IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, April 2022
Impact Factor: 6.252

An Overview of Microservice Architecture Impact in Terms of Scalability and Reliability in E-Commerce: A Case Study on Uber and Otto.De

Janita J. Phatak
Department of Informational Technology
S. S. and L. S. Patkar Varde College, University of Mumbai, Mumbai, Maharashtra, India
phatakjaita@gmail.com

I. INTRODUCTION

A Microservices architecture consists of a collection of services that are small and autonomous. Each service is self-contained and should imply a single business capability. Thus, these applications built on microservices architecture are loosely coupled. The basic idea of implementing an microservice architecture is that when an application is broken down into small, composable pieces or components that work together are easy to deploy, create, and maintain. Each service has a separate codebase structure, thus can be managed by a small group of development team. This is the opposite to Traditional monolithic architectural style where the final product of application is developed all in single and indivisible unit. In monolithic architecture, all features of the application are written as separate modules which are then packaged in a single main application. We should use microservice architecture when there is a large application and need high release speeds, high scalability, high resilience, easier debugging and maintenance.

From an e-commerce perspective, today's competitive world of e-commerce requires various strategies to increase revenue and customer satisfaction. E-commerce companies are required to share various services to achieve flexibility. They should have reusability services, automated service deployments, and fast service scalability. Today, monolithic applications in e-commerce, are becoming a barrier to innovation as they are deployed at once and need to be checked and tested at the end, it comes with complexity and flexibility. Microservices are the ultimate risk-free way to build e-commerce platforms for traffic peaks, as well as implement and test new trends, such as new payment methods, voice assistants or progressive web apps. Micro-service can also be used to set up complex omnichannel systems. In order to meet your customers' expectations in omnichannel systems, you need to gather all the information about products, shipments, stocks and orders and keep them up to date. Microservices enable companies to use API gateways that integrate POS, ERP, or WMS solutions that are in the best range and synchronize them with existing processes.

The concept of microservices was coined by the evangelists of microservices, with Martin Fowler at the forefront and global companies facing the wall in terms of business scalability, agility and speed of implementation of changes [1]. Amazon, a provider of major online marketplaces, was one of the first. After this, Other leading e-commerce companies have transformed their infrastructure into micro-services. eBay, Coca Cola, Netflix, Spotify, Uber, Etsy, Gilt and Zalando, just to name a few, used microservices to create a flexible, global system and a whole new work culture, easy to access and inspiring for developers.

Figure 1. E-Commerce platforms using Microservice Architecture [2]

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, April 2022
Impact Factor: 6.252

II. BACKGROUND

2.1 The Advantages of Microservices

A. Independent Usage

Each service will have a codebase. You can also separate the database layer. One service can be tested independently without worrying about the entire product. You can also choose different techniques to use in each service. Security can be different for everyone.

B. Improved defect isolation

Large applications can be remained unaffected by the failure of a single module.

C. Fast Deployment

Microservices architecture allows rapid development and deployment as it simplifies design. Each module is easily created, tested and deployed independently, thus giving your business increased agility and faster time to market.

D. Scalability

Microservices offers ease of scalability according to the architectural design because the modules work independently and the context of each is limited. The design is much simpler allowing new features to be added faster. Each module is tested separately, relying on DevOps methods and test automation, and the entire system is automatically tested in a matter of minutes, thus reducing the time to create, test, and release new features [3].

At the infrastructure level, each micro-service is measured and managed separately. You do not need to scale the entire system in terms of peak time and load. Only micro services are counted under load. This is done automatically without any human intervention with self-scaling capabilities and self-healing.

E. Reliability

A successful microservice architecture prevents any system failure. Its repeatable automation, and designing pattern features keep the system running.

2.2 Guidelines for Implementing Microservices Architecture

As a developer, when you decide to build an application, separate the domain and be clear about the functionality.

