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 Context

1.1 Monolith Architecture Style

A Mononlith Architecture Style is the one in which an application is deployed as a single artifact. The architecture inside the application can be modular and clean. In order to clarify, the figure 1.1 shows architecture of an Online-Store application. The application has clear separation of components such as Catalog, Order and Service as well as respective models such as Product, Order etc. Despite of that, all the units of the application are deployed in tomcat as a single war file.[Ric14a][Ric14c]

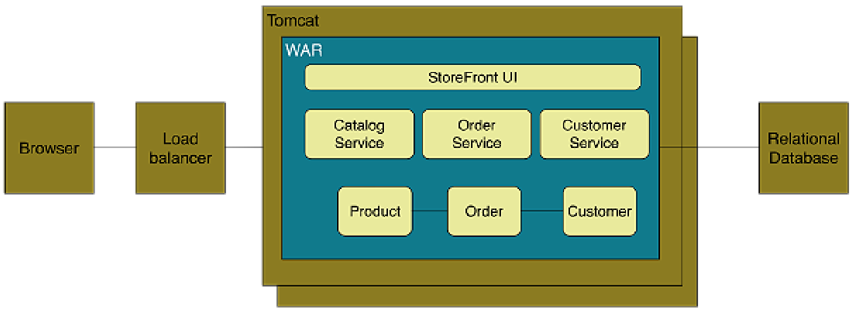

Figure 1.1: Monolith Example from [Ric14a]

1.1.1 Types of Monolith Architecture Style

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 Module Monolith Architecture. An example is show in figure 1.2. The application on the left has all the code in the same codebase in the form of packages and classes without clear definition of modules and get compiled to a single artifact. However, the application on the right is

developed by a number of modular codebase, each has separate codebase and can be compiled to different artifact. The modules uses the produced artifacts which is different than the earlier case where the code referenced each other directly.

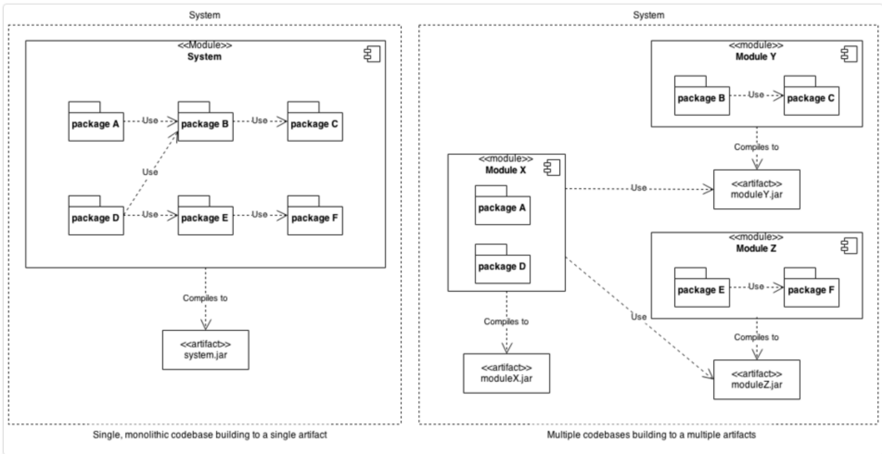

Figure 1.2: Module Monolith Example from [Ann14]

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

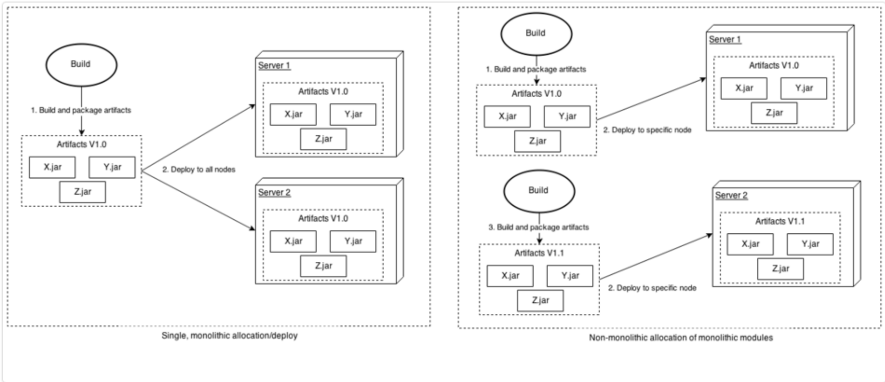

Figure 1.3: Allocation Monolith Example from [Ann14]

