Audience: Technical decision makers

Microservices: A paradigm shift for fast-growing e-commerce businesses

commercetools

Management Summary

Speed is today’s leading competitive differentiator: Facing changing customer demands, the rise of the mobile web and ever shorter innovation cycles, merchants need to make sure to have both an organizational as well as a technical structure that allows for agility and speed. Monolithic e-commerce applications, which still drive most of the biggest retailers’ sites worldwide, are bottlenecks for innovation.

Companies such as *Amazon, eBay, Netflix, Google* and *Uber* already rely on an architectural principle called Microservices: Small services which are individually developed, deployed and managed and which communicate via standardized APIs.

Instead of having one monolithic application containing all business logic, a flexible network of Microservices handles all complexity. Developers can work on small functional chunks rather than having to understand many millions of lines of code, resulting in better quality as well easier testing and adapting.

Using examples by *Zalando, The Gilt Groupe* and *REWE*, we will show how retailers benefit from switching to Microservices and give some practical guidance regarding the evaluation and introduction of this methodology.

1 Centralized teams and monoliths are everywhere

Back in 2002, Jeff Bezos, CEO of Amazon issued a mandate that would have enormous repercussions for the company's future: He requested that all teams be required to expose their data and functionality through service interfaces and only communicate through these[1]. Also, these services had to be set up in a way that they could be used by external developers. As a result, the company built an infrastructure that would allow them to scale their online retail business as well as provide a hugely successful platform to third-party developers and businesses.

In the digital age, fast-growing, fast-moving corporations need to find both an organizational and a technological structure that allows for agility and speed. However, to date only the most driven businesses have managed to implement such changes.

When it comes to team set-up, e-commerce IT departments traditionally follow a centralized structure with experts organized around technology tiers. So in practice, a database professional might work on checkout in one week, and on product search in another - both two completely separate parts of the application. Such a jack-of-all-trades approach usually does not allow deep and specialized knowledge to evolve and often gives way to mediocrity. Also, if a project requires the involvement of experts across several technology tiers, the result is a massive communications overhead.

According to *Conway's law*[2] , communication paths in an organization have a direct impact on the technical infrastructure. In other words, if a centralized team structure prevails, this will also result in a centralized application. So, not surprisingly, a vast majority of e-commerce sites today are based on monolithic software applications. They share a common code-base and have centralized storage and messaging.

When the first e-commerce systems were launched at the turn of the century, they inherited the monolithic paradigm of contemporary software, such as ERP solutions. They were designed and structured in a way that followed the *zeitgeist* and reflected the technological status quo of the time. Most people accessed the Internet through CRT monitors, and mobile web usage seemed light years away. Also the capability to connect systems via standardized APIs - something we very much take for granted today - was not available as yet. As a result, infrastructure components had to be created from scratch and added to the core of commerce solutions - where they still are today creating considerable technical debts.

Successively, e-commerce software developed into monolithic, full-stack suites, assuming some ERP functionality as well as other parts of the business logic such as WCMS or CRM.

2 New market demands lead to growing pains

Especially in recent years, global commerce has grown exponentially and there is no sign of this development ending anytime soon. Customers worldwide are becoming more demanding, looking for inspiring shopping experiences on all channels and on all devices. Personalization, high service levels and fast delivery are just a few of the challenges merchants are currently facing.

In this fast-moving, fast-growing market environment, the shortcomings of monolithic applications and an outdated software architecture become painfully apparent, especially to ambitious merchants around the globe.

7 reasons why monolithic applications can slow down innovation

1. High degree of software complexity
A commerce application is constantly evolving to keep up with changing demands. As a result, these applications become harder and harder to maintain, let alone be completely understood by the developers working with it.

2. No responsibility
A large commerce application also runs the risk of being perceived as a black-box that nobody fully grasps and that nobody wants to take responsibility for. In other words, there is a strong disincentive for individual developers to touch anything. For example, if they fix a problem with the ORM system, it could break everything else. As a result, inefficient or even faulty code can enter the system.

