Innovation Insight for Microservices

Published: 4 March 2019 **ID:** G00384511

Analyst(s): Anne Thomas, Aashish Gupta

Microservices enable unprecedented agility and scalability, but the architecture causes significant cultural disruption. Application leaders should understand this new design paradigm, its prerequisites and its disruptive impact before determining where and when to use it — if at all.

Key Findings

- Microservices support precise scalability and enable continuous delivery of new features and capabilities, but the microservices architectural model is complex and culturally disruptive.
- Microservices adoption efforts are destined to fail if you don't make corresponding changes to traditional application development, data management and operations practices, as well as organizational and communication structures.
- Application delivery teams must have strong architectural skills and be competent in agile and DevOps practices before adopting microservices. Example practices include extreme programming (XP), test-driven development (TDD), continuous integration (CI), continuous delivery (CD) and application release automation (ARA).
- Microservices require new and complex application infrastructure, the market for which is fragmented and immature.

Recommendations

For application leaders responsible for development and platform strategies:

- Determine whether your organization is mature enough to adopt microservices by assessing your competency in terms of technologies, skills, processes, collaboration and willingness to change.
- Ensure that you are adopting microservices for the right reason. If you aren't building dynamic systems, you shouldn't adopt microservices.
- Tie every microservices investment to specific business outcomes. Take a pragmatic approach to adoption by applying the paradigm selectively and incrementally, where microservices can deliver business value.

- Use alternatives such as miniservices or high-productivity platforms to gain agility benefits if your organization isn't prepared to adopt microservices, or where the cost of adopting microservices would outweigh the benefits.

Table of Contents

List of Figures

Strategic Planning Assumption

By 2022, 70% of organizations that attempt to adopt microservices will have found the effort too disruptive and will have switched to miniservices instead.

Analysis

Microservices enable unprecedented agility and scalability. Using microservices, organizations can build and deploy individual application features as soon as they are ready, and scale just the parts of an application that cause bottlenecks.

But microservices aren't for everyone. Application leaders must understand their disruptive impact before determining where and when to use them — if at all.

As application leader, you must exercise caution in adopting microservices architecture (MSA), because it requires:

- Significant investment in people, processes and platforms.
- Highly skilled application architects and DevOps practitioners who are comfortable adopting new architectural paradigms as well as designing and operating complex and dynamic systems.
- Fundamentally different ways of doing data management, reporting and analytics.

Definition

A microservice is a tightly scoped, strongly encapsulated, loosely coupled, independently deployable and independently scalable application component. Microservices architecture applies service-oriented architecture (SOA) and domain-driven design (DDD — see Note 1) principles to deliver distributed applications. MSA has three core objectives: development agility, deployment flexibility and precise scalability.

Description

MSA is an architectural paradigm and implementation model for building services and cloud-native applications. Modern applications should be built using the mesh app and service architecture (MASA), whereby an application comprises multiple front-end user interface components (apps) and multiple back-end components (services). MSA can be used to implement any application functionality, but microservices are typically used to implement features within a back-end service (see "Adopt a Multigrained Mesh App and Service Architecture to Enable Your Digital Business Technology Platform").

> The essential goal of MSA is to enable development teams to independently build, test, package, deploy and scale individual components within an application.

More than anything, the benefit afforded by MSA is agility. MSA was made famous by digital business leaders such as Netflix, Amazon and Twitter, which adopted it to support their ever-increasing traffic volumes and their competitive requirements to continuously deliver new features and capabilities. Most organizations don't have to deal with Netflix-level scalability concerns, but

they do need to deliver new features more quickly. Therefore, most organizations that adopt microservices do so to support their CD practice.

CD is a software delivery mindset to produce and deploy new features and capabilities into production incrementally rather than in project batches and release trains. Organizations with mature CD practices typically deploy multiple updates to production systems every day to meet rapidly evolving business requirements and demands. CD requires advanced agile and DevOps practices that automate the process of building, testing, packaging and deploying code.

But CD of features within a monolithic code base is risky. If something goes wrong, how do you determine what caused the failure? MSA enables organizations to deliver new features as independent components, which reduces CD risks. Today, leading-edge companies in banking, retail, consumer goods and other industries are adopting microservices to support CD (see "Scaling Digital Business Requires the Continuous Delivery of Incremental Value" for more information on CD).

The overarching principle governing MSA is independence. MSA principles, constraints and design patterns minimize the dependencies between components. New features can be developed and deployed as soon as they are ready; there's no need to wait for the next scheduled release to deliver a critical feature.

Service Granularity

The term "microservice" implies "small," but size is relative. MSA is about packaging and deployment more than about size: A microservice, by definition, is an independently deployable component — the term doesn't specifically indicate the size or scope of the component.

The proper size of a microservice should be determined based on agility requirements and the separation-of-concerns principle. Agility tends to increase with granularity — but only to a point. A core principle of MSA is cohesion, which aims to reduce dependencies between microservices. The ideal granularity is determined by the change boundaries of the system. Use the smallest possible granularity that allows you to deliver an enhancement within the scope of one microservice.

> Granularity delivers agility, but it also increases complexity.

Not all applications have extreme agility requirements, so it's often appropriate to relax the MSA constraints, to use more traditional patterns and models, and to build larger components. But as granularity decreases and constraints become more relaxed, the agility and scalability benefits decrease and the components begin to look less and less like microservices.

