skava®

Evolving with your customers

Why shift from a monolithic architecture to microservices

JON FELDMAN, Senior Director, Product Marketing
HASSINA OBAIDY, Senior Manager, Content Marketing

skava

Table of Contents

skava

Table of Contents

skava

Introduction

COMMERCE IS EVOLVING

You want to be disruptive. You want to deliver one-of-a-kind user experiences to your customers. Their expectations are rising. According to a CEI survey, 86% of consumers are willing to pay more for a better customer experience. You see once unique features such as personalization, voice commerce and shopping across channels becoming the norm, with a new set of trends on the horizon.

Is your commerce platform up to the challenge? Does it feel like integrating modern concepts into your application is more difficult than it should be? Is your legacy platform holding you back with an archaic architecture?

You aren't alone. And there is a modern alternative.

Microservices have emerged as the next step in modern commerce architectures, targeting many of the shortcomings of legacy monolithic platforms. By breaking the functions of your commerce platform down into individually contained applications, or microservices, you gain radical agility, flexibility and performance. Best of all, you can switch to a microservices based commerce platform gradually, deprecating parts of your monolithic platform as your microservices come online. You can keep the plane flying while you replace the engines.

In this e-book, we will explore why microservices are becoming key in today's rapidly evolving digital commerce era, address monolithic limitations, and what you need to do to meet customer expectations.

SECTION 1

Monolithic Platforms: Old Enough to Drive

Monolithic systems were designed in the 1990s when architectural best practices suggested a single application to handle the user interface, back end commerce rules, catalog and integrations all interwoven as a single system. In this “tightly coupled” architecture, code is not segregated by function and runs in the same process as every other part of the code.

This architecture was very effective in the early days of ecommerce when having a store was considered table stakes, and customer expectations were low. As customer expectations evolved, this architecture struggled to stretch beyond its design limitations, introducing friction into development.

SECTION 1.1

Monumental Challenges for Businesses and IT

On the surface, a monolithic framework might appear to have everything you need to run your systems. You can develop, test, deploy, and scale a monolith by running numerous copies behind a load balancer. However, as the application grows, the code base also grows. In the long run, your IT and business teams will face exponential challenges that will be difficult to overcome and quickly recover from.

In a monolithic platform, your abilities to innovate quickly and deploy unique customer experiences are limited. You typically deploy all feature sets in one giant slug of code, once or twice a month.

A single code change to a monolith requires the entire system to be restarted or, worse, break. As the code base grows, it is not unheard of for a change in one area to have unanticipated changes in seemingly unrelated areas. This inefficiency really shows itself when you try to implement a new customer experience.

For example, let's say a hotel company wants to elevate its guest experience by installing devices, such as iPads, in every hotel room. Hotel guests will be able to order any kind of room service via the iPads or the hotel app on their personal devices. It would be sensible to leverage your existing commerce system to settle these transactions—you have already built credit card processing, have the user's contact information in the commerce system, and can easily integrate into the hotel property system.

But there is a catch. To support your inroom iPad application, you need to interact with your legacy system through APIs. Monolithic systems approach API access in different ways, but ultimately their APIs were written and developed after the core functionality of the commerce system, and may not support all functions of the underlying platform. Rather than handing your robust and documented API contract to your iOS developer, instead you must carefully examine the code base and make sure the new app correctly interacts with the rest of the code. Likely, you will have to write new code that further entangles the existing code base, making it costly, time consuming, and resource intensive. Ultimately, this increases technical debt, and the risk of downtime and performance outages across the entire platform, including existing live features, such as online hotel reservations and check-in/check-out services.

Over time, this "all-in-one" platform or "cookie-cutter" approach becomes outdated and rigid in terms of addressing any business changes. It also reduces the IT team's flexibility to aid the development process and quickly meet customer needs.

SECTION 1.2

Limited Flexibility Leads to Technical Debt

In a monolithic framework, flexibility is very limited. Any update to the system requires extensive regression testing to avoid unanticipated side effects of your code changes. Even though monolithic architecture attempts to segregate components, ultimately everything runs using the same memory, on the same CPU, and shares access with other components of the platform. Implementing new features and capabilities, modifying the interface, or performing a simple update to an application impacts the entire monolith.