- Each micro service you design will focus on just one service of the application.
- Make sure you design the application in such a way that each service is individually deployable.
- Make sure that the communication between the microservices is done through a stateless server.
- Each service can be redeveloped into smaller services with their own microservices.

2.3 Microservices Architecture in Ecommerce

Electronic commerce, also known as e-commerce, is a way of buying or selling products on web or online services. Technologies such as m-commerce (mobile commerce), electronic fund transfer, online transaction processing, internet marketing, electronic data interchange (EDI), there are supply chain management, inventory management systems and automated data collection systems included in e-commerce.

Microservices architecture is used as a separate component or assembly connected by microservices, REST API, to create ecommerce applications. Multiple user interfaces can be created using the same backend microservices. While the features of this architectural approach are of major interest to development teams, its benefits to businesses are numerous. Small development teams can work simultaneously to create different services for faster application implementation and market entry. Migration into modular ecommerce architecture is an investment in instalments. You can rebuild and modernize your eCommerce solution step-by-step by replacing each business function with a micro-service. You can start with areas where custom workflows or designs can influence the customer experience and therefore lead to the highest sales. Scaling a microservices-based ecommerce application is easier and less expensive because each service lives through its own lifecycle - it is created, modified, tested, and (if necessary) removed separately from other services. This is beneficial for growth-focused companies that are planning to invest and gradually develop their e-commerce presence. Ecommerce applications

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, April 2022
Impact Factor: 6.252

(and therefore online sales) are more flexible because malformations within a single microservice do not interfere with the entire application. The cost of infrastructure can be optimized because microservices are cloud-native and each service can be hosted on a different cloud instance based on its bandwidth requirements.

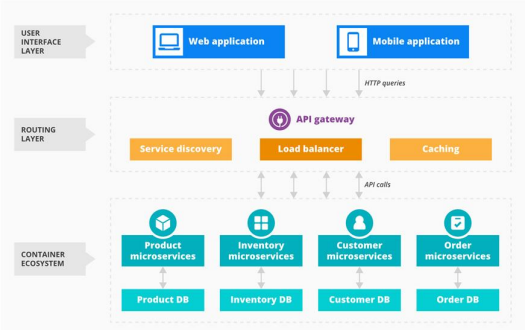

Figure 2: Microservice Architecture in ecommerce [4]

- The user interface layer is used to create multiple digital customer touchpoints using the same microservices on the back-end.
- The routing layer connects HTTPS queries to the corresponding microservices.
- API Gateway is used to build any amount of API.
- Service Discovery is used to find dynamically assigned microservices instances network locations.
- Load balancers are used to deliver API calls in micro services.
- Caching is used to store and return static data (e.g. text files) for faster uploading of web pages and good customer experience.
- The container ecosystem stores units of micro services. Microservices are built around specific business contexts: data types, responsibilities, functions.

2.4 Common Microservices Used in Ecommerce

There are various types of micro-services available in e-commerce. Some examples of microservices include:

A. Order Micro-Service

Order Micro-Services enables direct access to all orders on the channel in which your business operates, With web stores, mobile apps, IoT devices. There are micro-service orders Functionality that allows your business to maintain and route, cancel, refund, satisfy, or duplicate orders

B. Pricing Micro-Service

Micro-service ("pricing") allows the pricing manager to create and manage multiple price lists. Each price list has its own currency and its own individual alternative price types.

C. Customer Micro-Service

Customer Micro-Services enables your business to manage all customer information, such as registration. Allows you to store past orders, addresses and privacy and consent preferences and everything about your customers in one place.

D. Merchandising Micro-Service

Merchandising is a micro-service that gives merchants and managers the ability to create and manage product presentations on the storefront. Merchants can create and product catalog pages using static and dynamic (rule-based) categories, like product detail pages, category landing pages, site navigation and other digital experiences.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, April 2022
Impact Factor: 6.252

III. Related Work

3.1 Microservices at Uber

A. Uber's Previous Architecture

Like most startups, Uber embarked on a journey through a single city with a monolithic architecture designed for a single offer. It had one codebase at that time, and it was sufficient to solve Uber's core business problems. However, as Uber began to expand worldwide, it faced various challenges in terms of scalability and continuous integration.