Granularity also impacts management and governance requirements. Tightly scoped microservices are easily replaced and, therefore, quite disposable. As services increase in scope, they become more valuable and less likely to be viewed as disposable. And if you publish a service's API for reuse, you create dependencies that impact your ability to change it.

Accordingly, Gartner uses different names for different levels of granularity (see Figure 1):

- **Microservice —** *To enable continuous delivery.*

  A microservice is a physically independent and tightly scoped application component that strictly adheres to MSA design constraints in order to support extreme agility and scalability requirements. It is independently deployable and scalable. It has a single responsibility and implements an individual feature. If a microservice manages persistent state, it has exclusive rights to update the data store. It never publishes its API for external consumption.

- **Miniservice —** *To improve application agility while avoiding some of the more disruptive aspects of MSA.*

  A miniservice is like a microservice but has a larger scope and relaxed architectural constraints. Like a microservice, a miniservice is a physically encapsulated, loosely coupled, independently deployable and scalable application component. Unlike a microservice, a miniservice often implements more than one feature. If it manages persistent state, it should have exclusive rights to update the data store, although in some cases a small number of related miniservices may share the data store. You might publish the API for external consumption.

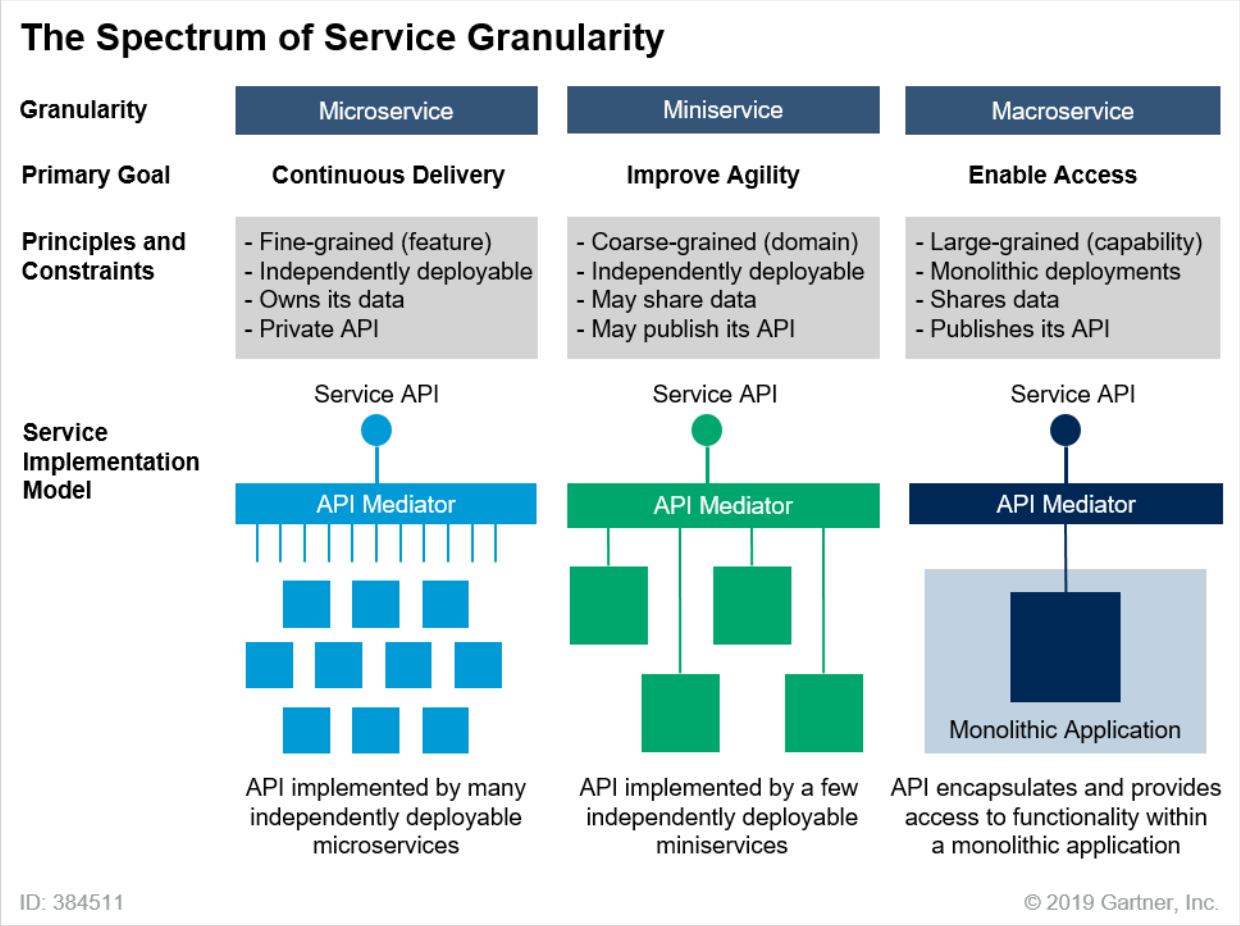
  Be aware that as you relax the architectural constraints, you introduce interdependencies that impede agility. Often you must deploy a group of services together. Miniservices enable agility, but not to the same degree that microservices do. As you relax the constraints that ensure independence, you reduce the application's agility and scalability benefits.

- **Macroservice —** *To enable access to functionality within a monolithic application.*

  A macroservice is what many people would call a traditional SOA service, or perhaps an API. It logically encapsulates a capability within a monolithic application and exposes that capability via a published API. You create a macroservice by building an API within the application's code or by using an integration tool to build an API adapter.