3. Lack of agility
With monolithic commerce solutions, teams are usually structured according to their individual functions, such as frontend, backend or database. When a request is made that affects all of these teams, the resulting projects can take a great deal of time because tasks have to be shared by and among multiple different team members. As a result, rolling out new features or entering new markets takes too long and leads to missed business opportunities.

4. Fragility
In a centralized architecture, the individual parts are highly coupled and depend on each other. This results in a single point of failure. If one little cog of the clockwork does not work as planned, this can bring down the entire system. As Abel Avram writes: “In a monolithic application, as the number of code changes grows, the risk rises exponentially because of all the dependences that tend to build up in such systems over time.” [3]

5. Inefficient testing
Because of the single points of failure present in these applications, comprehensive and repeated testing is crucial; if only one small part of the application is changed, it needs to be tested in its entirety – also with respect to features that have nothing to do with the original changes. However, because of the software’s internal dependencies, the effort involved in (automatic) testing and quality assurance rises exponentially; concepts such as continuous delivery become almost impossible.

6. No specialization
In a highly coupled application, individual components are usually treated equally with regard to the kind of the number of resources they have access to – regardless of a certain part needing more or less CPU, RAM or disk I/O.

7. Scaling issues
With more and more transactions taking place online, even major retailers struggle to keep pace because scaling their applications becomes harder every day. The recent outages of major retailers during Black Friday speak a clear language. With most monolithic applications, scaling horizontally is the only option, which in turn creates many other issues.

Quite understandably, companies facing the need to move faster and drive innovation are trying to find ways around these restrictions.

3 Microservices to the rescue

The software-oriented architecture community has recently been discussing the idea of Microservices. As a concept, it draws on the methodology of Service-Oriented Architecture (SOA), which has been around for decades. Although there is no bullet-proof academic definition for it, the basic idea is to build small applications with limited functionality which can be deployed separately.

Instead of having a single monolithic application handle all business logic and offer the required features, a Microservice-based approach encapsulates each business capability into individual services and lets them interact with each other
(see diagram 1) [4].