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

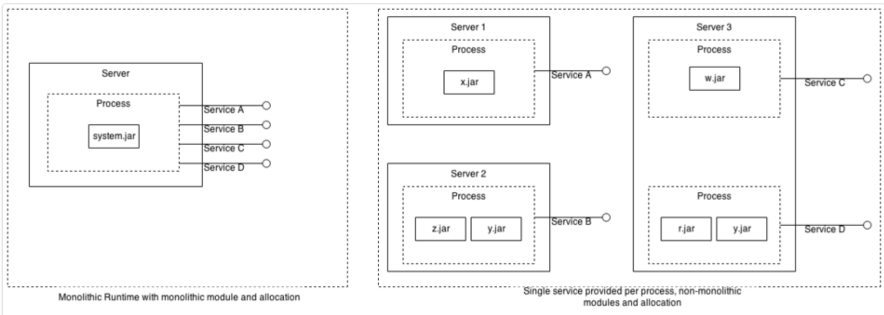

Figure 1.4: Runtime Monolith Example from [Ann14]

1.1.2 Advantages of Monolith Architecture Style

The Monolith architecture is appropriate for small application and has following benifits:[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. Nevertheless, 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 Monolith Architecture Style

As the requirement grows with time, alongside as application becomes huge and the size of team increases, the monolith architecture faces many problems. Most of the advantages of monolith architecture for small application will not be valid anymore. The challenges of monolith architecture for such agile and huge context are as given below:[NS14][New15][Abr14][Ric14a][Ric14c][Gup15]

- Limited Agility: As the whole application has single codebase, even changing a small feature to release it in production takes time. Firstly, the small change can also trigger changes to other dependent code because in huge monolith application 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 production, the whole application has to be deployed. Thus continuous delivery gets slower in case of monolith application. This will be more problematic when multiple changes have to be released on a daily basis. The slow pace and 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. 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, the 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. So, in overal it will slow down the speed of understandability, execution and testing.

- Difficult Team Structure: The division of team as well as assigning tasks to the team can be tricky. Most common ways to partition teams in monolith are by technology and by geography. However, each one cannot be used in all the situations. In any case, the communication among the teams can be difficult and slow. Additionally, it is not easy to assign vertical ownership to a team from particular feature from development to relase. If something goes wrong in the deployment, there is always a confusion who should find the problem, either operations team or the last person to commit. The approprate team structure and ownership 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 architecture need to follow the same techonology stack. However, if the requirement changes then there can be situation when the features can be best soloved by different sets of technology. Additionally, not all the features in the application are same so cannot be treated accordingly in terms of technology as well. Nevertheless, the technology advances rapidly. So, the solution thought at the time of planning can be outdated and there can be a better solution available. In monolith application, it is very difficult to migrate to new technology stack and it can be rather painfull process.

- Limited Scalability: The scalability of monolith application can be done 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 the 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 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 and does not improve resiliency.[Mac14][NS14]

1.2 Decomposition of Application

The section 1.1.3 specified various disadvantages related to monolith architecture style. The book [FA15] provides a way to solve most of the discussed problems such as agility, scalability, productivity etc. It 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.

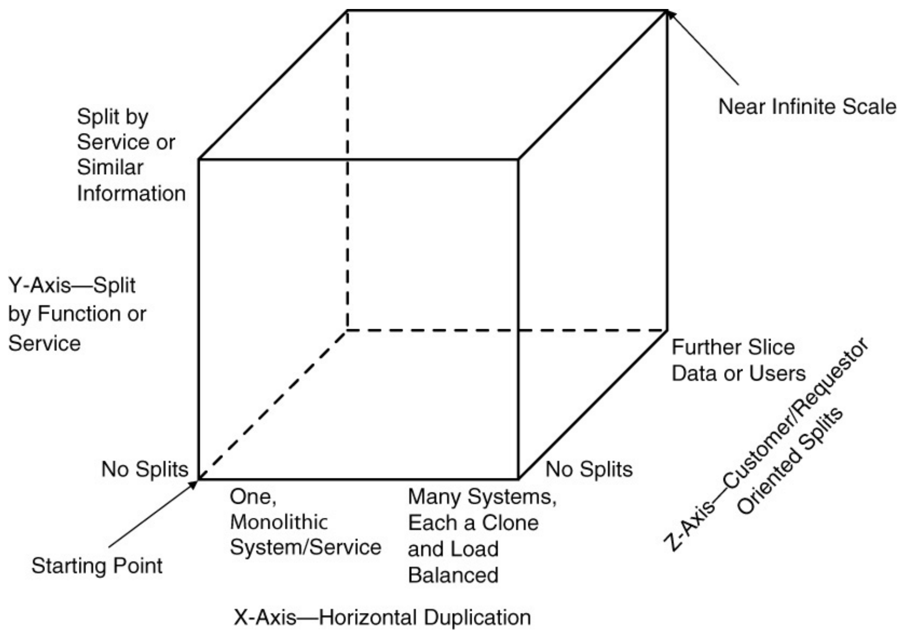

Figure 1.5: Scale Cube from [FA15]

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

1. X-axis Scaling: It is done by cloning the application and data along multiple servers. A pool of requests are applied into a load balancer and the requests are deligated to any of the servers. Each of the server has the full capability of the application and full access to all the data required so 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 is 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 or there dominant requests types 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 done by spliting the request based on certain critera 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 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 given added functionality or a new functionality can be tested to a small group and thus minimizing the risk.