  For example, you could build a REST API that provides access to functionality within a Java EE application that's deployed in an application server. Because the functionality is implemented within the monolithic application, the service isn't independently deployable or scalable. You must test and redeploy the entire Java EE application to make a change to the service, and you must scale the entire application in order to scale the service.

Figure 1. Different Names for Different Levels of Service Granularity

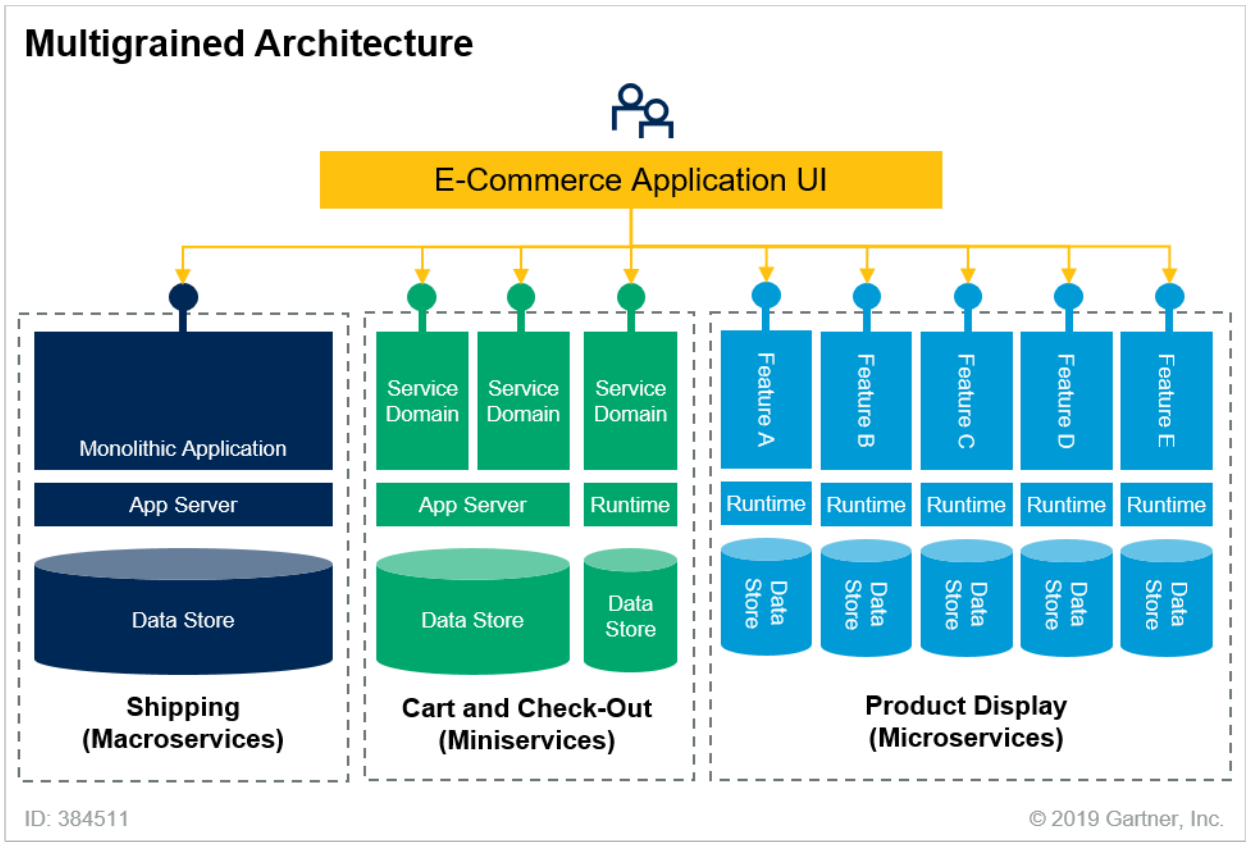


Source: Gartner (March 2019)

This service granularity spectrum refers to the way a particular service is implemented. Take a check-out service in an e-commerce application as an example: A **macroservice** provides an API to the check-out functionality, which is implemented and deployed as part of the monolithic e-commerce application. A **miniservice** would refactor the check-out functionality into an independently deployable unit. A **microservice** implementation would refactor the check-out functionality into multiple independently deployable units — each implementing a separate feature of the check-out functionality. All three implementation models could support the same check-out API.

A given application may and often does comprise multiple services of various levels of granularity. Figure 2 shows a multigrained e-commerce application including macroservices, miniservices and microservices implementing different capabilities. When building a new application, you should implement most capabilities as coarse-grained miniservices unless you have a compelling architectural reason to adopt coarser- or finer-grained granularity. When rearchitecting an existing application, leave functionality in the monolith until you have a reason to refactor it into a separately deployable component.

