Making microservices work for your organization

Are you ready for continuous innovation and business-driven development?

Microservices can provide a huge return for your innovation efforts and yield desirable business outcomes rapidly. You can get your microservices initiative going in the right direction by making sound decisions and taking one well-planned step at a time, starting with the needed infrastructures and processes.

This ebook helps you understand the microservices opportunity and offers some practical considerations as you forge ahead:

WHY ARE MICROSERVICES A THING?

So long, monolith

Microservices can help you get software functionality to internal and external customers faster, generating revenue and improving productivity. By adopting a microservices approach, an organization may find it easier to respond to fast-changing business needs and deliver the best possible customer and user experiences.

Companies come to microservices from different starting points. Some see the need to become more flexible and responsive in aligning software with business requirements. Others look for an efficient way to take advantage of the cloud to grow and manage their technology infrastructure.

Many organizations adopting microservices decide that monolithic, all-in-one software environments no longer fit the way they work. Over time, these legacy systems can become difficult and expensive to manage. They sometimes provide functionality hardly anybody uses. It might take staggering resources to adapt them to enable the rapid innovation and digital processes that companies want to realize. Neither are all monoliths able to draw on the computing power and scalability of the cloud.

Many leading companies—Netflix, Uber, Twitter, and others—have published their microservices experience, including problems they might address differently if they would encounter them again.

To transition away from monolithic legacy software, organizations create and implement microservices to gain just the functionality they need, and evolve it in the cloud. That means conceptually and technically decoupling large application suites into distinct services, which can be created, tested, deployed, and provisioned to users one at a time.

No need to start from scratch

Like any strategic technology project or deployment, adopting microservices can be risky, expensive, and lengthy if you don't plan well. When companies underestimate the effort, they tend to omit key considerations and their microservices practices turn out to be flawed. They may spend millions on development and may not be able to deploy. Or they deliver functionality that does not meet the needs of business groups.

The technologists who create and manage software products in your microservices initiative need to move quickly and efficiently, with optimal performance and minimal cycle times. You will need to make determinations and decisions regarding business priorities and the approaches, architectures, tools, and processes that are the best fit for your specific microservices practice.

Don't let that deter you, though. For one thing, many leading companies—Netflix, Uber, Twitter, and others—have published their microservices experience, including problems they might address differently if they would encounter them again. You can also find whitepapers and articles where analysts and project managers share their microservices insights and recommendations. Full-length books by reputable practitioners offer guidance on such topics as microservices architectures and tools.

Take direction from business needs

Contrasting monolithic applications and microservices in a simplistic manner doesn't do them justice. They can both meet various companies' requirements; the rhetoric sometimes leaves aside that some so-called software monoliths may simply be bundles of distinct software services. Some technologists discuss software systems in terms of *macroservices,* which can comprise microservices. Macroservices are organically connected areas of functionality that should be managed coherently. Your choice of a standard legacy system, a software suite consisting of macroservices, or a microservices approach should depend on what's best for the company.

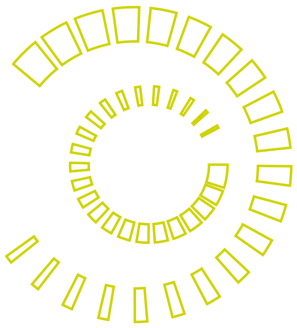
Monolithic applications may scale effectively to increasing numbers of transactions, but tend to become problematic when they cannot accommodate growing data volumes with the same ease. Development efforts may then find it hard to keep up. You probably need to create a very large team that can be difficult to coordinate and manage, whereas specialized, well-orchestrated teams in a microservices practice can be faster and more efficient. The flexibility of using different development tools and underlying technologies in monolithic environments tends to be limited, because it is often too difficult or risky to change them.

Microservices are polyglot or language-agnostic, meaning you have a choice of exactly which development language you use.

Where microservices shine

The modular approach of microservices replaces the application suite with a portfolio of components that can be built by different teams working in a concerted manner. Microservices can be prioritized by business requirements and truly begin showing their value when you perform continuous integration and continuous deployment (CI/CD) to keep your code aligned and connected, and get new capabilities to customers or users quickly.