To update information on a product page, the developer must access the same code base he or she would access to add a new customer service feature or change the functionality of a promotional carousel on the home page. As a result, the system experiences downtime and performance outages. When the IT team experiences these inefficiencies, their efforts to market products to customers are met with considerable friction, which slows down time to market.

Worse, the difficulty of continually extending these platforms on a tight timeframe encourages developers to cut corners, leading to only good enough solutions and extensive QA cycles. As a result, technical debt increases and agility decreases. It is not uncommon for monolithic systems to require as much QA time as development time. The size of the application makes it brittle, and it is an uphill battle to get funding to refactor operational but inelegant code.

Let's say you're a retail company that wants to test a "name your price" feature. Both your business and marketing teams have the concept nailed down, but it has never been tried with the existing technology. Unfortunately, your monolithic platform does not support name your price out of the box and will require new code, impacting existing checkout flow. And the catalog. And the pricing engine. And the existing promotions will need to be regressed. Introducing new third-party integrations and services to the platform is difficult and testing new features becomes expensive and inefficient.

> "Even if you know what you want to build, there's this proliferation of devices and user experiences. Building and maintaining those on a monolith is challenging. Also, you won't know how it is going to work, so you want to be able to experiment..."
>
> —Dave Barrowman, Head & VP of Innovation, Skava

Over time, the struggle to implement new features and technologies in a monolithic platform places you behind your biggest competitors and even your customers.

SECTION 2

Think Micro: What are Microservices?

Microservices are standalone, independently deployable business applications. Each has a dedicated database, well-defined APIs, and an admin console that runs in its own process. According to Accenture, "With microservices, you can create an architecture ecosystem that allows you to change all the time."

Microservices enable businesses to build, test, and deploy faster and are easier to build and maintain as they're broken into smaller, composable pieces that work well together.

Microservices can be built using any programming language, framework, and front end technology, with limited external dependencies, affecting other parts of the application. Tweaks or added features and capabilities can be easily re-deployed and maintained without compromising other parts of the platform.

"The goal of microservices architecture, as opposed to monolithic architecture, is to break up things into 'logical' forms as they fit together well," said David Levine, solution architect at Skava. "The key is to be independently evolved, in terms of added features, and independently deployed, in terms of running them. It's a little bit like Lego blocks and getting that wonderful Millennium Falcon that you can play with immediately and put it all together, and all the pieces work!"

Compared to monolithic systems, microservices provide key benefits that allow any business to scale, be more agile, and deploy faster. In this section, we'll dive into the key benefits of microservices and how they have transformed modern commerce.