Figure 2. Multigrained Services Within One Application



Source: Gartner (March 2019)

Further Reading: Guidance About Service Granularity

- "Adopt a Multigrained Mesh App and Service Architecture to Enable Your Digital Business Technology Platform"
- "Choose the Right Service Granularity to Enable Digital Business"
- "Choose the Right Approach to Modernize Your Legacy Systems"
- "Refactor Monolithic Software to Maximize Architectural and Delivery Agility"
- "How to Design Microservices for Agile Architecture"

Frequent Misuse of the Term "Microservices"

People frequently misuse the term "microservices" and confuse it with other terms, such as "service," "API" and "container." "Microservices" refers to a complex architectural paradigm that developers use to build new applications or to rearchitect existing applications. It does *not* refer to:

- **A reusable service —** Something that is designed for reuse is a service, not a microservice. Reuse creates dependencies that reduce independence and impact agility, therefore reuse is an antipattern in the microservices paradigm.
- **Any service with a REST API —** A microservice may have a REST API, but it doesn't have to. Many MSA practitioners use other protocols, such as gRPC or Kafka. On the flip side, many coarse-grained and monolithic services also expose REST APIs.
- **Any software component —** While all microservices are software components, not all software components are microservices.
- **Any application deployed in a container —** Typically, microservices are deployed in containers, but monolithic applications can also be.
- **An integration technology —** Developers use MSA as a way to design and build application functionality. While you could handcraft an integration adapter or service orchestration function using your favorite programming language and then deploy the code as an independently deployable microservice, that's not especially common. Most people use integration tools, such as an enterprise service bus (ESB) or integration platform as a service (iPaaS), for this type of functionality.
- **A SaaS capability —** This type of SaaS capability is a reusable service, not a microservice. In the Gartner taxonomy, we refer to this type of offering as API SaaS. The provider could implement the service using microservices but, as a SaaS consumer, you should not need to be aware of the service's implementation model.

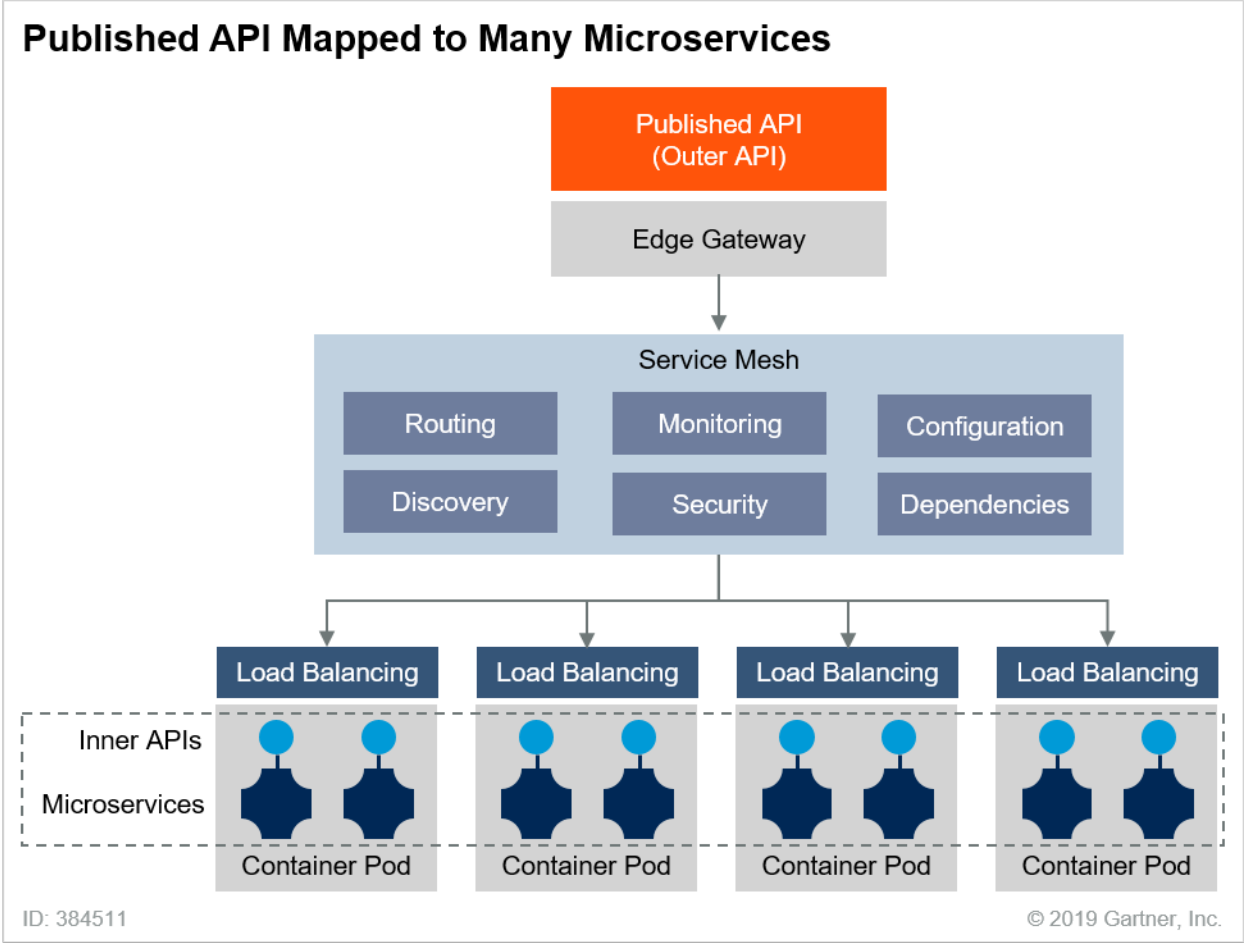
APIs: Published (Outer) Versus Private (Inner)

The fundamental goal of the paradigm is to enable independent and uninhibited deployment of new features to meet business demands. MSA allows you to deploy a new feature without needing to redeploy the rest of the application. This overarching goal for independence means that that you should avoid direct dependencies between application components. As a result, you should never directly publish a microservice's API for external consumption (outside the scope of its immediate context). In most circumstances, a published API exposes a collection of related capabilities, each implemented by separate microservices.