Microservices are polyglot or language-agnostic, meaning you have a choice of which development language you use. Sometimes, working with multiple languages can make it possible to realize functionality in a more powerful or efficient manner.



Why initiatives fail

When microservice efforts stray and don't yield the business outcomes the company expects, the misdirection almost always occurs in the earliest part of the process, when business requirements are identified and translated into possible solutions. That means product planning. Some organizations hire and task a business analyst with systemic product planning that reflects business priorities and requirements.

It takes a strategic effort to decouple application functions, prioritize them, and incorporate them into a framework that can help you structure the creative process that ensues. Sometimes, companies start decoupling functionality and move toward microservices development before that framework exists. This usually results in a delay and the need for a course correction.

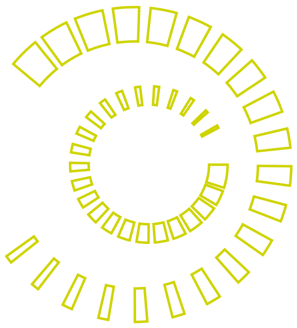
When you adopt microservices, you can test and deploy it quickly, and roll it out to users at the best time—maybe long before the competitors offer a similar product or before a competing company introduces a comparable, internal process improvement.

Savings and financial advantages

The right up-front decisions can keep the cost and risk of adopting microservices predictable and manageable. *Delaying* a microservices transition could cost you in terms of user productivity, technical obsolescence, and longer lead times to deliver functionality the business needs. On the other hand, the potential financial, competitive, and operational payoffs in transitioning to microservices can be significant:

- You can **create needed functionality faster** and purpose-design it better than in monolithic environments.
- You can **test and deploy it quickly,** and roll it out to users at the best time—maybe long before the competitors offer a similar product or before a competing company introduces a comparable, internal process improvement.
- You provide a product that **makes a real difference in user productivity.** The company benefits from more effective and satisfied internal customers and users.
- You can **handle functionality updates and modifications with great efficiency,** focusing on the microservice alone instead of having to consider possible service disruptions or delaying innovation in other areas.
- By building on a successful set of initial use cases with additional functionality, you may be able to continuously **increase employee motivation and productivity** across the entire organization.
- Over time, the **costs of owning and managing your technology infrastructure may decline.**

Some companies perform a detailed ROI or other cost-benefits analysis before they decide whether to go ahead with a microservices effort. This recommendable practice does not need to require substantial resources.



WHY IS AGILE CRITICAL FOR MICROSERVICES?

Agile maturity is key

Microservices bring a cultural shift to organizations that are organized hierarchically and with strong centralization. They share with lean thinking the emphasis on accountability, team autonomy, distributed leadership, and decentralization. You can almost entirely eliminate dependencies on the knowledge and influence of key individuals—and the bottlenecks that result when they leave or change roles—by bringing enablement and intelligence to teams of people who collaborate in a goal-driven, shared-values process. The basic enabling practices for microservices, including agile development methodology, DevOps, and continuous integration and continuous deployment (CI/CD), align with lean ideas.

You need to have a mature agile practice to benefit from microservices. In some companies, agile and waterfall approaches are mingled, but teams refer to themselves as agile. Elsewhere, when companies are already far along on the path to agile maturity, they still evolve their agile practice as they prepare for microservices. If you want to reap the full benefit of microservices, you should become fully immersed in agile and abandon waterfall or other development approaches.

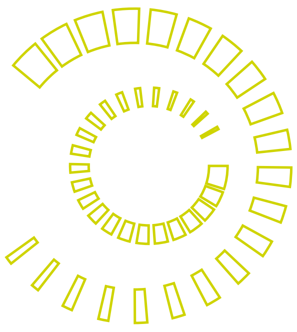
Assessments and guidance published by agile practitioners and consultants can help you assess your agile maturity and fill any gaps.

The agile journey

Agile is a practical path that supports continuous improvement, not a finite state. Assessments and guidance published by agile practitioners and consultants can help you determine your agile maturity and fill any gaps. Many companies are on an agile trajectory, which often starts with inconsistent, barely planned use of the methodology, spearheaded by committed individuals who want to bring about beneficial change. The quality of the outcomes can vary greatly, and testing is largely manual.