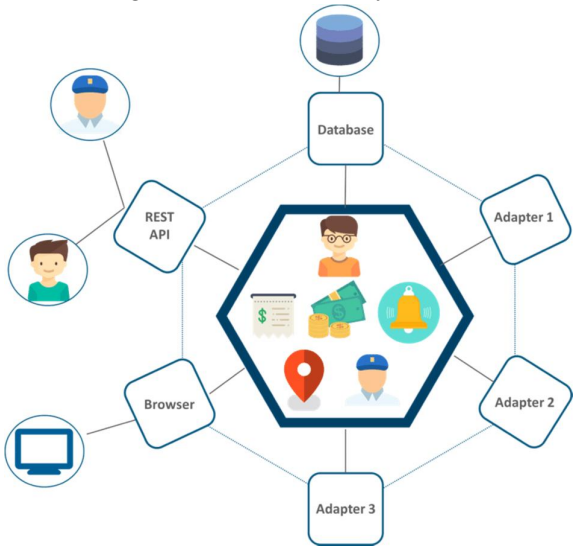

Figure 3: Uber's Monolithic architecture [5]

The above diagram shows Uber's previous architecture.

- The passenger and driver connect with the REST API.
- They use three different adapters with APIs, such as billing, payment, email / messaging, which we see when we book a cab.
- They use a MySQL database for storing all their data.

Therefore, if you notice all the features like passenger management, billing, notification features, payment, trip management and driver management have been created in one framework.

B. Problem Statement

This type of framework posed various challenges as Uber began to expand worldwide. The following are some of the major challenges

- All features needed to be recreated, deployed, and re-tested to update a single feature.
- Fixing a bug in a repository was extremely difficult because developers had to change the code over and over again.
- With the introduction of new features from worldwide, scaling all features simultaneously becomes very difficult to handle at a single place.
- Availability Risks. A single regression within a monolithic code base can bring the whole system (in this case, all of Uber) down.
- Risky, expensive deployments. These were painful and time consuming to perform with the frequent need for rollbacks.
- Poor separation of concerns. It was difficult to maintain good separations of concerns with a huge code base. In an exponential growth environment, expediency sometimes led to poor boundaries between logic and components.
- Inefficient execution. These issues combined made it difficult for teams to execute autonomously or independently.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, April 2022
Impact Factor: 6.252

C. Solution

To avoid such problems, Uber decided to change its design and follow Amazon, Netflix, Twitter and many other hyper-growth companies. Thus, Uber decided to break its monolithic architecture into multiple codebases to create microservice architectures.

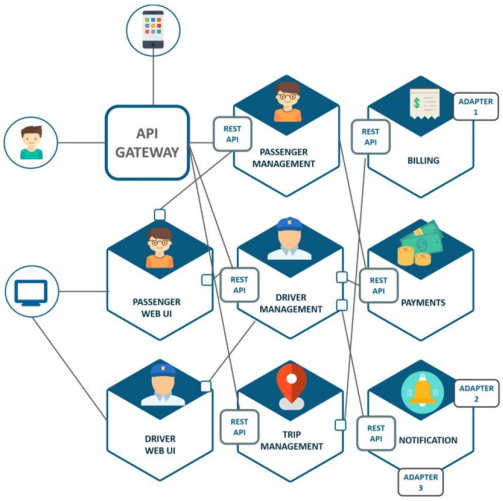

Figure 4: Uber's Microservice architecture [6]

The above diagram depicts Uber's Microservice architecture.

- The major change we see here is the introduction of the API gateway through which all drivers and passengers are connected. From the API gateway, all internal points like passenger management, driver management, trip management and others are connected.
- Units are separate deployable units that perform separate functions.
- For example: if you want to change anything in Billing Microservices, you only need to deploy Billing Microservices and no other deployments.
- All attributes are now measured individually, meaning that the interdependence in each attribute is removed.