Figure 3 illustrates a single published API (outer API) mapping to multiple microservices, each deployed in a separate container pod and each having a private API (inner API). An edge gateway maps outer API requests to specific inner API methods. A service mesh then supports dynamic discovery, routing and load balancing to deliver the request to an appropriate microservice instance. It also manages intermicroservice dependencies and communications.

Figure 3 illustrates that microservices live inside an application service, and it gives you a sense of the complexity of microservices infrastructure (see “How a Service Mesh Fits Into Your API Mediation Strategy”).

Figure 3. A Published API Maps to Many Microservices That Each Have a Private API



Source: Gartner (March 2019)

Benefits and Uses

> MSA is particularly beneficial to organizations that are building a CD practice and to organizations that have applications with extreme scalability requirements.

MSA delivers the following benefits:

- **Agility —** MSA facilitates a CD practice, enabling an organization to deploy new application features as quickly as they can be developed.
- **Risk reduction —** Because MSA deployments are small and independent, the risk associated with each deployment is reduced.
- **Distributed development —** MSA reduces dependencies between components, which enables feature teams to work more autonomously.
- **Technology flexibility —** Because microservices are loosely coupled, development teams can use a variety of technologies and languages to implement the different facets of an application.
- **Scalability —** MSA provides the means to scale just the parts of an application that cause contention and bottlenecks.
- **Resiliency —** MSA design patterns, when applied correctly, yield self-healing applications that can continue operating in the face of partial outages, and that can recover rapidly and gracefully.

Organizations that want to gain greater agility but are not yet doing CD or don't have extreme scalability requirements can get good results with a lot less complexity by using miniservices. Miniservices and microservices are compatible with other architectural models — most organizations that have adopted MSA have applied the paradigm iteratively to existing applications. They refactor their monolithic applications and services, and build miniservices and microservices for just the features of the application that require agility and scalability.

Adoption Rate

MSA is still in an early stage of adoption, although it is on the brink of crossing the chasm to reach the early majority. The innovators that first adopted MSA typically compete in web-scale digital business markets, such as:

- Digital media — iHeartRadio, Netflix and SoundCloud, for example
- Social networking — LinkedIn, Pinterest, Twitter
- E-commerce — Amazon, eBay, Gilt
- Social commerce — Airbnb, Hailo, Uber

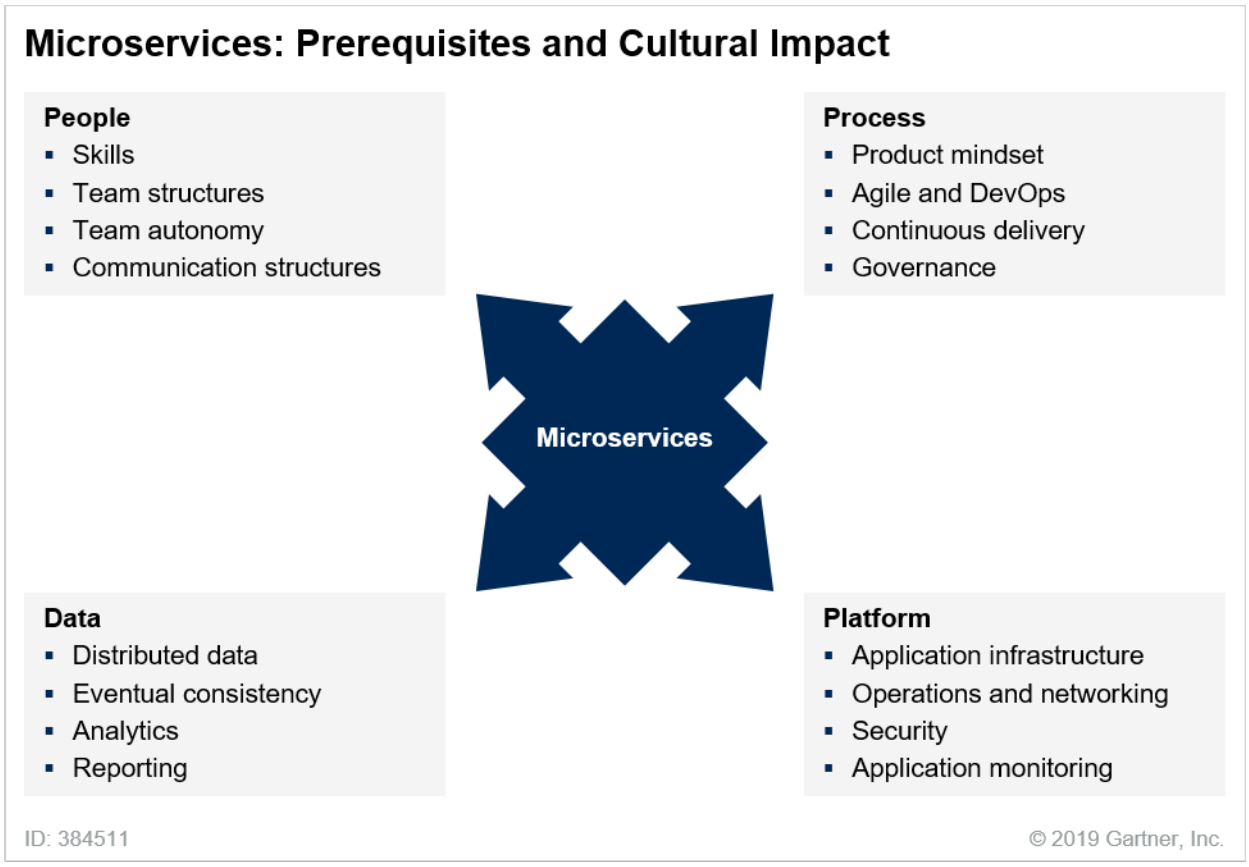
These organizations invested in the new paradigm to enable extreme agility and scalability. A growing number of leading-edge companies in more-mainstream industries, such as finance, retail and consumer goods, have also been successful in adopting MSA. Noteworthy early adopters include Capital One, Coca-Cola, Disney, GE, Goldman Sachs, Nike and Red Bull.