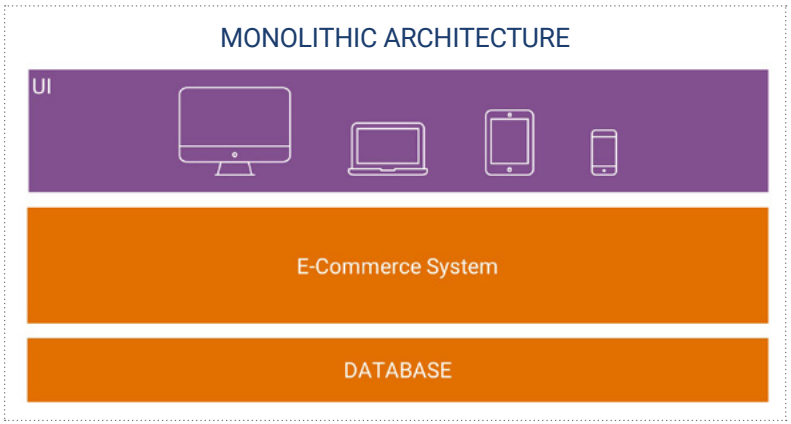
SECTION 2.1

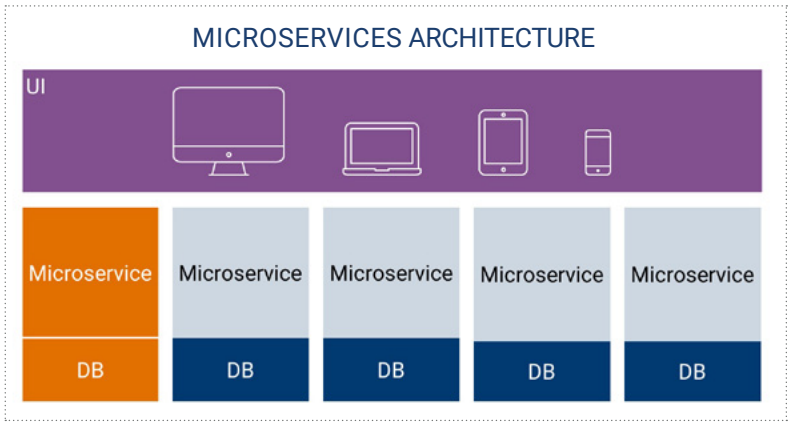
Independent and Highly Scalable

Thanks to its scalability, a microservices architecture is ideal when you constantly make updates and changes with the latest technologies, customer behavior trends, and new ideas. Microservices are decoupled—developers can make changes to a microservice without altering the entire software or breaking the platform. If a change is made to only one application, that specific application can be redeployed independently.

Suppose that, due to slow execution, a single microservice (such as promotions) becomes a bottleneck during the flow of building an application. That microservice can run on powerful hardware to increase performance, or it can run multiple instances of the microservice on different machines to process data elements in parallel. A microservice can quickly process and deploy without depending on or affecting other pieces within the system. If your promotion engine is slow, you can target it with software optimization or hardware deployment without affecting other parts of your application, and without spending money to needlessly add hardware to performant parts of the system.

A crucial part in delivering a quality experience is having a robust platform that can scale up and down as necessary without failing. Since microservices are independent, they're easier to manage compared to applications in a monolith. Unlike a monolith where a surge in one module can put the entire system at risk, things are quite different with microservices. As each microservice can scale independently, a surge to one service is mostly contained within that service and system resources can be rerouted to further support it. From a business perspective, it allows





SECTION 2.1

Independent and Highly Scalable (cont.)

brands to offer the same capacity load much longer, more efficiently, and more cost effectively, avoiding technical debt. Better, it is straightforward to build performance defense in depth. Each microservice can have rules on what to do if it can't find all its dependent services. A failure in one microservice doesn't stop demand and can be handled gracefully.

Microservices architecture deconstructs the memory and CPU need of each service, allowing each module of your application to scale up and down individually. The handling of traffic is done just as well for 200 or 10 million concurrent requests. Demand balancing helps accelerate continuous development without sacrificing reliability or data integrity.

The following microservices characteristics define the key differentiators from a monolithic architecture:

- **Data independence** Changes to microservices don't impact other areas of an application or architecture because they store data independently of one another. By comparison, monolithic applications store and call upon data in the same format, so a change anywhere in the monolithic architecture impacts all other areas.
- **Independent scaling** Organizations can horizontally scale each microservice. For example, if the work load is increasing in only one area (such as more people checking out online) the company can scale just the microservice responsible for powering the checkout process. In a monolithic, tightly interdependent architecture, such a change would likely trigger changes to surrounding processes and/or require that the company's commerce site be pulled offline during the upgrade.
- **Independent evolution** Businesses can frequently and continuously release microservices updates without disrupting ongoing operations, directly in line with the Continuous Integration/Continuous Delivery (CI/CD) concept. In fact, companies can embrace designed-focused and accelerated innovation since development teams can be built around specific microservices they enhance independently. As a result, they can evolve capabilities much faster than is possible in a monolithic architecture.

SECTION 2.2

Amazon—from One Giant Monolith to Individual Microservices

In 2001, Amazon's website was a large, multi-tiered architectural monolith; within those tiers lie many tightly coupled components. Over time, as the development team's size and code base grew, projects matured, and the architecture became more complex, the software development life cycle began to slow down.

The developers of Amazon's giant monolithic website each handled a tiny piece of the application. Every time they wanted to add a new feature, fix a bug, or even do a quick fix to push out to customers, they had to coordinate with every other developer to process changes simultaneously, rather than on their own schedules. This frustrated software engineers, slowing the development life cycle, innovation, and attempts to launch new experiences to customers. Finally, Amazon made architectural and organizational changes, providing each team full ownership of one or a few microservices.

