# Turning products into services: using microservices to ease servitization

*13 March 2024 - 7 min. read*

*Decoupled Architectures* *Microservices* *Servitization*

In the dynamic world of cloud computing, where agility and scalability are critical, service decoupling emerges as a cornerstone for building resilient and adaptable systems. In our articles, we've delved deep into various technological aspects of servitization, exploring asynchronous communication patterns, decoupled architectures leveraging event buses, and strategies for microservices decoupling.

This article aims to synthesize these insights, providing a comprehensive perspective on the importance of service decoupling and its relationship with servitization. We will illuminate the path toward more robust, flexible, and service-oriented cloud architectures by weaving together the threads of our previous explorations.

Throughout this article, we will reference specific topics covered in previous articles to provide readers with a comprehensive understanding of the subject matter and facilitate a more profound exploration of relevant themes.

Without further ado, let's dive in!

## Servitization and Microservices-based Architectures

Servitization entails transforming products into services, enabling organizations to deliver value to customers through more flexible and personalized experiences.

**Servitization is more about business than technology**; however, it is closely linked to microservices as the primary architectural model in cloud architectures.

The fact is that building a successful service is becoming more and more challenging due to very high customer expectations and market standards. Building a quality service takes

time and effort and is often complex, especially if the customer base is extensive and geographically distributed.

Here is where microservices come in handy.

The essence of the microservices paradigm lies in **breaking down complex applications** into smaller, autonomous components(microservices) that operate independently yet collaboratively to fulfill various business functions. This approach enhances the modularity and flexibility of the system and facilitates seamless adaptation to changing market demands and customer needs.

One of the key advantages of adopting a microservices-based architecture is its intrinsic **flexibility**. By decomposing applications into discrete, self-contained services, organizations gain the ability to modify, update, and scale individual components without disrupting the entire application. This modularity allows for rapid innovation and iteration, empowering teams to respond swiftly to market dynamics and customer feedback.

**Scalability** is another significant benefit of the microservices approach. Unlike monolithic architectures, where scaling frequently involves replicating the entire application stack, microservices enable granular scalability that optimizes resource utilization and enhances system performance and responsiveness under unpredictable workload conditions.

Furthermore, the **isolation of failures** innate in microservices architectures enhances system resilience and fault tolerance. In a monolithic system, a failure in one component can cascade through the entire application, leading to widespread downtime or service disruptions. In contrast, with microservices, failures are contained within individual services, preventing them from completely disrupting the user experience and minimizing the overall impact on system availability.

Microservices architectures also facilitate agile development and deployment practices. With each service operating independently and potentially managed by autonomous teams, organizations can adopt agile methodologies and DevOps practices to accelerate development cycles and streamline release processes. This **agility** enables continuous innovation and iteration, driving competitive advantage and customer satisfaction.

Moreover, the **modular** nature of microservices simplifies integration with third-party services and external systems, enabling seamless interoperability. Decoupling services and defining clear interfaces makes integrating new features and functionalities easier with minimal disruption to existing systems, promoting innovation and collaboration across the ecosystem.

Of course, building an application based on microservices architecture may also present challenges compared to traditional monolithic applications.

The biggest one is the **increased complexity of managing multiple services and their interactions**. Unlike monolithic applications, where all components are tightly integrated, good microservices are loosely coupled, which requires careful orchestration and coordination.

There is also the overhead of managing the infrastructure for multiple services. Each component typically runs in its container/instance/FaaS, leading to a proliferation of resources to manage and increasing operational complexity regarding deployment, monitoring, and scaling.

Additionally, debugging and troubleshooting can be more challenging in a microservices environment. With multiple services interacting, identifying and diagnosing issues can be complex, mainly when errors propagate across service boundaries.

Despite these challenges, the benefits of microservices often outweigh the difficulties, making this paradigm more and more and more relevant.

## Decoupling

Decoupling plays a pivotal role in an excellent microservice-based application. It means designing components within a system to operate independently, with minimal or no reliance on each other's internal workings. This principle lies at the heart of microservices architecture, where services are loosely coupled to enable better fault isolation and scalability.

In **synchronous systems**, where components interact in real time, techniques like load balancers play a crucial role in decoupling. Load balancers distribute incoming traffic across multiple instances of a service, allowing for horizontal scaling and ensuring that no single component becomes a bottleneck. This synchronous decoupling mechanism enables organizations to handle increasing workloads without compromising performance or reliability.