The Costs of Microservices Adoption

The agility and scalability benefits afforded by microservices are compelling, but they come at a significant cost. Microservices adoption is highly disruptive to many aspects of the organization, in the following ways (see Figure 4):

- **Technical and cultural —** Application leaders must be willing to facilitate and manage unsettling technical and cultural changes as they introduce the architecture.
- **Agile and DevOps —** MSA requires solid competency in agile and DevOps practices, and developers need strong application architecture skills and a willingness to adopt new design models and patterns.
- **Data management —** Microservices patterns fundamentally change how data is managed within an application, which impacts data governance, reporting and analytics practices.
- **Product mindset —** MSA requires a product mindset, which is an organizational change that affects budgeting, staffing and planning outside of IT. It impacts funding models, roles and responsibilities, development team structures, and corporate communication structures.
- **Team autonomy —** Teams must be given autonomy and authority to build their components in the best way they see fit, which impacts established governance practices.
- **Application infrastructure —** MSA also requires new and complex application infrastructure, which in turn impacts runtime operations, networking and security models.

Figure 4. The Prerequisites and Cultural Impacts of Microservices



Source: Gartner (March 2019)

Further Reading: The Disruptive Impact of Microservices

- "Mastering the Role of Products in the Digital Era"
- "Use These Three Best Practices to Power Your Microservice Adoption"
- "Digital Business Requires a New Mindset, Not Just New Technology"
- "Emerging Practices in Organizational Design for Digital Product Delivery Organizations"
- "How to Build Effective Microservices Teams"
- "Adopting the Spotify Model for Better Enterprise Agile Scaling"
- "Innovation Insight for Microservices Runtime Infrastructure"
- "Solution Path for Using Microservices Architecture Principles to Deliver Applications"
- "How to Succeed With Microservices Architecture Using DevOps Practices"
- "How to Design Microservices for Agile Architecture"
- "Working With Data in a Microservices Architecture"

Risks

MSA is complex. When it's applied inappropriately, the resulting applications may perform poorly and not yield the desired agility benefits. Many organizations that say they are doing MSA are in fact doing a combination of miniservices and macroservices, and getting disappointing results. Gartner has heard a growing number of organizations express frustration with their MSA experiments. Many organizations aren't willing to fully embrace the paradigm and the cultural disruption that goes with it. Some organizations are abandoning their MSA investments due to high costs, project delays and unmet expectations.

MSA is a significantly different architectural paradigm whose adoption disrupts traditional development, operations and data management practices. It requires advanced architecture skills, mature agile processes and robust DevOps automation. It requires new roles, responsibilities and organizational structures.

MSA also requires new application infrastructure that will impact operations teams. Most MSA adopters use containers and PaaS for their deployments. Additional MSA infrastructure technologies include API gateways, a service mesh and new application monitoring technology. The

microservices infrastructure market is fractured, immature and evolving quickly. Much of the infrastructure is available via open-source projects, although some commercial products are starting to emerge. (See Further Reading for more information about what it takes to design and operate a microservices infrastructure.)

Further Reading: Designing and Operating Microservices Infrastructure

- "Innovation Insight for Microservices Runtime Infrastructure"
- "Selecting a Cloud Platform for DevOps Delivery With Microservice Architecture"
- "Innovation Insight for Service Mesh"
- "How a Service Mesh Fits Into Your API Mediation Strategy"
- "Selecting the Right API Gateway to Protect Your APIs and Microservices"
- "Building Identity Into Microservices"
- "Advance Your Application Performance Monitoring Strategy to Support Microservices"
- "Container Security — From Image Analysis to Network Segmentation, Options Are Maturing"

Beware of Application Software Vendors Touting "Microservices"

Be skeptical of vendors claiming to support microservices. Many vendors have been "microservice-washing" their marketing messages without actually adding support for independently deployable components. Support for REST APIs and/or containers is not the same as support for microservices. If a vendor talks about reusing services or deploying services into an application server, it probably doesn't understand the objectives of microservices. Its tools are probably more appropriate for building macroservices.

Also beware of application vendors that tell you they have implemented their application using microservices. Using the definition provided in this research, ascertain whether the vendor is truly delivering microservices. If it supplies its application as dozens or hundreds or thousands of independent components that you must deploy and operate yourself, you may want to steer clear of that product.

Evaluation Factors

If you have pressing agility and scalability requirements, consider adopting MSA. But don't attempt to adopt MSA unless you meet the following prerequisites:

- You have one or more business units with both a need for continuous change and the ability to cope with frequent changes to their systems.
- Your existing architectures inhibit your ability to meet agility and scalability requirements.
- You have mature agile and DevOps practices, have robust automation throughout your development life cycle (including testing and deployment), and are working to improve your CD practice.
- Your development teams are prepared to adopt unfamiliar design methodologies, design patterns and infrastructure technologies.
- Your data management teams are prepared to develop new data management strategies that accommodate distributed and denormalized data and eventual consistency. (This factor is a deal-breaker for many organizations.)
- Your operations teams are prepared to deploy new application infrastructure, as well as employ new practices and technologies to host, monitor and manage microservices.