For example, we all know that the number of people looking for a cab is actually higher than the number of people actually booking and paying for a cab. From this we conclude that the number of processes working on the travel management micro service is more than the number of processes working on payment.

D. Results

Uber deployed so many microservices till now. Following picture shows its growth:

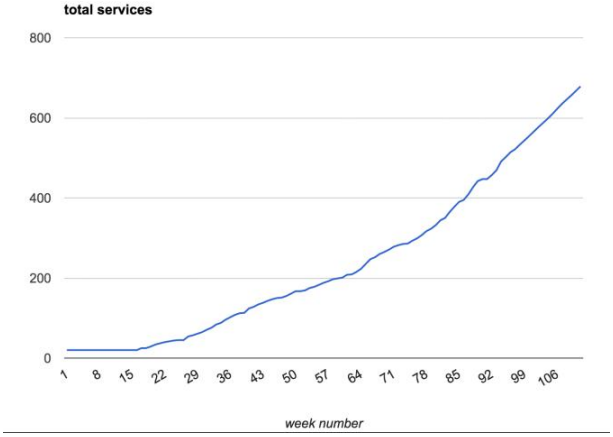

Figure 5: Uber's Microservices deployment growth [6]

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, April 2022
Impact Factor: 6.252

Matt Ranney is the Chief Systems Architect at Uber and was previously a founder and CTO of Voxer. At QCon San Francisco, he gave a talk called Scaling Uber [6].

Scaling from monolithic to microservices: Horizontal scaling means scaling by adding more machines to your pool of resources (also described as “scaling out”), whereas Vertical scaling refers to scaling by adding more power (e.g. CPU, RAM) to an existing machine (also described as “scaling up”).

Uber used Horizontal scaling. Currently Uber supports Java, Python, Node,js , Go.

3.2 Microservices at Otto.De

OTTO has been one of the most successful e-commerce companies for over 60 years and the largest online retailer of fashion and lifestyle products for end customers in Germany.

With a turnover of over 2.563 billion euros and 1 million daily visitors in the business year 2015/2016, otto.de is one of the largest online stores in Europe. In 2011, Otto relaunched its e-commerce software from scratch. The drivers for this decision were primarily non-functional requirements such as scalability, performance and fault tolerance. In terms of scalability and agility, they weren't just thinking about technical scalability in terms of load or data. In particular, a measurable solution was needed in relation to the number of teams and / or developers working on the software at a given time. In addition, it was planned to practice DevOps with continuous deployment, in order to deliver features quickly to customers.

What was found initially was somewhat unusual, but in the end highly successful: Instead of setting up a single development team to create a new platform for the shop, Otto was actively employing Conway’s Law by starting development with initially four separate teams. Consequently, they were not building a single, monolithic application, but a vertically decomposed system consisting of four loosely coupled applications: Product, Order, Promotion, and Search/Navigation. In the following years, Otto founded more teams and systems. Today, there are 18 Teams working on 45 different applications in 12 so-called “verticals” as illustrated in Figure 6.

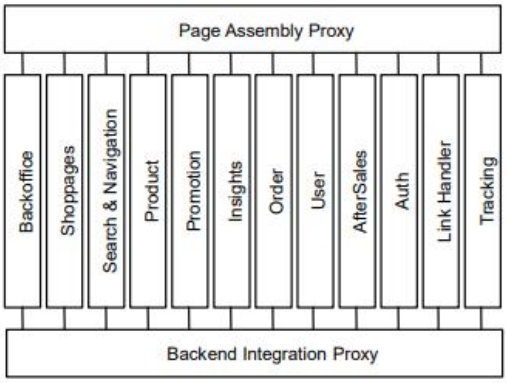

Figure 6: Current Vertical Decomposition at otto.de [7]

Vertical is part of the platform responsible for single bound context in the business domain [8]. Verticals can be as small as microservices, but most of the time, they are more rugged. Vertical communication is allowed only by accessing the REST API using the "Backend Integration Proxy" in the background - see Figure 1. This makes it easy to make sure that slow or unavailable apps can't destroy other apps or the entire store with a snowball effect.