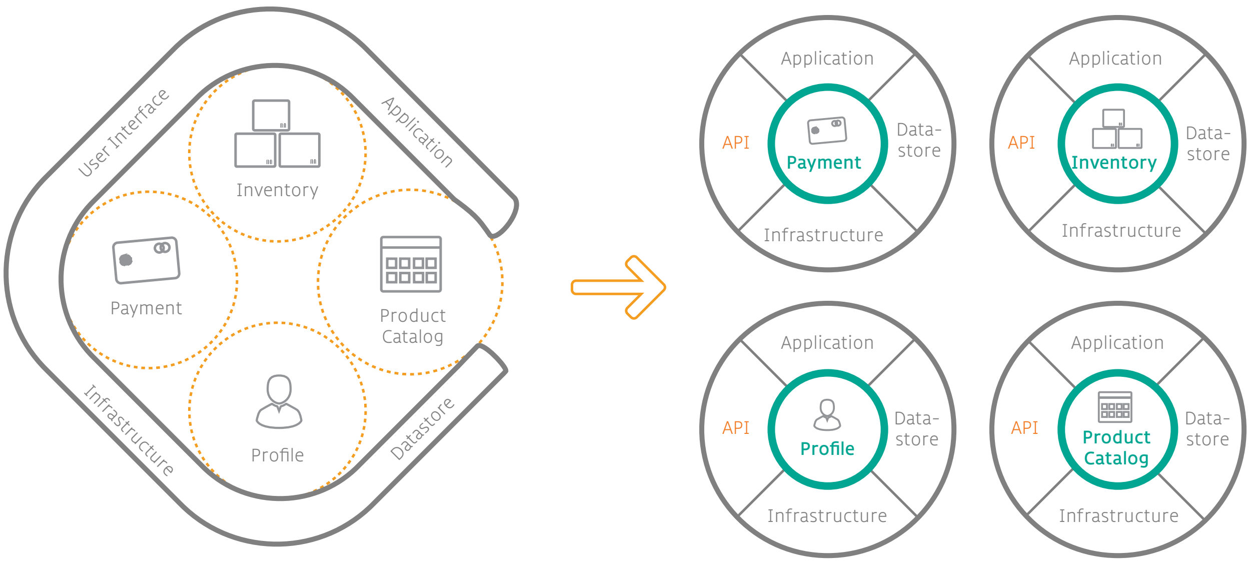

Diagram 1: Each microservice has its own UI, application and datastore layers.

“ *[...] the microservice architectural 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.”* [5]

6 advantages of a Microservices-based approach

In practice, following a Microservice-based approach has a number of advantages in comparison to the shortcoming of monoliths listed above.

1. Reduced software complexity
By definition, the scope of a single Microservice’s functionality is limited, so that maintaining and updating becomes much easier. You only have to care about messages from other Microservices that you subscribe to (inputs) and your API that can be called (outputs). At *Gilt Groupe*, for example, the first Microservice that was put in production was designed to only output a small marketing message on a per-user basis.

2. Full responsibility
According to Werner Vogels’ paradigm[6] “You build it, you run it, you own it” developers can take full responsibility for “their” Microservice: “Smaller codebases help developers focus and have a higher empathic relationship with the users of their product, leading to better motivation and more clarity in their work.” [7]

3. Increased agility:
Building a Microservice requires a cross-functional development team, which works on their project independently. Synchronization effort between teams is reduced and features can be deployed significantly faster. The team for product service, for example, works on its own and makes adjustments and deployments as it sees fit - independently of what the payment team is currently working on. As a result, it also becomes easier to test individual services and establish a continuous delivery workflow.

4. Increased resilience
By definition, cloud services are designed for failure. If a business application is made of an array of Microservices in the background, there is no single point of failure. If one service no longer responds, this does not automatically break the whole application. You can still continue streaming movies on Netflix even if the search is down.

5. More effective scaling
Microservices are small, fully functional on their own, have their own application layers and work independently. As such, it is easier to scale them vertically and increase the

overall performance of the whole business application. If, for example, an application requires a lot of calls to an inventory service, this service can be scaled individually - without wasting resources on scaling other parts of the application that receive much less traffic.

6. Specialization
Each Microservice uses a technology stack best suited for the task at hand and preferred by the team using it. For services requiring extensive background calculation, developers might choose to use Java, whereas others might rely on more lightweight technologies such as PHP or Ruby.

“*Microservices don’t reduce the complexity of a system, but they do make complexity more explicit and visible, which facilitates bounding and containing complexity in the right ways. [...] For example, with microservices, the thousand rules might be packaged into 100 modules instead of five, which means that complexity moves from inside the modules to the connections between them and to managing the larger number of them.”* [8]

Microservices are no silver bullet

As intriguing as the advantages of a Microservice-based approach sound, it should be noted that it is neither suited for every business context nor does it result in development and maintenance complexity magically vanishing.

There is no clear pro or contra distinction between monoliths and Microservices; opting for one of the two always involves a trade-off.

When it comes to development of complex e-commerce projects, Microservices offer a clear advantage over monoliths: Reduced functionality leads to less software complexity which in turn makes developing new features easier. However, when it comes to deployment and operations, the monolith can be deployed at once. With Microservices, you need a common container orchestration system, multiple tiers of load balancing, and service discovery to make sure that the services are deployed correctly and to monitor whether they work together as planned.

There are also other issues with Microservices that need to be considered, such as latency in the case of weak connections or the *eventual consistency* that comes with an architecture primarily aimed at high availability.