Microservices Alternatives

You can increase your agility using other methods, although none can deliver the same scalability advantages that MSA can.

Miniservices

The most popular alternative to microservices is miniservices. Organizations that have mature SOA skills can increase their agility and improve the maintainability of their applications by refactoring monolithic systems into independently deployable, coarse-grained components, where each component implements a different entity domain.

Individual miniservices are much easier to update than monolithic applications, and developers can deliver new applications quickly by composing existing miniservices. Without a rigorous focus on independence, though, miniservices will not deliver the same agility and scalability advantages as MSA. Miniservices will be less disruptive to most organizations, however.

Low-Code Methods

Organizations can also gain significant agility advantages using low-code development technologies, although these alternatives often have significant scalability limits. Popular low-code methods include:

- **Low-code application platforms —** Available as both on-premises software and PaaS offerings, these environments allow professional and nonprofessional developers to rapidly build applications using metadata-driven visual tooling, fourth-generation languages (4GLs) and domain-specific languages (DSLs).

  See "Magic Quadrant for Enterprise High-Productivity Application Platform as a Service" and "Market Guide for High-Productivity Rapid Application Development Tools."

- **Business process management suites and business rule engines —** These suites allow professional and nonprofessional developers to rapidly build and maintain applications using visual business process modeling tools and declarative business rules.

  See "Magic Quadrant for Intelligent Business Process Management Suites" and "Critical Capabilities for Intelligent Business Process Management Suites."

Recommendations

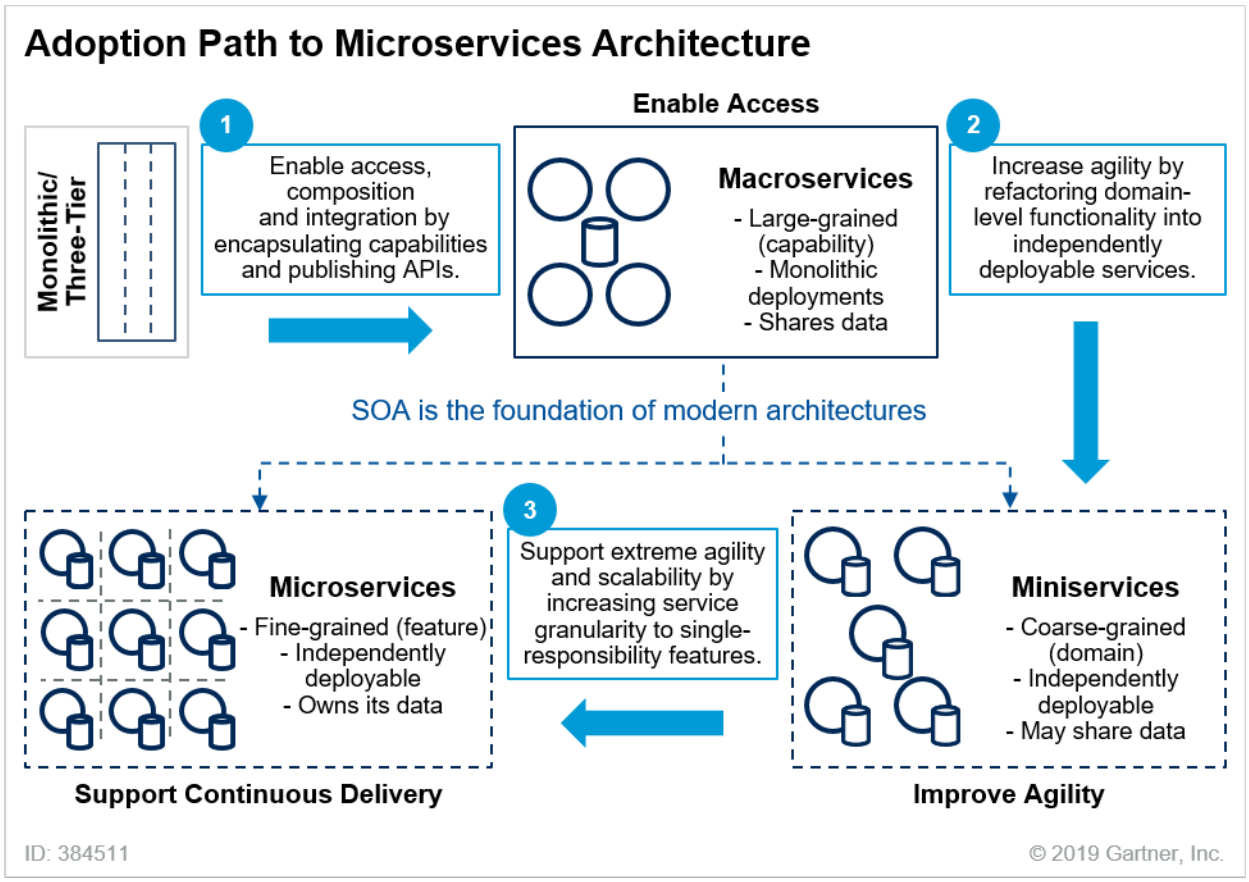
Approach MSA with caution. Don't get caught up in the hype, and don't feel compelled to adopt the paradigm just because everyone is talking about it. MSA can deliver better agility and scalability advantages than any other paradigm, but most organizations don't have the type of extreme agility and scalability requirements that push teams to adopt it. Most organizations will find MSA too costly and disruptive to warrant the investment. And there are other ways to increase agility — organizations can derive tremendous value from the miniservices model, for example.

