Where are you on the Digital Transformation Journey?

FOR MORE INFORMATION
info@skava.com | 877·554·2176 | www.skava.com

SKAVA®

Are you on the Digital Transformation Journey?

> "With microservices, you can create an architecture ecosystem that allows you to change anytime, all the time"
>
> —Accenture

With declining store traffic, increased pressure from digital competitors like Amazon, and demand for omnichannel and omni-touchpoint customer experiences, enterprise leaders know that digital transformation is critical for survival.

93% of CXOs believe digital is disrupting their business (*Forrester*).[1] CEOs expect digital to drive 41% of revenue by 2020 (*Gartner*).[2] And 87% of companies believe digital transformation is a competitive opportunity (*Capgemini*).[3] To stay competitive, the digital enterprise needs to differentiate their experience and deliver innovation quickly.

However, ideas are easy — implementation is hard. We're in an era where legacy monolithic architecture simply cannot support the pace of change that the customer, the competition and your company demand.

Enterprise commerce in the post-monolithic era

Conceived in the late '90s, traditional monolithic enterprise ecommerce platforms were never built to support digital transformation. Customizing, extending and serving commerce to multiple touchpoints is costly and time consuming. Every change introduces risk, requires extensive regression testing and a restart of the system, making development and delivery too slow to support a rapidly evolving business.

To keep pace with the customer, digital experience trends and modern development methodologies such as DevOps, modern marketers and technologists need freedom and flexibility. For this reason, more and more modern organizations are embarking on a *digital transformation journey* — a migration towards a modular technology environment that can easily extend to new touchpoints and support quick delivery and scalability.

This digital transformation journey towards flexible architecture in the post-monolithic era typically follows a three-stage progression:

STAGE 1	STAGE 2	STAGE 3
Going Headless	Setting up a Framework	Deploying Microservices

What stage is *your* enterprise at?

1. https://go.forrester.com/blogs/14-11-07-digital_business_transformation_will_gain_critical_mass_in_2015/
2. https://www.gartner.com/smarterwithgartner/ceos-look-for-serious-growth-in-gartner-2018-ceo-survey/
3. https://www.capgemini.com/wp-content/uploads/2017/10/dti_the-digital-talent-gap_20171109.pdf

SKAVA

Stage 1: Going Headless

What are microservices?

Microservices are decoupled, stand-alone business capabilities with their own business logic, databases and set of APIs. They can be used alone or interwoven with other services without disrupting the entire application structure. For example, ecommerce components such as catalog, search, PIM, promotions, accounts, loyalty, order management, cart and checkout can all be decomposed into microservices, decoupled from one another.

Because microservices are decoupled and store their own data, they can be independently modified and deployed without affecting each other or the monolith, and without a full restart of the system. This allows you to build, test, and deploy new commerce projects faster and push continual updates without interrupting operations.

With monolithic commerce platforms, front and back ends are tightly coupled. This means every change to the front end has the potential to compromise back end code. Updates require coordination between front and back end development teams, along with the same regression testing and full system reboots as any back end change.

Monolithic front ends also have difficulty extending to mobile apps, in-store digital experiences, shoppable content and more. They can't scale independently from the monolith and they don't always integrate with modern touchpoints.

The first step towards flexible commerce is to decouple the front end from its monolithic back end. *Headless commerce* separates the front and back ends into two applications communicating over API. Advantages to this approach include:

- Front end developers can work independently and deploy on their own schedules.
- Supporting multiple touch points is easier, as code and design are seperate.
- The enterprise can use best-of-breed "heads" to build and manage experiences, including enterprise-grade content management.
- Heads can be swapped in and swapped out over time.
- Multiple heads can connect to the back end if desired (e.g. custom heads for new touchpoints and experiences).

While many monolithic platforms ship with an API, heavy customization of an enterprise-grade monolith like Oracle ATG or IBM Websphere typically renders the API unusable. Consequently, many retailers must build their own API gateway layers to support headless commerce, or replatform to a system that's natively headless.

The challenge for these retailers–even with a functional API–is the underlying back end system is still a tightly coupled monolith. The enterprise is still burdened by a heavy and complex codebase, a lack of flexibility and scalability, and slow development and delivery.

Going headless is only the first step towards modular architecture. To truly take advantage of fast and nimble front *and* back end changes, the enterprise needs a modular back end comprised of microservices.