Two approaches in a nutshell

Monolithic applications and Microservices are structured completely different, both regarding technological as well as organizational aspects. The following table sums up the most important points:

Software monolith	Microservices
A single application	Array of many small services with limited functionality
Entire application needs to be deployed	Microservices can be deployed separately
One datastore for entire application	Each Microservice has its own datastore.
Communication within application	Remote calls, usually REST calls via HTTP
Separation between developers and ops	Cooperation of developers and ops to maintain stable operations
State lies in external application at runtime	States are stored centrally, individual instances are stateless

4 Who benefits from Microservices?

A number of fast growing, fast moving corporations such as *Netflix*[9] , *Uber* and *Google* already rely on a Microservice-based architecture. In retail, apart from Amazon - one of the first to move from their monolithic application to a service-oriented platform – there are also other merchants who apply a Microservice-oriented approach to address the challenges of their digital transformation.

REWE uses Microservices to scale effectively

The *REWE Group* is a co-operative retail group based in Germany, with total external sales of more than €51bn in 2014. Running the second largest supermarket chain in Germany, REWE invests heavily to build a strong offering in online food retail. The company started the *REWE digital* branch in 2013 to hire the necessary talent and build the required technical infrastructure to meet the ambitious goals of the group[10] .

The challenge for REWE digital was to find both an organizational structure and a technical approach which would allow them to grow, scale effectively and support the various strategic initiatives. According to their CEO J.J. van Oosten[11] , this would only be possible having decentralized, self-organized teams that can make their own decisions and work on tasks in parallel. As a result, new features can be put online almost daily, improving the customer experience one deployment at a time.

Regarding matching technology to the team structure, Dr. Robert Zores, CTO of REWE digital, thinks that “a microservice component should only be as big as a small development team can build and test in a single month. And if the service or API needs to be enhanced or extended, build a new one.”[12]

Gilt Groupe relies on Microservices for their unique business model

The *Gilt Groupe* offers a flash-sales e-commerce site for luxury brands and lifestyle goods and was founded in 2007. It is based in New York, aims at selling highly discounted merchandise to its members and has generated revenue of $650m in 2014. Each day at noon, *Gilt* sends out a blast of emails and other notifications to its more than 6 million members, informing them of discounted products with only limited stock. As a result, they see a massive traffic spike following these send-outs, resulting in a rush of visitors to their site. From a technical point of view, one of the major goals for *Gilt* was to find a software or an architecture that would be able to handle the shift of this traffic dynamic.

After relying on software monoliths based on Ruby-on-Rails and Java for the first years, *Gilt* decided to move towards a Microservice-based architecture in 2015. Today, they are running more than 250 different services that create the individual pages for their different sites and form the basis for their inventory and logistics. Regarding the benefits of this approach, the Gilt Groupe emphasizes the following:[13]

- Lessens dependencies between teams - resulting in faster code to production
- Allows lots of initiatives to run in parallel
- Supports multiple technologies/languages/frameworks
- Enables graceful degradation of service
- Promotes ease of innovation through 'disposable code' - it is easy to fail and move on

Zalando builds a fashion platform on top of Microservices

Founded in Germany in 2008, fashion retailer *Zalando* is now active in 15 countries, has generated revenues of more than 2.2bn € in 2014 and sees more than 130 million page visits per month. Recently, the company has started an initiative dubbed "radical agility"[14] , changing their organizational structure to form small, autonomous teams and create software using cloud and Microservice principles. One of the first projects following the radical agility approach was to exchange their fashion store frontend:

Our team — seven engineers and a UX/UI designer — decided to replace "Jimmy", our monolithic shop application, with microservices built by multiple autonomous teams. Under Jimmy, all of the shop teams shared the same code base and lacked true ownership or the freedom to make decisions, plus couldn't move quickly because of Jimmy's slow startup time and our overly complicated deployment processes. Radical Agility promotes microservices to get around monolith-generated problems. [15]

This strategy allowed the *Zalando* development teams to bring new features into production without depending on others. Although there still needs to be communication between the teams in order to create a seamless frontend design and a homogenous user experience, relying on Microservices has increased the level of innovation at *Zalando*. Also, because of the polyglot nature of Microservices, which means that they can be built with any given technology, the fashion retailer is able to attract new talent.