The MSA paradigm requires advanced architectural skills, and the MSA technology market is immature. Gartner expects that vendors will develop frameworks that simplify MSA adoption, but robust mainstream tools are still a year or two away.

If the immature state of the market doesn't deter you, determine whether your organization is prepared to adopt MSA based on the evaluation criteria discussed above. If you do decide to explore MSA, take it slowly, following the steps below and shown in Figure 5:

- Start with basic SOA adoption and encapsulate functionality in your monolithic applications, exposing them as macroservices.
- From there, start to refactor the monolith and create miniservices that implement domain-level functionality.
- Shift to microservices when you think you are ready to start decomposing your monolithic databases into flat, microservice-ready data models.
- Use microservices only where you need to — in applications with extreme agility or scalability requirements. Experiment with the new patterns and technologies, and build expertise.
- Don't be afraid to mix paradigms. Most people use microservices in combination with monolithic and miniservice architectures.

Figure 5. Adoption Path to Microservices Architecture



Source: Gartner (March 2019)

Deploy your microservices in a managed container environment, such as a Kubernetes or Cloud Foundry system. Most of these systems provide the foundational infrastructure required for microservices. Alternatively, you can deploy microservices into a function PaaS (fPaaS) such as Amazon Web Services (AWS) Lambda or Microsoft Azure Functions. Also, look for frameworks that help developers apply DDD and adopt MSA design patterns (see “Innovation Insight for Microservices Runtime Infrastructure” for more information and recommendations for assembling your infrastructure).

Representative Providers

The MSA infrastructure market is fractured and immature. The following providers represent suppliers of MSA tools, frameworks, libraries and management technologies:

- Amazon Web Services — AWS Lambda (an fPaaS), Amazon Elastic Container Service (ECS), Amazon Elastic Container Service for Kubernetes (EKS), AWS Fargate, AWS App Mesh (a service mesh)
- Buoyant — Linkerd (a service mesh)

- Google — Google Cloud Functions (an fPaaS), Google Kubernetes Engine (GKE), Knative (frameworks for building functions and microservices), Istio (a service mesh)
- HashiCorp — HashiCorp Consul (a service mesh), Nomad (a cluster scheduler)
- Istio — Istio (a service mesh)
- Microsoft — Microsoft Azure Service Fabric (a microservices framework), Microsoft Azure Kubernetes Service (AKS)
- Netflix — Netflix Common Runtime Services and Libraries (a service mesh)
- Pivotal — Pivotal Cloud Foundry (multicloud application platform supporting both Cloud Foundry and Kubernetes container systems); also Spring Boot and Spring Cloud Netflix (frameworks for building independently deployable services and for using the Netflix service mesh, respectively)
- Red Hat — Red Hat OpenShift Container Platform (a multicloud application platform supporting Kubernetes), OpenShift Service Mesh (a service mesh)

Acronym Key and Glossary Terms

API	application programming interface
CD	continuous delivery
DDD	domain-driven design
MSA	microservices architecture
PaaS	platform as a service
SaaS	software as a service
SOA	service-oriented architecture

Gartner Recommended Reading

Some documents may not be available as part of your current Gartner subscription.

“Adopt a Multigrained Mesh App and Service Architecture to Enable Your Digital Business Technology Platform”

“Solution Path for Achieving Continuous Delivery With Agile and DevOps”

“Use These Three Best Practices to Power Your Microservices Adoption”

“Innovation Insight for Microservices Runtime Infrastructure”

“Innovation Insight for Service Mesh”

“How a Service Mesh Fits Into Your API Mediation Strategy”

“Solution Path for Using Microservices Architecture Principles to Deliver Applications”

“How to Succeed With Microservices Architecture Using DevOps Practices”

“How to Design Microservices for Agile Architecture”

“Working With Data in a Microservices Architecture”

Evidence

Analysis is derived from interviews with leading microservices practitioners, including Adrian Cockcroft, VP of cloud architecture strategy at AWS and former CTO for Netflix, and Russell Miles, CEO at ChaosIQ.io.

Additional sources include presentations at application architecture conferences and the technology blogs published by numerous organizations that have built systems using MSA. For example:

- Capital One
- Gilt
- Groupon
- Netflix
- Nike
- SoundCloud
- Uber

Note 1 What Is Domain-Driven Design?

Domain-driven design (DDD) is a software design philosophy for building complex systems in which the design is centered on the real-world business domain that the software serves. It strives to tame system complexity to ensure that the software can be easily changed or extended as the business domain evolves. Learn more at:

- “What Is Domain-Driven Design?” Domain-Driven Design Community
- “Domain-Driven Design Quickly,” InfoQ
- E. Evans, “Domain-Driven Design Reference,” Domain Language

GARTNER HEADQUARTERS

Corporate Headquarters
56 Top Gallant Road
Stamford, CT 06902-7700
USA
+1 203 964 0096

Regional Headquarters
AUSTRALIA
BRAZIL
JAPAN
UNITED KINGDOM

For a complete list of worldwide locations,
visit http://www.gartner.com/technology/about.jsp