At the top end of the agile scale, agile and the lean principles that inform it permeate the entire organization, not just the technology group. Innovation proceeds at a fast, sustained pace, with efficient delivery and continuous improvements in the quality and productivity of the development and DevOps workflows and practices.

Few businesses operate at that level of agile maturity. Many others, however, have successfully scaled agile practices globally, have implemented methods to measure and track the business outcomes of development and DevOps, and rely on automation to keep testing and other processes fast and predictable. They also have realized the kind of collaborative, quality-driven teaming with clear, consistent roles and accountabilities that is essential in bringing agile into reality. Most important, the steadily improving quality of the output demonstrates agile success.



Incomplete agile is counterproductive

Companies can run into issues with evolving their agile practice when they have not clearly defined the business goals that should drive agile or when they underestimate the cultural change and resource commitment that agile necessitates. Often, these organizations practice a sort of incomplete agile. That may mean, for instance, that teams lack the autonomy and tools to produce ever better results, or that product owners are not effective at defining measurable goals together with developers. In partially agile environments it also happens that testing is inconsistent or not automated, or that daily stand-up meetings are poorly managed and tend to run long.

At a tactical level, these practices can easily be corrected, although that may not be all you need to do. Sometimes, the understanding of agile is incomplete or has slipped under daily pressures and good-enough routines, and some people may identify it with scrum or Kanban. A sort of concealed waterfall habit underlying an immature agile practice results in large deployment backlogs that greatly lengthen the time-to-value of developer innovation. This requires a more thorough redirection.

WHY IS DEVOPS TRANSFORMATIONAL?

The vehicle that drives innovation

DevOps—short for development operations—is a critical discipline for microservices. The basic thought behind DevOps is that development followed by a hand-off to technology operations can't ensure that the software reaches its destination—the user's business environment—in a timely and efficient manner. DevOps provides the processes and resources to make that possible. (We will follow up on the very high-level view here in greater depth in a future ebook.)

Similar to agile, where some organizations may think of themselves as agile but really aren't, misconceptions can confuse DevOps conversations. What you could think of as advanced or pure DevOps involves developers who understand that they have to resolve infrastructural concerns and optimize deployment management so that their output can make the impact the business needs.

In some companies, technologists with a system administration background and a sound understanding of automation work in DevOps. Their effectiveness can depend on the quality of their collaboration with developers and their understanding of coding, scripting, and debugging. Other companies think of DevOps primarily as an effort to ensure optimal efficiencies and automate any step they can.

Building a culture of collaboration

Companies will find it difficult to realize the benefits of DevOps when they imagine it as a fixed set of practices that depends on commodity tools, not

Understanding DevOps and IaC tools

The best possible tools for your AWS or Azure platform environment are as critical in making DevOps successful as are optimized processes. Tooling and automation help merge development and operational workflows. Teams can also benefit from communication tools like chat applications or issue tracking and project management systems.

Tools are particularly important when you adopt infrastructure as code (IaC). This makes it easier for developers, engineers, and operations managers to ensure the best possible functioning of infrastructural resources. You rely on code and practices from software development—such as continuous integration or version control—to manage infrastructure elements. By means of cloud APIs, you can provision and scale infrastructures programmatically instead of handling them manually. Infrastructure components defined by code can more easily be deployed or updated in a standardized manner. Technologists often use several IaC tools, such as Cloudformation when deploying services to AWS and Terraform when they deploy services to multiple platforms.

as a transformational practice that is part of an organization's agile culture shift and which complements CI/CD. Sometimes, developer and DevOps teams communicate and collaborate poorly, which may lead to outcomes that are much delayed or don't deliver what's best for the business. This can happen because of interpersonal conflicts, but more often it's the case when team members in both groups are not fully aware and aligned with the company's vision and strategy.

DevOps succeeds best with a collaborative team culture that closes all gaps between development and operations, automation-driven efficiency, and cloud-enabled, continuous deployment. DevOps managers need to harmonize the innovative momentum of developers with the operational drive toward responsive, stable, fast, and secure systems. That can mean merging developer and operational teams, or combining quality assurance and security teams with developers and operations. However, many successful microservices and DevOps practices keep these teams intact. They allow them to maintain accountability for microservices and make it easy for them to collaborate. An optimal ratio of developers to DevOps team members is 10 to one—that enables the best efficiencies in meeting business requirements.