"We went through the code and pulled out functional units that served a single purpose and wrapped those with a web service interface," said Amazon Web Service's senior manager of product management, Rob Brigham. "We then established a rule, that from now on, they can only talk to each other through their web service APIs." From an organizational-level perspective, Amazon broke down their teams to 6–8 developers per team —what they call two-pizza teams. "We originally wanted teams so small that we could feed them with just two pizzas," said Brigham.

> "We originally wanted teams so small that we could feed them with just two pizzas"
>
> —Rob Brigham, Senior Manager of Product Management, Amazon Web Services

Amazon's development teams can now quickly make changes or add new features to their microservices. According to Brigham's presentation during an AWS conference in 2015, the AWS team performs 50 million code releases per year. In addition, Amazon dramatically improved its front end development life cycle. The solution: they built a microservices-based architecture.

SECTION 2.3

Agile Development with Ownership and Autonomy

Developing for microservices in an agile methodology means you don't need armies of developers to build prototypes and applications. Teams are self contained, formed around specific business functions and contain everything the developers needed to perform their function. The inventory team would contain developers for the inventory system of record, the ecommerce system, QA and a product manager. This unit can build and deploy a microservice without relying on other teams, allowing autonomy and agility in code production. Correspondingly, each team owns a business function and will only make changes to that particular microservice when necessary.

This organizational structure is designed to remove the need to simultaneously align the schedules of your UX, UI, Back End, OMS, Tax and WMS teams to introduce a new product.

Let's say you're a Product Manager for a credit card company and you want to introduce NLP (natural language processing) into your existing platform to build a chatbot messenger. This chatbot will allow customers to check account summaries, purchase from credit card partners, check reward points, and ask basic questions about policies. Adding NLP into your application means touching many parts of your application, most of which were created without any thought towards this type of extensibility. It becomes a significant amount of code in flight, increasing risk, and expanding the QA effort.

In contrast, under a microservices framework, developers can work atomically to build a new microservice using any programming framework without impacting other parts of the system. This will not affect the other microservices built around the platform so long as your new NLP microservice adheres to the existing API contracts of your other microservices. And your chatbot messenger experience will be up and running in no time.

> "You don't need an army of developers with deep and comprehensive understanding of your monolith. Rather, you can have small teams of specialists."
>
> —Dave Barrowman, Head & VP of Innovation, Skava

Microservices are not just about rapid time to market for new tools and technologies. Each microservice typically performs one business function, responsible for a specific task. This allows small, tightly knit development teams to become experts in that specific business function and not the entire technology infrastructure. Fundamentally, structuring teams in smaller groups results in scalability. The goal is to deliver value at speed and scale without sacrificing the brand's quality. Rather than teams shackled together with too many dependencies, smaller teams of 5–10 independent developers deliver and deploy faster to the end users.

SECTION 2.4

Better Control of the Upgrade Path

One of the key problems of a monolith platform whether cloud or on-premise is the upgrade path. An upgrade is usually at the discretion of the vendor and impacts the entire system. Integrations change, contracts are broken, and the entire platform needs QA regression. Oftentimes, features that worked well in the previous version don't seem to work well after the upgrade. In other cases, teams that grew accustomed to operating in certain ways and built robust work processes around existing features are forced to retrain as a result of an upgrade.

With microservices, this is radically mitigated as each microservice has its own upgrade and can be upgraded one at a time. Upgrades are driven as a result on specific innovations that were implemented to a microservice, keeping the rest of the system uninterrupted. Innovations are more focused and faster to implement and brands can choose whether they would like to keep the current version or upgrade.

SECTION 2.5

The Freedom to Innovate and Experiment

Modern commerce is evolving. Brands want to evolve with it and continue to be disruptive. As discussed earlier, this means more flexibility with room for innovation to experiment with new ideas and quickly translate them to enticing experiences.

The flexibility of microservices gives IT and business teams the freedom to choose the latest front end and back end technologies best suited for their particular functionalities. Since microservices are encapsulated, a development team can use any front end technology, distinct from other development teams.