The vertical adheres to the "nothing to share" principle: they do not share states, there are no infrastructure on either side of the two proxies, no databases or other shared resources. Does not use vertical HTTP sessions, shared cache or similar. Only a very limited number of client-side states (using cookies or local storage) are shared across different systems to gain a general understanding of who is entering the store. The major advantages of a shared-nothing architecture are excellent horizontal scalability and improved defect-tolerance. The reason for this is obvious: if two elements do not share anything, they cannot have a negative effect on each other.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, April 2022
Impact Factor: 6.252

A. Integrating Verticals on otto.de

All shop pages have different vertical pieces: preview of shopping cart, navigation structure, maybe some products or other parts. To assemble these pieces, the following principles are used for "Page Assembly Proxy" [9]:

- Pieces that are not part of the primary content or pieces that are initially invisible are assembled on the client side using AJAX [10]]. The shopping cart preview, for example, is one of those features that is included on almost every page
- The primary content is aggregated on the server side using Edge Side Includes [11] resolved by Varnish Reverse Proxy (www.varnish-cache.org).

In this way, verticals are integrated into a one-page website on the user interface. Despite the backend, users experience the store as a consistent entity.

B. Communication Among Verticals at otto.de

All verticals contain redundant data using pull-based data replication. This ensures that the content can be delivered without the other vertical access during the vertical request. On otto.de, the pull principle is applied in conjunction with Atom feeds (tools.ietf.org/html/rfc4287) via the Apache Kafka High-throughput Distributed Messaging System (kafka.apache.org).

C. Vertical and Microservices on otto.de

Microservices, like other software components, must be designed for proper granularity [12]. Vertical domains, as described above, may be small enough to be executed as a microservice - but they can also be large. Thus, sometimes those verticals need to be further refined: if possible, by extracting independent new features from the existing code into a new vertical (ideally as a microservice) or by cutting verticals into a distributed system of micro-services.

D. Scaling Delivery Pipeline at otto.de:

For frequent and automatic deployment, continuous deployment pipelines [13] are used for each application. Each vertical commit is first checked, compiled, packaged, deployed, and tested in a continuous-integration phase. After all the tests are passed, the container is deployed to the next stage, called testing. This phase is used to run load and integration tests. A second set of automated (and some manual) integration tests is implemented to ensure compatibility with newer and older versions of deployed software. The final step is to deploy application directly into the atmosphere. Due to the large number of deployment pipelines on otto.de, pipeline are implemented, described and operated with the internal domain-specific language LambdaCD (www.lambda.cd). Because the LambdaCD pipeline is nothing more than a microservices responsible for creating, testing, and deploying a single application, they operate in the same infrastructure as other microservices. They can be tested, run locally on notebooks without any additional continuous-integration or application server, and - in particular - they can be debugged just like any other software system.

E. Agility and Reliability on otto.de:

On otto.de, most microservices are deployed fully automatically after each push on the version control system. Automation is the key to DevOps success: automatic building of systems out of version management repositories; Automated implementation of unit tests, integration tests, and system tests; Automated deployment in testing and production environments. Since the beginning of 2015, more and more micro services have been launched in Vertical. Meanwhile, the number of direct deployments has increased from 40 to more than 500 deployments per week, which says that scalability has increased. Figure 7, the blue line, shows the number of live deployments per week for the years 2015-2017. At the same time, reliability is maintained and improved: the number of live events remains the same, very low levels; See Figure 7, red stripes. Incidents have been calculated since 2015, thus no incident data is available for 2014. However, this suggests that the quality assurance measures implemented for continuous integration and deployment are indeed effective for reliability.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, April 2022
Impact Factor: 6.252