5 What steps are required to introduce a Microservices architecture?

As convincing as the case for a Microservices architecture might be in theory – putting the necessary steps into practice is an entirely different story. This is mainly because in order for this new approach to be successful in the long run, organizations need to change structurally. New teams have to be formed so they can work as decentralized, autonomous units building, owning, and maintaining their own Microservices.

Step 1: Define strategic goals

In a first step, get some clarity regarding your strategic goals and how these benefit from this new approach. Broadly speaking, there are three objectives you could aim for:

1. Gain agility
For customer-centric initiatives, it becomes increasingly important to get new features out quickly, so customers can benefit directly and remain loyal. Instead of showing something fresh every quarter, aim at deploying updates at least daily.

2. Scale with growing traffic and bigger team sizes
In a similar fashion, you need to make sure to prepare your infrastructure for the anticipated growth. On one hand, scaling refers to the software being resilient enough to handle large amounts of traffic – on the other it describes the architecture's ability to allow for growing teams and parallelized tasks.

3. Mitigate vendor lock-in
In order to manage the risks of relying solely on one technology for business-critical processes, you might consider developing your own Microservices for these areas and thus regain technical independence from any third-party vendor.

Step 2: Define business domains according to the customer journey

Next, analyze your current software ecosystem from the perspective of the customer journey it helps to provide. Usually, there is an evolved, highly individualized monolithic commerce application in which very different business functionalities are encapsulated, as the diagram (2) shows.

Try to define the boundaries between the various stages of the customer journey and which services are needed for the respective customer experience. The process of discovering a product, for instance, is completely separate from the checkout and fulfillment logic that happen later on in the course of the journey. Then, within the bounded context of product discovery, find smaller functional units that can work independently from one another, such as search or product details.