Stage 2: Setting Up a Framework

Before an organization can start using microservices, it needs a *framework* — a stable infrastructure to house microservices and coordinate communication between them, optimize performance and support a fast development pipeline.

A typical microservices framework contains messaging and orchestration tools to manage communication and workflows between microservices. It should also have an API gateway that handles requests from various clients such as desktop and mobile devices or back office applications. The framework should also support caching, monitoring, fault-tolerance, data processing and content management.

The challenge for organizations building their own microservices environment is there are no standardized guidelines for setting up a microservices framework. This leaves ample room for problems. For example, running microservices without *service discovery* requires manual updates to client libraries. This creates dependencies that make it more difficult to evolve your platform and release code quickly. Overlooking *monitoring tools* means neglecting regular, automated health checks that would catch problems before they become failures. And performance and stability can be severely affected without *resilience* and *distributed caching* tools.

To overcome this learning curve and avoid critical omissions, adopting a pre-built, ready-to-use microservices framework from a trusted vendor can help accelerate the migration to microservices.

How DevOps and CI/CD Support Digital Transformation

When supported by the right framework and internal processes, microservices architecture allows you to make weekly, daily and even hourly updates. DevOps and CI/CD (Continuous Integration/Continuous Delivery) go hand in hand with microservices for rapid delivery.

What is DevOps?

DevOps is today considered one the most efficient software development methodologies. It combines the breakdown of organizational silos between Development and Operations teams with internal processes and automation tools to rapidly accelerate software delivery. Teams work together closely through every phase: from planning and design to development, release and support.

The CI/CD Pipeline

Continuous Integration/Continuous Delivery (CI/CD) works hand in hand with DevOps. The continuous process of writing, testing, deploying and retesting code is supported by a series of automated technologies (CI/CD pipeline) that eliminate error-prone manual work and efficiently catch bugs *before* they hit production.

Continuous Integration (CI)

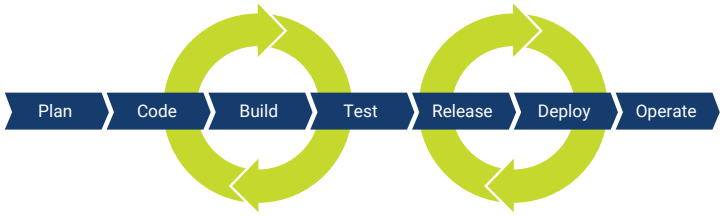
With Continuous Integration, developers push code to a common repository as it's written (which can be several times per day), rather than building features in isolation and rolling them out together at the end of a cycle (which could be months).

This means code conflicts can be caught earlier in the process, and are quicker and simpler to fix. It also reduces overall build costs and helps the organization work more efficiently, helping deliver a more stable product. CI leverages test suites and automated testing tools to offload manual work.

Continuous Delivery (CD)

Continuous Delivery further automates the software delivery process through a deployment pipeline after the CI phase. CD uses a progressive set of test suites that provide checks and balances against the build. If code fails at any stage of testing within the pipeline, a trigger alerts the DevOps team for quick identification and resolution of the problem. If code passes a test, it proceeds through the pipeline automatically, reducing manual work and shortening the development and QA cycle.

Both CI and CD work together to help teams deliver code in smaller batches, more frequently and without unhappy surprises. The CI/CD pipeline ensures that only clean code gets pushed to staging and production environments.



Continuous Deployment

Code that successfully passes through the CI/CD pipeline can be automatically pushed to the production environment with confidence. Release automation tools support automatic deployments to eliminate manual work for even faster code delivery.

DevOps and microservices

Combined with microservices architecture, CI/CD-enabled DevOps teams can be solely responsible for their own projects without having to confer with other groups. Independence allows for more wiggle room to experiment and iterate on their services, faster delivery and seamless rollbacks.

Microservices Framework: Build or Buy?

While any organization with sufficient internal IT resources can build a microservices framework from scratch, there's a lot of effort involved and, for many teams, a steep learning curve. There's also an opportunity cost to spending time on setting up the infrastructure that could be spent deploying and taking advantage of microservices that much sooner.

Leveraging a pre-built framework accelerates migration to microservices architecture. When the framework includes a microservices stack, it also allows developers to start building custom microservices on a standardized code base and documentation. Though individual microservices can be built with virtually any programming language, teams just getting started with microservices benefit from cohesion versus going rogue.