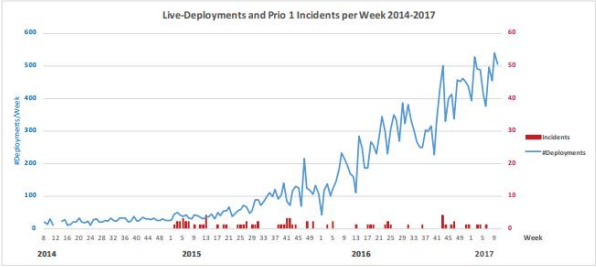

Fig. 2. Number of life deployments per week at otto.de over the last two years. Despite the significant increase of deployments, the number of live incidents remains on a very low level.

Figure 7: otto.de's Microservices deployment growth [7]

IV. CONCLUSION

In recent years, micro-services are becoming a large standard. As they have many benefits, they are super helpful within the era of e-commerce. We discussed the benefits of microservices than monolithic architecture. We have an overview of the Microservice Architecture used in e-commerce. Microservice Architectures can enable scalable, agile and reliable software system, such as otto.de's successful re-execution is mentioned above. Other E-Commerce or M-Commerce System, such as Amazon, Uber also adopted microservice architecture. We at otto.de discussed how coupling, integration, scalability of entire application as well as scalability for deployed pipelines for continuous delivery, and microservices are developed in teams. Micro-Services use genus API and Communication protocols to move along with each other, however they do not believe each other. Micro-Services Design means every micro-service separately developed, maintained and deployed by each individual the group. This type of single-responsibility result is also an alternative edge. The Applications made of micro-services are higher scales, because when you are required, you can scale them one-one. Besides, there is more failure tolerance for services.

V. ACKNOWLEDGMENT

I would like to express my deep and sincere gratitude to my research supervisor Mrs. Manali Patil, M.Sc. Professor, Information Technology, Patkar College for giving me the opportunity to do research and providing in valuable guidance throughout this research. I would also like to thank other professors, my parents who helped me a lot in finishing this research within the limited time frame.

REFERENCES

[1]. MACH eCommerce Blog by Divante https://www.divante.com/blog/what-are-microservices-introduction-to-microservice-architecture-for-ecommerce-2
[2]. Microservices Use Cases Blog by Alpacked.io, AUG 10, 2020, https://alpacked.io/blog/microservices-use-cases/
[3]. Benefits of Microservices for Your Business Agility Blog by sumerge, November 2, 2020 https://www.sumerge.com/benefits-of-microservices/
[4]. Microservices-Based Architecture in Ecommerce | Modular Solutions Blog by Science Soft https://www.scnsoft.com/ecommerce/microservices
[5]. Blog on Microservices Zone,by Sahiti Kappagantula,Jul. 03, 18.https://dzone.com/articles/microservice-architecture-learn-build-and-deploy-a
[6]. "Uber Rush and Rebuilding Uber's Dispatching Platform" https://qconnewyork.com/ny2015/system/files/presentation-slides/uberRUSH.pdf
[7]. "Microservice Architectures for Scalability, Agility and Reliability in E-Commerce" , IEEE ICSAW Conference:2017
[8]. E. Evans, Domain-driven design. Addison-Wesley, 2004.
[9]. G. Steinacker, "On monoliths and microservices," 2015, http://dev.otto. de/2015/09/30/on-monoliths-and-microservices/.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, April 2022
Impact Factor: 6.252

[10]. E. Woychowsky, AJAX: Creating Web Pages with Asynchronous JavaScript and XML. Prentice Hall, 2006.
[11]. M. Tsimelzon et al., "ESI language specification," 2001, w3C Note 04 August 2001, https://www.w3.org/TR/esi-lang.
[12]. W. Hasselbring, "Component-based software engineering," in Handbook of Software Engineering and Knowledge Engineering. World Scientific Publishing, 2002, pp. 289–305.
[13]. J. Humble and D. Farley, Continuous Delivery. Pearson, 2010.