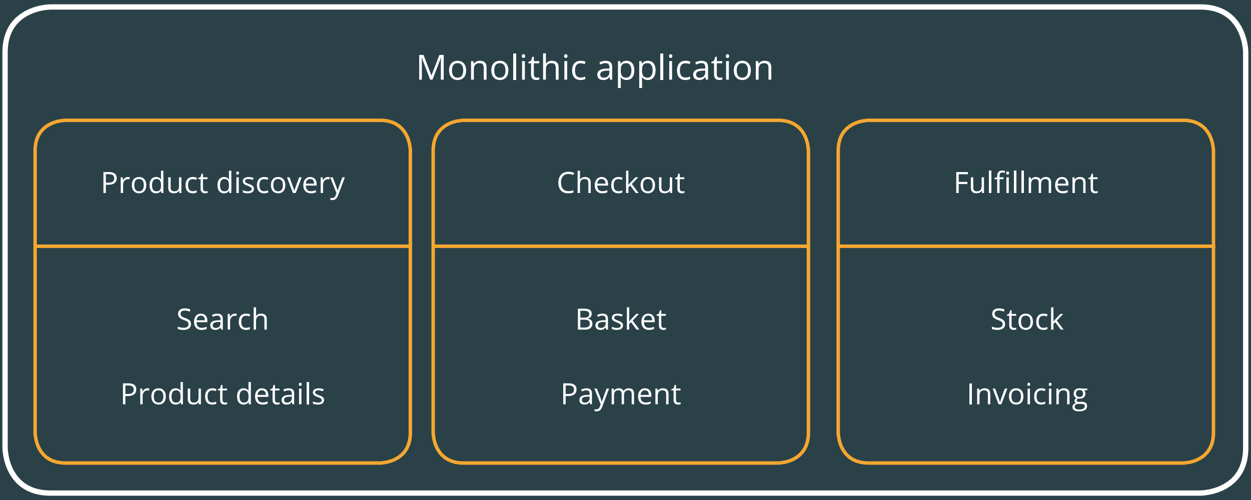

Diagram 2: Setup of a monolithic application

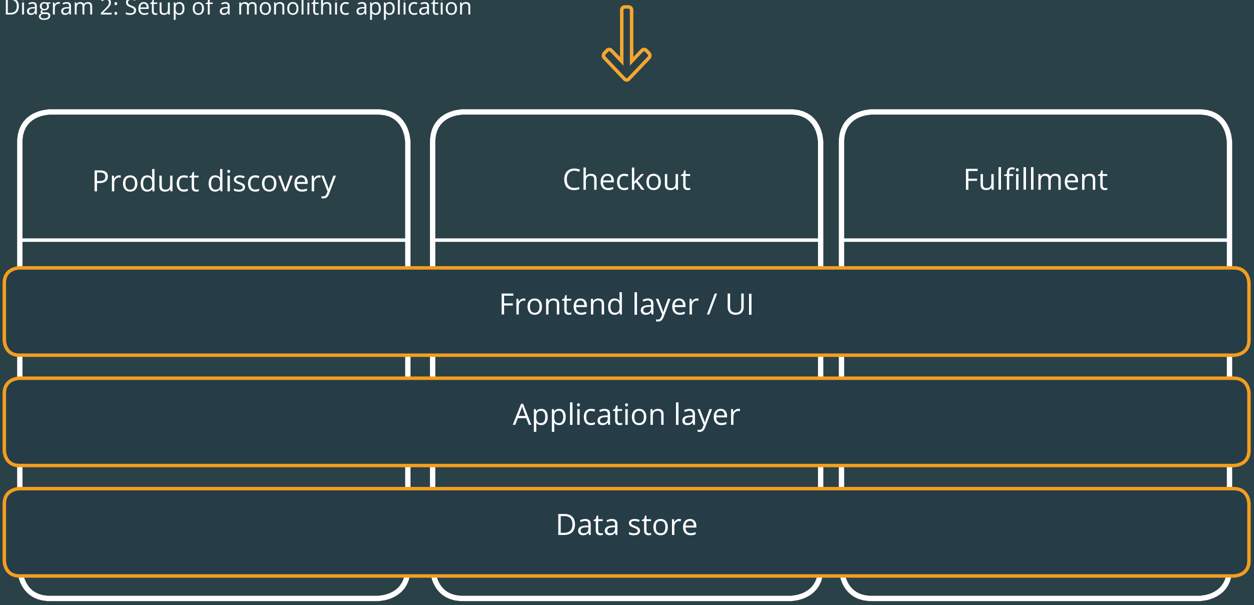

Diagram 3: Classic, horizontal technology tiers.