Stage 3: Deploying Microservices

The final stage of digital transformation is deploying microservices to support rapid innovation. Microservices can be added one by one around the existing monolith (also known as the **Strangler Pattern**), or with a full replatform to a microservices-based commerce solution.

The Strangler Pattern offers flexibility and control over migration, allowing the organization to prioritize which modules to convert first. For example, components that have the most immediate need to scale, or that would benefit most from frequent updates and continuous delivery.

Unlike Big Bang rip-and-replace replatforming from monolith to monolith, using the Strangler Pattern to migrate to microservices architecture allows for much faster delivery of most critical areas with minimal disruption to existing processes.

Why microservices?

When supported by the right framework and internal processes, microservices architecture allows you to make weekly, daily and even hourly updates. DevOps and CI/CD (Continuous Integration/Continuous Delivery) go hand in hand with microservices for rapid delivery.

Agility	Shorter build, test and deployment cycles help organizations deliver new features faster and more efficiently. Microservices are perfectly suited to DevOps and Continuous Integration/Continuous Deployment.
Flexibility	Loosely-coupled, modular components can be extended more easily than monolithic platforms, and can be independently swapped in and swapped out as needed.
Scalability	Instead of having to scale the entire platform, a microservice can scale independently (and on demand when hosted in the cloud).
Reliability	Bugs within any microservice impact only that microservice, not the entire system. Containerized code means issues are easier to identify and quicker to fix.
Availability	Unlike monoliths that require a full restart, microservices are updated independently with minimal downtime.

Microservices: Build or Buy?

When migrating to microservices architecture, you can build (refactor or write services from scratch), buy pre-made microservices or employ a combination of both.

Refactoring your existing monolith

Some organizations opt to extract and refactor existing monolithic code into microservices. However, this approach is not without its challenges — especially for enterprise platforms that have been heavily customized. Reusing code is not necessarily faster than a from-scratch rewrite. Undoing existing dependencies and refactoring data structures is extremely time-consuming, and in some cases impossible.

From-scratch rewrites

Depending on the age, size and complexity of your existing monolith, building new greenfield microservices may be faster than working through code dependencies and data structures. It also allows you to select the "best tools for the job" with respect to programming languages, databases, and features that best reflect today's (and tomorrow's) business requirements. However, from-scratch rewrites delay time-to-market and may require you to hire new developers skilled in these languages, rather than leverage resources familiar with the monolith's stack.

Allowing smaller, independent development teams to build their own microservices can also be problematic if the overall strategy is not overseen by an experienced software architect. The end result can be a mishmash of languages, frameworks and data structures, with a lack of consistent documentation, inconsistent (and nonexistent) business tooling, and unnecessary redundancies between microservices.

Leveraging ready-to-use microservices

With a compatible framework, ready-to-use microservices accelerate your time to market. Choose microservices that are feature-rich, extensible and built with open source technologies accessible to developers.

Because microservices are modular, you have the flexibility to choose any combination of pre-built microservices, implement them in the order and at the pace that best suits your business, and if needed, build your own bespoke microservices to achieve your ultimate commerce solution.

Skava Commerce supports the Digital Transformation Journey at any stage

> "Skava is built for the complex requirements of large enterprises. It has a wide range of deployment models and a modular architectural approach suitable for incremental delivery and platform migration."
>
> —Gartner, Magic Quadrant for Digital Commerce, 2018

No matter where you are on *your* digital transformation journey, Skava Commerce's end-to-end microservices solution has you covered.

Ranked highest of all evaluated platforms for modular, flexible implementations by Gartner's Critical Capabilities report,[4] Skava's ready-to-use **React.js Storefront, Framework and Microservices** suite help you take advantage of modular architecture faster than building yourself — without the learning curve.

React.js Storefront

Skava's flexible, decoupled and highly customizable front end is built with modern technologies such as Node.js and React.js to support mobile-first, optimized user experiences to any device and form.

Atomic and Modular Design Methodology

For fast and efficient development, an Atomic Design Pattern allows developers to reuse code from small UI elements (Atoms) like buttons or SVG icons to larger building blocks (Molecules) to rapidly construct UI components (Widgets), Templates and Pages. Descriptive elements make it easier for developers to understand each other's code and for new developers to onboard.