For example, if the loyalty microservice team uses AngularJS and the promotions team uses JavaScript, neither will affect the other's application development - the components interact via an agreed API contract and do not share code. Furthermore, microservices' scalability supports a range of platforms and devices that brands can adopt: mobile, desktop, wearables, Internet of Things, and voice-enabled technology (such as Amazon Alexa and Google Home).

You constantly develop new ideas to delight your customers and expand upon your existing system. Microservices are designed to experiment with new ideas and build prototypes quickly. The idea is to fail fast — launch new experiences, experiment, and recover immediately upon failure.

> "The dream is that if you build with microservices, you can easily maintain, experiment, and innovate on those services and systems over time"
>
> —Dave Barrowman, Head & VP of Innovation, Skava

The flexibility of a microservices-based architecture enables business and IT teams to collaborate, launch new experiences quickly, and figure out what does and doesn't work.

SECTION 3

How to Think Micro: Adopting in Increments

Many companies started with a monolithic ecommerce platform to power customer-facing experiences. When transitioning from a monolithic to a microservices-based platform, it's critical that the system continue to take demand, and not freeze your progress on the customer experience during the upgrade. The road to a new architecture is not only about the technology itself, but about identifying existing issues and the teams you're working with.

> "The key is to look and think of all the problems in small discrete buckets. Look at specific pain points you want to solve and how you can fix them with a microservice, instead of fixing the entire platform."
>
> — David Levine, Solution Architect at Skava

One of the greatest advantages for moving to microservices from a monolith is that you can do it gradually without a big bang transition. Evolve your platform by adopting microservices in increments rather than replacing the entire monolithic platform. The key difference between implementing a modern and legacy systems is that you can have both systems running concurrently. Initially your legacy application remains the system of record, as you thoughtfully introduce microservices to replace functions of the monolith.

This approach, called the Strangler Pattern, allows you to slowly chip away at your legacy platform and choose when it becomes the system of record. You can keep existing projects running, minimize disruption to your business and your users, and still end up with a state of the art platform. A true microservices-based model allows you to plug in the pieces you need and build on top of those microservices. Even then, integrations are made to tie those microservices to a monolith. The concept here is to rip out pieces that don't work for your organization and replace with independent applications you can scale quickly.

SECTION 3.1

Pathway to a New Development Methodology

Though not required, microservices allow your development team to transition to efficient continuous integration and continuous delivery (CI/CD) processes to achieve the agility that microservices promise. Microservices help accelerate the journey to CI/CD, but it also depends on the maturity of an IT organization.

If you already have a team of highly skilled developers working on your legacy application, organizing them into agile pods focused on a single business function, and therefore making them responsible for a single microservice, can dramatically speed up development. You can work with a trusted vendor or SI partner to establish the DevOps best practices and methodologies as you switch.

In *Selecting a Cloud Platform for DevOps Delivery with Microservice Architecture* (Gary Olliffe, 27 November 2017), Gartner notes "A microservice team should be accountable for the delivery of the running service, not just writing the code, deploying the code or monitoring the deployed code." A small, tight-knit team focused on one service (capabilities, delivery, and deployment) is scalable, reduces complexity, and introduces agile development processes.

SECTION 4

A Granular Approach Already in Mass Adoption

Acclaimed digital disruptors have already adopted a microservices-based architecture and are benefiting from its robust yet scalable technology. Many of these brands, such as Amazon, Netflix, Walmart, and Uber, started with monolithic platforms to power and maintain their customer experiences. Let's explore how Spotify is embracing the microservices approach.

Spotify: Keeping up with the Competition

SECTION 4.1

Spotify: Keeping up with the Competition

Spotify is part of a fast-moving market, competing with Apple, Google, Pandora, and even YouTube. The digital music service company serves over 75 million active users per month, running complex business rules in the back end. They needed an architecture that could scale to millions of users, support multiple platforms, handle complex business rules, and most importantly, react quickly in a competitive, fast-moving market.

More than 90 teams, 600 developers, and five development offices on two continents built the same product. Spotify needed to reduce these dependencies as much as possible to scale quickly on both the architectural and organizational levels.

At the GOTO software development conference in Berlin in 2015, Kevin Goldsmith, vice president of engineering at Spotify said, "If you're worried about scaling to hundreds of millions of users, you build your system in a way that you scale components independently."