3. Y-axis Scaling: The scaling along this dimension means the spliting of the application responsibility. The separation can be done either by data, by the actions performed on the data or by 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 were rather duplication of work along servers, the 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 on 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.

1.3 Microservice Architecture Style

The section 1.1.3 listed various disadvantages of following monolithic architecture style especially when the application grows in size rapidly. To counteract those issues, the section 1.2 proposed a way of using scale-cube to decompose the application into individual features, each feature being scaled individually. The same approach is utilized by Microservice Architecture Style, in which an application is decomposed into various individual components, each component runs in a separate process and can be deployed as well as scaled individually.
There are several definitions given by several pioneers and early adapters of the style.

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]

" Microservice 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."

The authors have their own way of interpretation of microservices but at the same time agree upon some basic concepts regarding the architecture. However, each definition can be used to understand different aspect of the microservices. A distinct set of keywords can be identified which represents different aspects pointed by the authors and various concepts they agree. Moreover, the table lists important keywords. These keywords provide us some hint regarding various topics of discussion related to microservices, which are listed in columns. Finally, understanding these keywords can be one of the approaches to understand microservices architecture and answer various questions.

#	keywords	size	Quality of good microservice	communication	process to create 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 1.1: Keywords extracted from various definitions of Microservice

2 Architecture at Hybris

2.1 overview

SAP YaaS provides a variety of business services to support as well as enhance the products offered as SAP hybris front office such as hybris Commerce, hybris Marketing, hybris Billing etc. Using these offered services, developers can create their own business services focussed on their customer requirements.
The figure 2.1 provides the overview of YaaS. YaaS provides various business processes as a service (bPaaS) essential to develop applications and services thus filling up the gap between SaaS and HCP. For that purpose, it consumes the application services (aPaaS) provided by HCP.

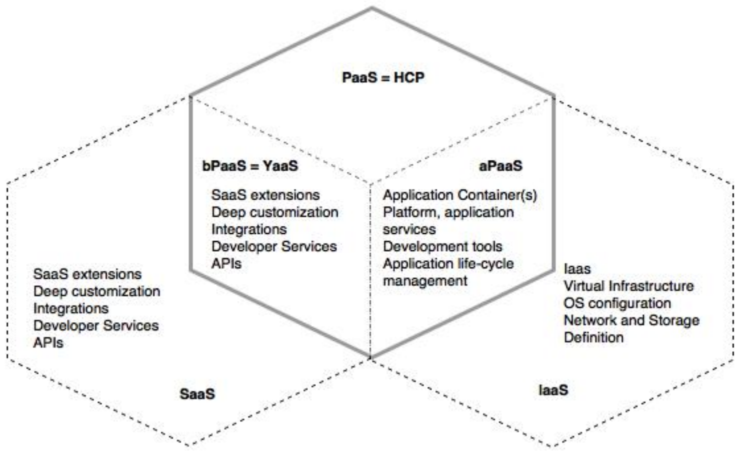

Figure 2.1: YaaS and HCP [Hir15]

2.2 Vision

The vision of YaaS can be clarified with the following statement.

"A cloud platform that allows everyone to easily develop, extend and sell services and applications."
[Stu15]

The vision can be broadly categorized into following objectives.

1. **Cloud First**
   The different part of the application need to be scaled independently.
2. **Autonomy**
   The development teams should be able to develop their modules independent of other teams and able to freely choose the technology that fits the job.
3. **Retain Speed**
   The new features should be able to be released as fast as possible.
4. **Community**
   It should be possible for the components to be shared across internal and external developers.

Acronyms

CRUD create, read, update, delete.

DEP Dependency.

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.

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.

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

[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.

[Hir15] R. Hirsch. *YaaS: Hybris' next generation cloud architecture surfaces at SAP's internal developer conferences*. Mar. 2015. URL: http://scn.sap.com/community/cloud/blog/2015/03/03/yaas-hybris-next-generation-cloud-architecture-surfaces-at-sap-s-internal-developer-conferences.

[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.

[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.

[Stu15] A. Stubbe. *Hybris-as-a-Service: A Microservices Architecture in Action*. May 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`.