A Modular Design Pattern makes customization and maintenance so much easier. *Dead code elimination* and *tree shaking* identify and remove unnecessary code while maintaining the core codebase. Modular design allows you to safely integrate with third-party applications and satisfy every client's unique requirements (even as they change) without impacting main source code.

Lightning fast performance

Along with less code to load, Single Page Applications (SPA) and Progressive Web Apps (PWA) leverage modern browser enhancements and asynchronous server-side rendering to enhance UI performance. Rather than waiting for back end operations to complete, the front end retrieves only the data it needs when it needs it. Autoloading and faster "time to first content" mean higher visitor satisfaction and conversion rates.

SEO friendly

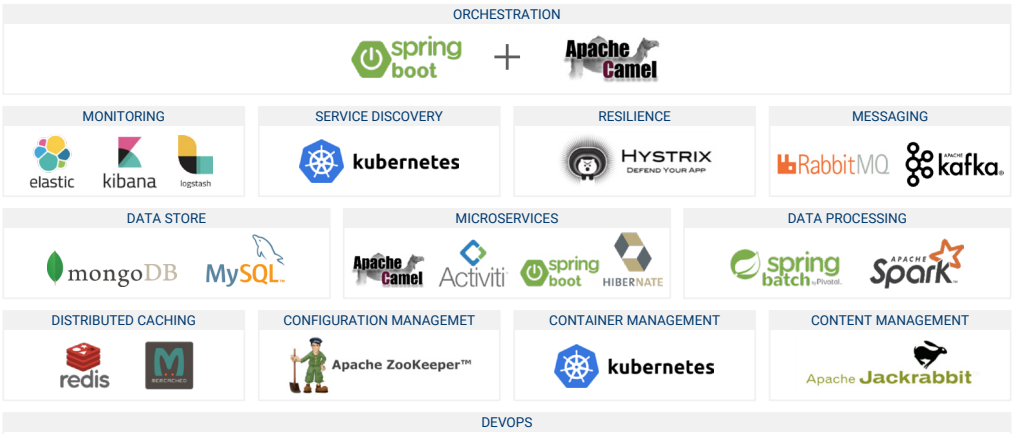
Server-side rendering also supports SEO, ensuring that Google can crawl your links and read your content.

Cutting-edge user experience

Extend your front end to any device or screen — your way. From features that mimic native app UI and functionality like push notifications for abandoned carts, new sales events and more, to innovative interactive experiences that access cameras, accelerometers, and geolocation.

Framework

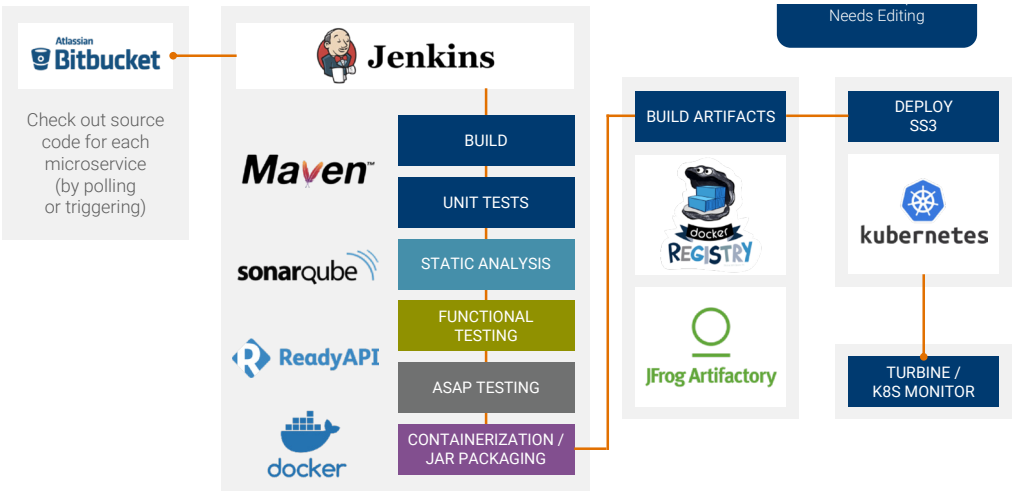
From development to DevOps, Skava Commerce's microservices Framework helps your IT team hit the ground running. Rather than build your own infrastructure, leverage Skava Commerce's ready-to-use Framework built with modern, scalable and best-of-breed open source technologies.