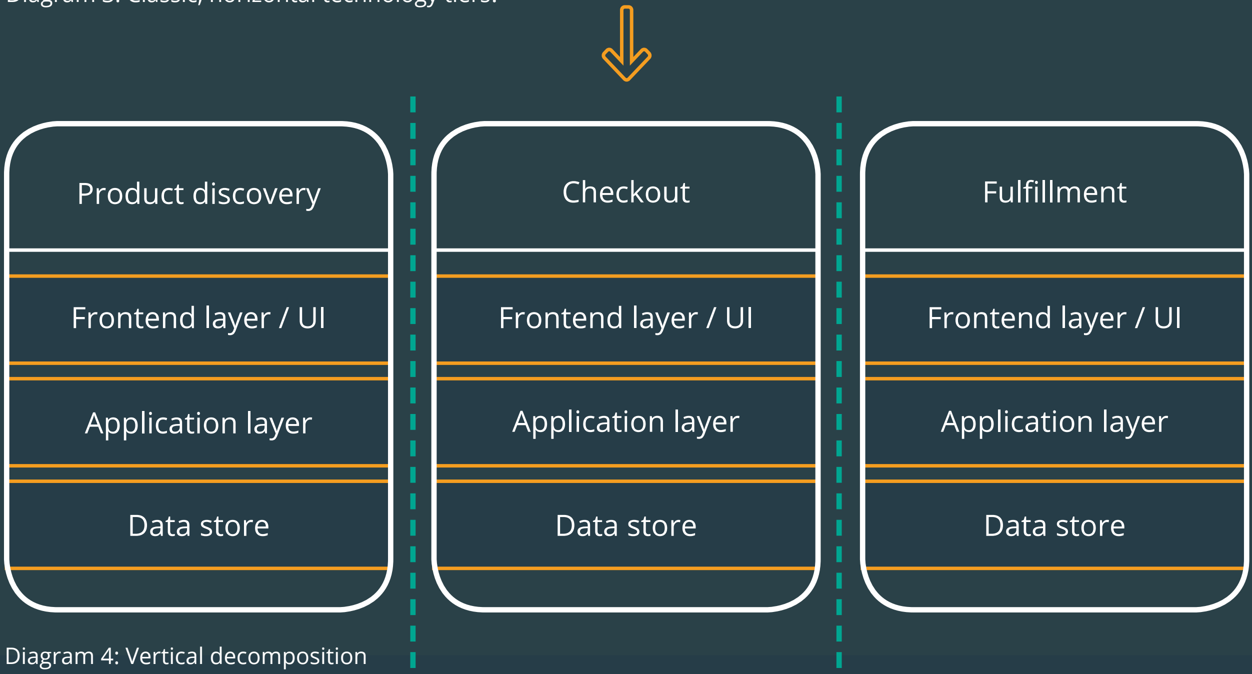

Diagram 4: Vertical decomposition

Step 3: Review current systems and match domains to goals

In this step, look at each domain individually and determine how well the respective parts of the application match the business objectives. Use a simple, traffic-light system to specify if the objective is already fulfilled (green), if some work needs to be done (yellow) or if the objectives cannot be met at all (red). For instance, the organization could focus on the shopping cart, think about what should be possible with it in the future (e.g. multiple locales, multiple channels, mobile-friendly, etc.) and then assess what the current status looks like. This simple method also helps prioritize the necessary effort. And as a side-effect, using this threefold classification makes it easy for non-technical staff to better judge the challenges at stake and follow the progress.

Step 4: Form new teams around domains

A fundamental part of a successful Microservices infrastructure is to align teams to the services and make them autonomous. Ideally, each team owns one (or more) Microservices and is completely responsible for what they are and how they perform.

Traditionally, teams are organized around horizontal technology tiers, such as frontend or data storage (see diagram 3).

Here, developers do their work across all business domains and might be working on search functionality in one week, and on checkout features in the next.

In contrast, Microservices are planned and built as self-contained, isolated units which do not share either code-base or data store. As a result, a given application will be cut into vertical parts, resulting in fully functional parts that contain a specific part of the overall business logic (see diagram 4).

Organize your teams according to these vertical parts, making sure that each team has a broad skillset – involving specialists for UI, backend development as well as operations – so that they can work autonomously.

Step 5: Soft migration

The decentralized nature of Microservices is a major advantage for their introduction into a company's existing IT landscape. They can take over processes formerly assumed by the monolith – one part at a time. Rather than making the transition with one fell swoop, teams can gradually introduce more Microservices to the picture.

A good starting point for an e-commerce business is to think about an alternative way to store and deliver their catalog data. Implement a product data Microservice parallel to the existing legacy application and update its data regularly. As a result, this new product service can serve data to other clients, such as internal search or price comparison sites, and successively take over responsibility from the original application – to a point, where products are not handled by legacy software anymore.

Summary

Businesses that have a clear customer-centric strategy require a flexible and scalable software infrastructure in order to be successful in a disruptive marketplace. Introducing a Microservice methodology - as complex and startlingly it may seem at first glance - is the next logical step.