In more advanced DevOps practices, deployment to production is fully automated, and so is the ability to reverse a change or turn a new feature off.

Realizing targeted automation

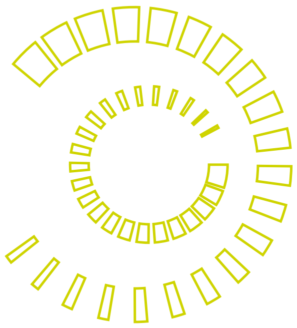
Automation is so ubiquitous in DevOps that some people equate the two. Automation can range from developing, testing, and deploying code, to collecting production metrics and orchestrating the DevOps pipeline. How exactly you realize it depends on the maturity of your agile practice and what kind of automated processes may already be taking place in unit and functional testing.

In more advanced DevOps practices, deployment to production is fully automated, and so is the ability to reverse a change or turn a new feature off. And, at every check-in, you run automated testing as you perform continuous integration and deployment.

Get to DevOps one step at a time

It may only take you manageable, incremental steps to get to a full DevOps environment. It's best to take these one at a time, based on the outcomes you want to commit to, instead of simultaneously. You could start with your top-priority goals for DevOps automation, such as reducing the cost of downtime or accelerating the release of software changes into production.

In a next step, you would identify any bottlenecks that could prevent you from reaching these goals. Maybe developers check in bad code and thereby break the build, or it takes a long time for testers to test minor changes. You document the causes of these bottlenecks before you can start remedying them by applying automation. When you include



automation tasks in your backlog, you can maintain the prioritization and ensure transparency of the improvements that automation introduces.

HOW DO I KEEP MICROSERVICES EFFORTS MOVING APACE?

Accelerating momentum with CI/CD

Continuous integration and continuous deployment (CI/CD) is a key aspect of DevOps. CI/CD aims to reduce cycle times, which aligns with the lean thinking behind agile principles. Cycle time in this case means the period between a concept expressed in a user story and the handing over of the new functionality to users.

You can rely on established best-practice guidance when you implement CI/CD. Always make sure that business requirements are clearly defined and that the company has followed agile practices in building its team and defining and prioritizing requirements. The CIO or another key stakeholder needs to own the CI/CD process and understand its value.

Rapid path forward

CI/CD works best when your team culture fosters collaboration, efficiency, and a fast pace. It really comprises three parts, each of which is a combination of team practices and the use of specialized, automated tools:

» **Continuous integration** integrates work from several developers into a repository several times a day to resolve bugs and speed up collaborative development. Often, integration is expensive and laborious because it involves many manual steps to ensure that code complies with standards, does not ruin existing functionality, and is free from bugs. Continuous integration reduces the complexity of development efforts with automated testing of new code to check for issues and by providing developers with alerts to resolve issues and conflicts.

» **Continuous delivery** automates the steps to propagate a build so code can be released without risk at any time. Continuous delivery takes the complexity out of deployment by ensuring that code is always deployable without painstaking coordination or additional testing. Rigorous, automated testing of the deployment pipeline either promotes code to the next stage or alerts the developer team. Later testing stages closely replicate the eventual production environment.

» **Continuous deployment** automatically deploys each code build that has passed testing. Automatic, continuous deployment makes new functionality or software fixes available quickly to customers or users. User feedback can be received sooner than in a more traditional process. It then becomes easier for developers to incorporate it as they revise their code.

Best-practice tools to power CI/CD and automation

Some organizations use standard tools like Jenkins and Docker, the well-known containerization software, with custom-created processes to realize CI/CD. For instance, you could let all builds and deployments take place as Jenkins jobs, which invoke a Docker container with all of the dependencies to build services, lambda functions, and front-end applications.

Widely adopted automation tools include command-line interfaces for Python 3.6 and 2.7.10, and Jenkins, which supports the Python framework. The command-line tools make it easier for developers to control CI/CD processes such as running builds or deploying releases. They work best when you integrate them closely with communication tools, so you can receive text notifications regarding service status, temporary access to production and quality assurance databases, creation of new repositories, and more. You can enforce unit and application testing for all software you develop, using such tools as Mocha or WebDriver.

