling, certain group of customers can be provided added functionality or a new functionality can be tested to a small group and thus minimizes the risk.

3. Y-axis Scaling: The scaling along this dimension means the spliting of the application's responsibility. The separation can be done either by data, by the actions performed on the data or by the combination of both. The respective ways can be referred to as resoure-oriented or service-oriented splits. While the x-axis or z-axis split are rather duplication of work along servers, y-axis is more about specialization of work along servers. The major advantage of this scaling is that each request is scaled and handled differently according to its necessity. As the logic along with the data to be worked on are separated, developers can focus and work on small section at a time. This will increase productivity as well as agility. Additionally, a fault in a component is isolated and can be handled gracefully without affecting rest of the application. However, scaling along Y-axis can be costly compared to scaling along other dimensions.

1.2.1.2 Shared Libraries

Libraries are a standard way of sharing functionalities among various services and teams. This capability is provided as a feature of the programming languages. The shared libraries however do not provide technology heterogeneity. Furthermore, independent scaling, independent deployment and independent maintainance cannot be achieved unless the libraries are dynamically linked. In such cases, any small change in the library leads to redeployment of the whole system. Moreover, sharing the code is a form of coupling which should be avoided.

Decomposing an application in terms of composition of individual features gives various advantages such that each feature can be a) scaled independently b) deployed independently c) agile d) assigned easily to teams. Thus, microservices uses the decomposition technique presented by the scale cube. The detail advantages of microservices architectural approach are discussed further in Section **??**.

1.2.2 Definitions

There are several definitions given by several pioneers and early adapters of the microservices architectural approach.

Definition 1: [Ric14b]

"It is the way to functionally decompose an application into a set of collaborating services, each with a set of narrow, related functions, developed and deployed independently, with its own database."

Definition 2: [Woo14]

"It is a style of software architecture that involves delivering systems as a set of very small, granular, independent collaborating services."

Definition 3: [Coc15]

"Microservice is a loosely coupled Service-Oriented Architecture with bounded contexts."

Definition 4: [FL14][RTS15]

"Microservices are Service-Oriented Architecture done right."

Definition 5: [FL14]

"Microservices architecture style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies."
Similar to other architectural styles, microservices presents its approach by increasing cohesion and decreasing coupling. Besides that, it breaks down the system along business domains (following single responsibility principle) into granular and autonomous services running on separate processes. Additionally, the architecture focuses on the collaboration of these services using light weight mechanisms.

1.3 Motivation

The various definitions presented in the previous section highlights different key terms such as:

1. collaborating services
2. developed and deployed independently
3. build around business capabilities
4. small, granular services

These concepts are very important to understand and realize the microservices architecture correctly and effectively. The first two terms relate to the runtime operational qualities of microservices whereas the later two address modeling qualities. With that consideration, it indicates that the definitions provided highlights on both modeling as well as operational aspects of any software applications.
However, if an attempt is made to have a clear indepth understanding of each of the key terms then various questions can be raised without suitable answers. The different

questions (shown in column 'Questions') related to modeling and operations (shown in column 'Type') are listed in the Table 1.1.

#	Questions	Type
1	How small should be the size of microservices?	Modeling
2	How do services interact and coordinate with eachother?	Operation
3	How to deploy and maintain services independently when there are dependencies among services?	Operation
4	How to derive and map microservices from business capabilities?	Modeling
5	What are the challenges need to be tackled when implementing microservices and how to tackle the challenges?	Operation

Table 1.1: Various Questions related to Microservices

Without clear answers to these questions, it is difficult to say that the definitions presented in Section 1.2.2 are completely enough to follow the microservices architecture. To a large extent, the process of creating microservices is not clearly documented. The research in this area of formalizing the process of modeling and operating microservices is still in its infancy. Additionally, eventhough a lot of industries such as amazon, netflix etc. are following this architecture, the process of modeling and implementing microservices is not clear. The purpose is to provide a clear understanding about the process of designing microservices by focusing on the following questions.

Research Questions

1. How are boundary and the size of microservices defined?
2. How to map business capabilities to define microservices?
3. What are the best practices to tackle the challenges introduced by microservices?
   a) How does the collaboration among microservices happen?
   b) How to deploy and maintain microservices independently when there are dependencies among them?
   c) How to monitor microservices?

1.4 Research Approach

A research is an iterative procedure with the goal of collecting as many relevant documents as possible so that the research questions could be answered rationally. Therefore, it becomes very important to follow a consistent research procedure. For the current research, the approach is chosen by refering to [np07]. It consists of two major phases. a. Data Collection Phase and b. Data Synthesis Phase
The following sections explain each of the phases in detail.