Luckily, from a technical point of view, you do not have to start from scratch. As an innovative omnichannel e-commerce platform, *commercetools* offers all the necessary tools and capabilities and exposes them through a range of very flexible APIs - making it the perfect basis for a Microservice approach. Also, it is a resilient, cloud-based solution which scales automatically depending on your organization’s requirements.

However, technology is only one side of the equation. In order for this strategy to be successful, it requires a long-term effort in terms of the organizational structure as well. By breaking down business functions into independent apps, you allow new features to get to market very quickly because you remove dependencies between teams. Only by enabling them to create, maintain and run their own Microservices do you have the opportunity to scale your business and stay at the top of the food chain in the next decades.

Endnotes

1. Steve Yegge, Stevey's Google Platforms Rant, https://plus.google.com/+RipRowan/posts/eVeouesvaVX
2. Wikipedia "Conway's Law", https://en.wikipedia.org/wiki/Conway%27s_law
3. Abel Avram, The Benefits of Microservices, March 2015, http://www.infoq.com/news/2015/03/benefits-microservices
4. Kelly Goetsch, Microservices + Oracle | A Bright Future, January 2016, http://www.slideshare.net/KellyGoetsch/microservices-oracle-a-bright-future
5. Martin Fowler and James Lewis, Microservices - a definition of this new architectural term, March 2014, http://martinfowler.com/articles/microservices.html
6. acmqueue, A Conversation with Werner Vogels, June 2006, https://queue.acm.org/detail.cfm?id=1142065
7. Abel Avram, The Benefits of Microservices, March 2015, http://www.infoq.com/news/2015/03/benefits-microservices
8. Randy Heffner, Microservices Have An Important Role In The Future Of Solution Architecture, Forrester Research, Inc., https://www.forrester.com/report/The+Future+Of+Solution+Architecture/-/E-RES123031
9. Adrian Cockcroft, Migrating to Microservices, July 2014, http://www.infoq.com/presentations/migration-cloud-native
10. Mike Dawson, Rewe arms for Amazon food delivery challenge, March 2014, http://www.german-retail-blog.com/topic/past-blogs/Rewe-arms-for-Amazon-274
11. J.J. van Oosten, Transforming a Retail Power House in the Digital Age, November 2014, https://www.youtube.com/watch?v=0Uq5vzQ6dB4
12. Ted Schadler, Forrester Blogs, Some Thoughts On Shippable Software And Microservices, October 2015, http://blogs.forrester.com/ted_schadler/15-10-01-some_thoughts_on_shippable_software_and_microservices
13. Daniel Bryant, Scaling Microservices at Gilt with Scala, Docker and AWS, April 2015, http://www.infoq.com/news/2015/04/scaling-microservices-gilt
14. Lauri Apple, Radical Agility with Autonomous Teams and Microservices in the Cloud, June 2015, https://tech.zalando.com/blog/radical-agility-with-autonomous-teams-and-microservices-in-the-cloud/
15. Dan Persa, From Jimmy to Microservices: Rebuilding Zalando's Fashion Store, October 2015, https://tech.zalando.com/blog/from-jimmy-to-microservices-rebuilding-zalandos-fashion-store/

commercetools

About commercetools

commercetools revolutionizes the enterprise commerce platform market by combining the flexibility of an on-premise solution with the speed of SaaS. Following a unique API-first approach that radically reduces complexity, the platform enables large businesses to deliver engaging shopping experiences across all channels and drive innovation.

Contact us

Europe
commercetools GmbH
Adams-Lehmann-Str. 44
80797 Munich
Germany
Phone: +49 (89) 9982996-0
Email: marketing@commercetools.de

US
commercetools Inc.
American Tobacco Campus | Reed Building
318 Blackwell St. Suite 240
Durham, NC 27701, USA
Phone: +1 212-220-3809
Email: mail@commercetools.com

CT-WP-MiSe-EN_V004-11.17