Skava Framework

Support DevOps and CI/CD

Our Framework provides all the automated checks and balances necessary to support your CI/CD (Continuous Integration/ Continuous Delivery) pipeline and streamline development and production. Push updates weekly, daily — even hourly.



How Skava supports the CI/CD pipeline

Have custom requirements?

Use our standardized Framework to build your own microservices, or bring third-party microservices.

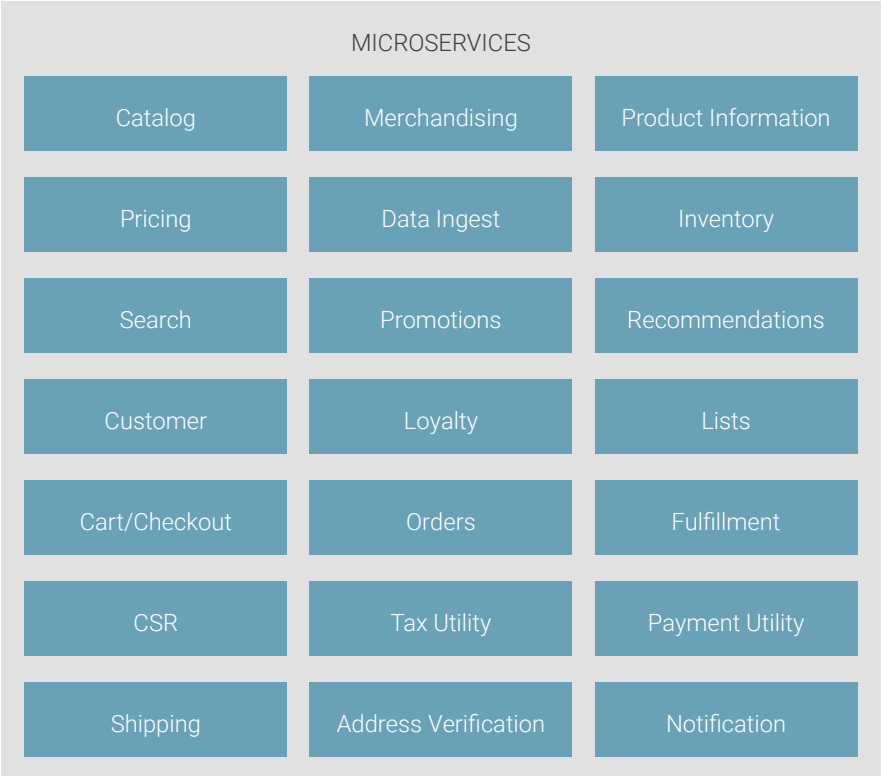
Future-proof your platform

Our modular architecture allows you to swap in and swap out technologies over time, so your environment can evolve without sacrificing best-of-breed.

Microservices

Flexible and customizable, robust and highly scalable, Skava's full suite of ready-to-use microservices helps you migrate off your monolith much faster than building your own.

Architected for agility, availability, performance and testability, Skava microservices can integrate with your existing systems or be deployed in special projects.



Skava Microservices Suite

Your platform, your way

Choose your microservices

Unlike monolithic platforms that require you to use and support every piece of the system, Skava can be deployed your way. Use the entire platform, or only what you need.

Mix and match data

Adaptable Collections lets you mix and match data between microservices. Share user data between stores, while keeping separate catalogs. Launch a microsite with a single product as easily as a new product. Scalability is never a problem, ramp up only what needs to scale.

Cloud or on-site hosting

Deploy any subset of microservices or the entire platform on-site or in the cloud — it's your choice.

Learn how Skava can help you at any stage of your Digital Transformation Journey

SCHEDULE A DEMO

About Skava

Skava's cloud-native commerce platform, Skava Commerce, uses microservices-based technology to help retailers and enterprise-sized companies quickly create personalized omnichannel experiences. Businesses can deploy our full platform or choose only the components they need to enhance their existing digital commerce stack. Skava's modular architecture enables brands to continuously innovate, accelerate time to market, and delight customers with fewer resources. Headquartered in San Francisco, the company has offices in Europe and India. To learn more, please visit Skava.com.

SKAVA