On the other hand, **asynchronous systems** rely on message-based communication to achieve decoupling. Queues, event buses, and notification systems are critical components in asynchronous architectures, facilitating loose coupling and fault isolation.

**Queues** act as intermediaries between producers and consumers, buffering messages and allowing them to be processed later. By decoupling message production and consumption

timing, queues enable more efficient resource utilization and smoother system operation.

**Event buses** are the backbone of distributed architectures, enabling seamless communication between services without direct coupling. They provide a publish-subscribe model where producers publish events to a central bus, and consumers subscribe to specific event types of interest. This decoupled communication model allows for greater flexibility and extensibility, as new services can be added or removed without disrupting the overall system.

Similarly, **notification systems** like Amazon SNS (Simple Notification Service) enable asynchronous communication between components, allowing real-time updates and alerts. By decoupling the sender and receiver of messages, notification systems enable more resilient and responsive architectures, where components can react dynamically to environmental changes.

## Serverless

In the ever-evolving landscape of cloud computing, serverless architecture has emerged as a game-changer, offering a paradigm shift in building, deploying, and managing distributed systems. At the forefront of this revolution are services like AWS Lambda and Fargate to power microservices applications with reliability and built-in scalability.

AWS Lambda allows developers to run code without provisioning or managing servers. Instead of worrying about infrastructure, developers can focus on writing code and defining the events that trigger its execution. This serverless approach offers unparalleled scalability and flexibility, as Lambda automatically scales to handle (nearly) any workload.

However, **serverless services can also be used to build the decoupling layer of modern microservice applications**.

Amazon SQS (Simple Queue Service), Amazon SNS (Simple Notification Service), and Amazon EventBridge are crucial in decoupling.

SQS provides a reliable and scalable message queueing service, enabling asynchronous communication between different system parts.

On the other hand, SNS enables pub/sub messaging, allowing components to communicate in real-time without direct coupling.

Amazon EventBridge serves as a central event bus, facilitating the integration and orchestration of events across different services and systems. EventBridge allows

developers to define rules and triggers for processing events, enabling seamless communication and coordination between disparate components.

In addition to scalability and reliability, serverless architecture offers other benefits, such as rapid development and deployment cycles. With Lambda, developers can quickly iterate on their code, deploying changes in seconds rather than days or weeks.

This agility allows organizations to respond rapidly to changing business requirements and market conditions, giving them a competitive edge in today's fast-paced digital landscape.

## Wrapping up

In our journey, we explored servitization and learned how it emphasizes transforming products into services. Since a business's success is more influenced by the performance of services instead of selling products, delivering high-quality digital services is increasingly challenging and critical for a successful organization.

Meeting high-quality standards and customer expectations regarding uptime, responsiveness, and overall user experience requires technological solutions that enable rapid development, high quality, and resilience of the customer-facing service.

Adopting decoupling techniques and asynchronous approaches, such as leveraging serverless services like AWS Lambda and Fargate, exemplifies how technological responses align with the principles of servitization. By breaking down applications into small, more manageable services and utilizing cloud-native technologies, organizations can develop agile, responsive solutions that meet the evolving needs of their customers. This fusion of servitization principles with technological advancements facilitates the delivery of superior digital services

So, this is all for today's article; stay tuned for other exciting content!

## Resources

- A detailed dive into asynchronism in distributed systems
- Decoupling lambda services using queues
- Decoupled Cloud Architectures with Event Buses

---

## About Proud2beCloud

**Proud2beCloud** is a blog by beSharp, an Italian APN Premier Consulting Partner expert in designing, implementing, and managing complex Cloud infrastructures and advanced

services on AWS. Before being writers, we are Cloud Experts working daily with AWS services since 2007. We are hungry readers, innovative builders, and gem-seekers. On Proud2beCloud, we regularly share our best AWS pro tips, configuration insights, in-depth news, tips&tricks, how-tos, and many other resources. Take part in the discussion!

---

**Alessio Gandini**

Cloud-native Development Line Manager @ beSharp, DevOps Engineer and AWS expert.Since I was still in the Alfa version, I'm a computer geek, a computer science-addicted, and passionate about electronics all-around.At this moment, I'm hanging out in the IoT world, also exploring the voice user experience, trying to do amazing (Io)Things.Passionate about cinema and great TV series consumer, Sunday videogamer

---