1.4.1 Data Collection Phase

In this phase, research papers related to the reseach questions are collected. The Figure 1.6 shows basic steps of this phase.

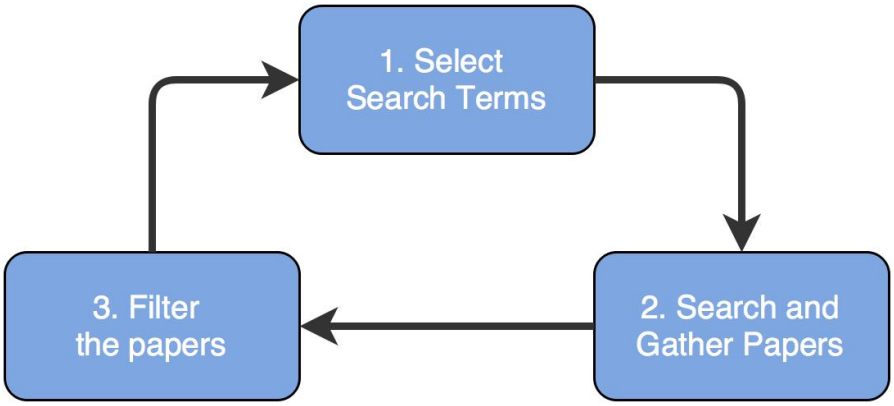

Figure 1.6: Data Collection Phase

1. **Select Search Terms**
   At first, various search terms which defines the research topics and questions are selected using the following strategies.
   a) keywords from research questions and various definitions
   b) synonyms of keywords
   c) accepted and popular terms from academics and industries
   d) references discovered from selected papers

   A consise list of initial keywords which are selected from various definitions of microservices are listed in Table 1.2.

2. **Search and Gather Papers**
   The search terms selected in the previous step are utilized to discover various research papers. In order to achieve that, the search terms are used against various resources listed below.
   a) google scholar
   b) IEEExplore
   c) ACM Digital Library
   d) Researchgate

e) Books

f) Technical Articles

3. **Filter the Papers** Finally, the research papers collected from various resources are filtered first to check if they have profound base to back up their result. Using only authentic papers is important to provide rational base to the current research. The various criteria used to filter the papers are listed below.
   a) Is the paper relevant to answer the research question?
   b) Does the paper have a good base in terms of sources and provide references of the past studies?
   c) Are there any case studies or examples provided to verify the result of the reseach?

1.4.2 Data Synthesis Phase

During the data synthesis phase, data is gathered from the selected research papers to create meaningful output and provide direction to the research. The Figure 1.7 shows various steps within this phase.

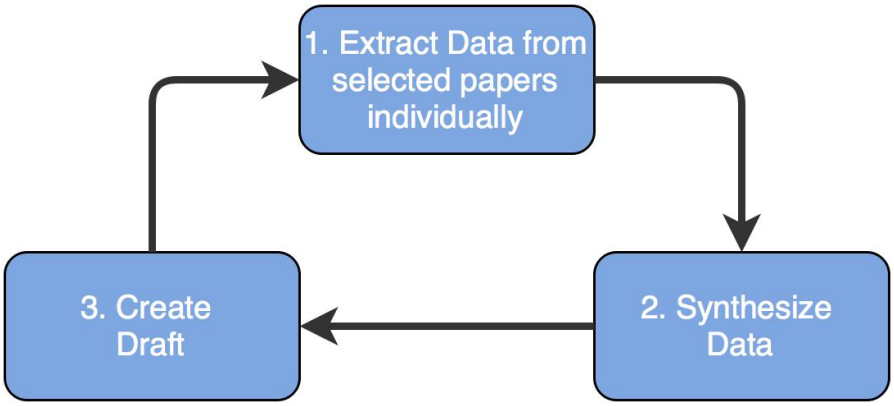

Figure 1.7: Data Synthesis Phase

1. **Extract Data from selected papers individually**
   Firstly, each research paper is scanned and important data relevant to research are collected.

2. **Synthesize Data**
   The data collected from each paper are revised and compared for their similarities

in concepts as well as differences in opinion. These individual content from all research papers are then synthesized.

3. **Create Draft**
   Finally, draft for the observations are created for future reference which will be used later for creating the final report.

1.5 Research Strategy

The Section 1.4 emphasizes the importance of having a consistent research procedure. Additionally, it is also important to define a good strategy for achieving goals of the research. A good strategy is to narrow down the areas for conducting research. In this section, a list of areas is identified.
The various definitions in Section 1.2.2 show that the authors have their own interpretation of microservices but at the same time agree upon some basic concepts. Nevertheless, each definition can be used to understand different aspect of the microservices. A distinct set of keywords can be identified from these definitions which represents different aspects. The Table 1.2 shows various keywords to focus. Additionally, there are other columns in the table which represents various aspects of microservices architecture. These columns are checked or unchecked to represent the relevance of each keyword.