Specialized tools like NewRelic, Grafana, Graphite, and others can help monitor application health and performance.

In CI/CD, your choices of tools are more than simple decisions regarding technical capabilities. They have to enable people to work productively and efficiently, producing the highest-quality code possible. The tools have to fit your organization's view of how teams and processes should work. It's best to try out several comparable CI/CD tools, and see how smoothly they interoperate and how well they serve your teams before you commit to them.

Efficient, anytime testing

To perform the ongoing integration, delivery, and deployment of new software code, you need to run extensive testing to validate it. That's why testing and CI/CD are inseparable. You often find CI/CD together with test-driven development, an agile technique that relies on very short development cycles. You begin by writing a usually failing automated test case to identify the functionality you want to deliver. In a next step, you produce just the amount of code needed to pass the test, and then you refactor and optimize the code repeatedly until you deliver the needed capability at the standard the business needs.

Successful microservices initiatives often start with a proof-of-concept project that focuses on a closely defined functionality.

Many CI/CD teams streamline frequent build-and-test sequences by running tests in parallel. Most often they perform smoke, unit, integration, system, and acceptance testing.

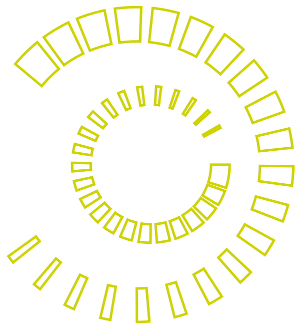
HOW CAN WE GET STARTED?

The next step

Some of the most important decisions related to microservices occur in the area of managing people and teams and creating a collaborative culture. You may need to realign teams and onboard additional skills such as business analysts, product managers, or product directors. In the most successful microservices practices, developers are fully dedicated to specific microservices and don't contribute on multiple teams.

Successful microservices initiatives often start with a proof-of-concept project that focuses on a closely defined functionality. The teams refine their practice based on the findings. They also have to find the microservices approach that works best to meet the organization's needs. That may mean the organic decoupling of legacy functionality by means of macroservices, or the gradual creation and deployment of new code in what's known as the Strangler Pattern.

These crucial decisions depend to a degree on the nature of your legacy monolith. If that software is not a unified code set but a collection of systems, your decoupling strategy will need to be different than if it's all



one large entity. In some situations, you can prioritize the most important features, deliver them as microservices, and add other capabilities by and by.

Before your eager developers get started on their first microservice, you need to have agile and DevOps processes and resources, and the technology architecture in place so you can deploy and deliver microservices. As mentioned, we plan to cover DevOps in greater depth in an upcoming ebook. Topics like team and people management, microservices architectures, and test-driven development might also be worthwhile to discuss in greater detail. Let us know where your interests lie.

For right now:

Let's talk if you want to launch or optimize your microservices practice or take advantage of one of our capability assessments. We offer:

- Operations Capability Review
- Cloud Capability Review
- Code Maintainability Review
- Code Design Review

About Tiempo

Tiempo is widely recognized as one of the leading software engineering companies in the US. Using a combination of nearshore engineering resources, high-performance teams and relentless focus on client outcomes, Tiempo designs, builds and deploys software that makes lives better.

Tiempo is headquartered in Tempe, Arizona, with four world-class software development facilities in Mexico. Tiempo has been recognized annually by Inc. Magazine as one of the Fastest-Growing Private Companies in America.

Contact us at:
contact@tiempodev.com
602-910-4646

U.S. HEADQUARTERS
1050 W. Washington St, Suite 120
Tempe, AZ 85281 USA

GUADALAJARA
2464 Justo Sierra
Guadalajara, Jalisco, México

GUADALAJARA
San Dinisio 25
Jardines de San Ignacio
Zapopan, Jalisco México

HERMOSILLO
545 García Morales Blvd.
Hermosillo, Sonora, México

MONTERREY
CIT2 Eugenio Garza Sada
Monterrey, Nuevo León, México

www.tiempodev.com