Spotify organized autonomous teams from back and front end developers, quality testers, a UI designer, and a product owner. Individual team goals did not overlap with other teams. According to Goldsmith, Spotify's developers personally deploy their microservices with teams that have operational responsibility.

> "If they write crummy code, and they are the ones who have to wake up every night to deal with incidents, the code will be fixed very soon"
>
> —Kevin Goldsmith, VP of Engineering, Spotify

Spotify has built its systems around the assumption that any microservice can fail at any time. If a large number of services experience downtime concurrently, the system handles it gracefully. Spotify is reaping the benefits of a microservices-based architecture: it's easier to scale, test, deploy, and monitor, and it's less susceptible to large failures.

SECTION 5

A Microservices Architecture That Works with You

When researching microservices-based architectures, one finds that many vendors claim to have microservices but very few actually deliver more than a monolith with an API layer. Thus, when replatforming to a new solution, you may actually replatform to another monolith.

Skava Commerce was built from the ground up as a collection of modular microservices. With Skava Commerce, you don't have to replatform entirely; you can make the best of your existing commerce stack. Skava's hyper-flexible platform enables you to either add front end / back end Skava Commerce components and capabilities to your current commerce stack or build your own. Skava's microservices for ecommerce allows you to make "micro" releases all the time. Truly achieve continuous integration, continuous delivery, and continue to innovate and give customers exactly what they want, when they want it.

Free yourself and your teams to transform ideas into experiences.

SECTION 5.1

Case Study

A leading auto manufacturer had over 20 car models and about 40,000 variants. Their unique user experience was based on the fact that their products were massively configurable. However, they didn't have a product catalog to allow potential car buyers to seamlessly browse and buy. They had built a strong yet complex front end, but they had no ecommerce capabilities, such as payment and processing.

They were not looking to completely replatform and move to another solution; they only wanted to integrate ecommerce capabilities into their existing solution. With Skava's range of microservices, they integrated the cart and checkout microservice into their existing "configure-and-price" front end for a full end-to-end experience. In addition to cart and checkout, they leveraged Skava's payment processing and order management to help fulfill online orders and car reservations.

Since microservices are independent applications, the car manufacturer had the flexibility to select and pay only for particular components of Skava Commerce without investing in the entire system. They were able to keep their existing technology stack and simply implement Skava's ecommerce microservices alongside it.

SECTION 6

Let's Recap

Monolithic platforms are becoming outdated and companies that use them are at risk of not keeping up with the fast-paced world of modern commerce. Over time, monolith deployment will only get more entangled, increasing the inability to innovate and evolve.

1. LEGACY MONOLITHIC PLATFORMS PUT YOU BEHIND YOUR CUSTOMER EXPECTATIONS AND THE COMPETITION
2. A MICROSERVICES-ARCHITECTURE EMPOWERS YOU TO EXPERIMENT NEW IDEAS AND INNOVATE FASTER
3. ADOPT A MICROSERVICES APPROACH IN INCREMENTS AND DIVIDE UP YOUR TEAMS
4. A MATURE IT ORGANIZATION WITH DEV-OPS BEST PRACTICES IS THE MOST EFFICIENT LONG TERM TEAM STRUCTURE
5. DIGITAL DISRUPTORS ARE BENEFITING FROM A ROBUST MICROSERVICES-BASED ARCHITECTURE
6. SKAVA COMMERCE IS DIGITALLY NATIVE, BUILT AS A MODULAR MICROSERVICES PLATFORM

skava®

What's next?

Think like Amazon and other digital disruptors by Thinking Micro. Learn more about microservices and how it meets your customer's expectations.

SCHEDULE A DEMO

skava®

About Skava

Skava offers modern ecommerce and digital platforms to retailers and enterprise-sized companies. Skava's cloud-native, microservices-based technology enables continuous innovation in a shop-anywhere world. We help leading brands and enterprises across the world deliver engaging omnichannel, customer-centric experiences.

Headquartered in the heart of San Francisco, Skava has offices in the U.S., Europe and India. Learn more about Skava by visiting our website and blog or contacting marketing@skava.com.

skava®

microservices for ecommerce

FOR MORE INFORMATION

marketing@skava.com | 877·554·2176 | www.skava.com