#	keywords	size	Quality of good microservice	communication	process to model microservices
1	Collaborating Services			✓	
2	Communicating with lightweight mechanism like http			✓	
3	Loosely coupled, related functions		✓	✓	
4	Developed and deployed independently				✓
5	Own database		✓		✓
6	Different database technologies				✓
7	Service Oriented Architecture		✓		✓
8	Bounded Context	✓	✓		✓
9	Build around Business Capabilities	✓	✓		✓
10	Different Programming Languages				✓

Table 1.2: Keywords extracted from various definitions of Microservice

Finally, answering various questions around these keywords is the **first approach** to understand the topics shown in columns such as size, quality of microservices etc.
The **next approach** is to understand various drivers for defining microservices architecture. According to [Bro15], the important drivers for software architecture are:

1. **Quality Attributes**
   The non-functional requirements have high impact on the resulting architecture. It is important to consider various quality attributes to define the process of architecture.

2. **Constraints**
   There are always limitations or disadvantages faced by any architectural approach. The better knowledge of the constraint and moreover the solution to these constraints is useful in the process of explaining the architecture.

3. **Principles**
   The various principles provides a consistent approaches to tackle various limitations. They are the keys to define guidelines.

So, in order to define the process of modeling microservices, the key aspects related to **quality attributes**, **constraints** and **principles** is studied thoroughly.

Considering both the Table 1.2 and List 1.5, the first area is to understand various quality attributes influencing microservices architecture. Additionally, the process of identifying and modeling individual microservices from problem domain is defined. These areas are first reseached based on various academic research papers. Then, the approach used by the industry for adopting microservices architecture is studied in order to answer the research questions. The industrial case study is performed in SAP Hybris. Finally, the various constraints and challenges which affect microservices is identified. The results of the previous research is used to create various principles for implementing microservices architecture. These principles provide sufficient ground to create guidelines for modeling and operating microservices.

1.6 Problem Statement

Considering the case that the size of microservice is an important concept and is discussed a lot, but no concrete answer about what defines the size of a microservice. Moreover, the idea of size has a lot of interpretations and no definite answer regarding how small a microservice should be. So, the first step is to understand the concept of granularity in the context of microservices.

List of Figures

List of Tables

Bibliography

[Abr14] S. Abram. *Microservices*. Oct. 2014. URL: http://www.javacodegeeks.com/2014/10/microservices.html.

[Ann14] R. Annett. *What is a Monolith?* Nov. 2014.

[Bro15] S. Brown. "Software Architecture for Developers." In: (Dec. 2015).

[Coc15] A. Cockcroft. *State of the Art in Mircroservices*. Feb. 2015. URL: http://www.slideshare.net/adriancockcroft/microxchg-microservices.

[DMT09] E. M. Dashofy, N. Medvidovic, and R. N. Taylor. *Software Architecture: Foundations, Theory, and Practice*. John Wiley Sons, Jan. 2009.

[FA15] M. T. Fisher and M. L. Abbott. *The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise, Second Edition*. Addison-Wesley Professional, 2015.

[FL14] M. Fowler and J. Lewis. *Microservices*. Mar. 2014. URL: http://martinfowler.com/articles/microservices.html.

[Gup15] A. Gupta. *Microservices, Monoliths, and NoOps*. Mar. 2015.

[Mac14] L. MacVittie. *The Art of Scale: Microservices, The Scale Cube and Load Balancing*. Nov. 2014.

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

[np07] np. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Tech. rep. Keele University, 2007.

[NS14] D. Namiot and M. Sneps-Sneppe. *On Micro-services Architecture*. Tech. rep. Open Information Technologies Lab, Lomonosov Moscow State University, 2014.

[Ric14a] C. Richardson. *Microservices: Decomposing Applications for Deployability and Scalability*. May 2014.

[Ric14b] C. Richardson. *Pattern: Microservices Architecture*. 2014.

[Ric14c] C. Richardson. *Pattern: Monolithic Architecture*. 2014. URL: http://microservices.io/patterns/monolithic.html.

[RTS15] G. Radchenko, O. Taipale, and D. Savchenko. *Microservices validation: Mjolnirr platformcase study*. Tech. rep. Lappeenranta University of Technology, 2015.

[Woo14] B. Wootton. *Microservices - Not A Free Lunch!* Apr. 2014. URL: `http://highscalability.com/blog/2014/4/8/microservices-not-a-free-lunch.html`.