Microservices – Definition, Principles and Benefits

Last Updated: December 26, 2020 By: Lokesh Gupta Microservices Microservices, SOA

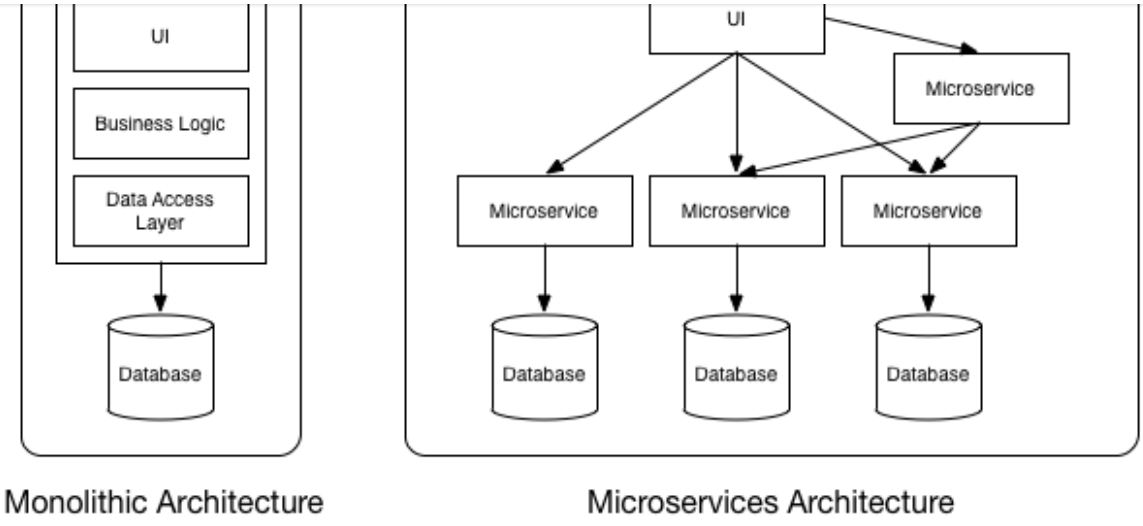
Microservices is the latest buzzword in the industry and everyone seems to be talking about it, in one way or another. Let's understand **what are microservices**? In this tutorial, we will try to understand the definition, concepts and **principles of microservices**.

Table of Contents

1. Definition of Microservices

Today, microservices are one of the increasingly popular architecture patterns next to SOA (Services Oriented Architecture). If you are following industry trends, then you realize that today business houses are no longer interested in developing large applications to manage their end-to-end business functions as they did a few years ago, rather they opt for quick and agile applications which cost them less money as well.

Microservices help in breaking the boundaries of large applications and build logically independent smaller systems inside the system. E.g. using Amazon AWS you can build a cloud application with minimum effort. It's a good example of what microservices can do.



Monolithic vs MicroServices Architecture

As you can see in the above diagram, each microservice has it's own business layer and database. By doing so, changes to one microservice do not impact others.

In general, **microservices communicate with each other using widely adopted lightweight protocols**, such as HTTP and REST, or messaging protocols, such as JMS or AMQP. In specific scenarios, they can go for more specialized protocols as well.

2. Principles of Microservices

Now let's examine the "must-have" principles of a microservice.

1. **Single responsibility principle**

   The single responsibility principle is one of the principles defined as part of the SOLID design pattern. It implies that a unit, either a class, a function, or a microservice, should have one and only one responsibility.

   At no point in time, one microservice should have more than one responsibility.

2. **Built around business capabilities**

helps in getting things done. A microservice shall never restrict itself from adopting appropriate technology stack or backend database storage which is most suitable for solving the business purpose.

This is often the constraint when we design monolithic applications where we try to solve multiple business solutions with some compromises in some areas. Microservices enable you to choose what's best for the problem at hand.

3. **You build it, you own it!**

Another important aspect of such design is related to responsibilities pre-and-post development. In a large organization, usually one team develops the app location, and after some knowledge transfer sessions it hand over the project to the maintenance team. In microservices, the team which builds the service – owns it, and is responsible for maintaining it in the future.

> You build it, you own it !!

This ownership brings developers into contact with the day-to-day operation of their software and they better understand how their built product is used by customers in the real world.

4. **Infrastructure Automation**

Preparing and building infrastructure for microservices is another very important need. **A service shall be independently deployable** and shall bundle all dependencies, including library dependencies, and even execution environments such as web servers and containers or virtual machines that abstract physical resources.

autonomy. While most SOA implementations provide service-level abstraction, microservices go further and abstract the realization and execution environment.

In traditional application developments, we build a WAR or an EAR, then deploy it into a JEE application server, such as with JBoss, WebLogic, WebSphere, and so on. We may deploy multiple applications into the same JEE container. In an ideal scenario, in the microservices approach, each microservice will be built as a fat Jar, embedding all dependencies and run as a standalone Java process.

5. **Design for Failure**

   A microservice shall be designed with failure cases in mind. What if the service fails, or go down for some time. These are very important questions and must be solved before actual coding starts – to clearly estimate **how service failures will affect the user experience**.

   Fail fast is another concept used to build fault-tolerant, resilient systems. This philosophy advocates systems that expect failures versus building systems that never fail. Since services can fail at any time, it's important to be able to detect the failures quickly and, if possible, automatically restore service.

   Microservice applications put a lot of **emphasis on real-time monitoring** of the application, checking both architectural elements (how many requests per second is the database getting) and business relevant metrics (such as how many orders per minute are received). Semantic monitoring can provide an early warning system of something going wrong that triggers development teams to follow up and investigate.

3. Benefits of Microservices

Microservices offer a number of *benefits over the traditional multi-tier, monolithic architectures*. Let's list down them:

architectures and technologies for each microservice (polyglot architecture). This gives the flexibility to design better-fit solutions in a more cost-effective way.

- As services are **fairly simple and smaller in size**, enterprises can afford to experiment with new processes, algorithms, business logic, and so on. It enables enterprises to do disruptive innovation by offering the ability to experiment and fail fast.
- Microservices enable to implement **selective scalability** i.e. each service could be independently scaled up or down and cost of scaling is comparatively less than monolithic approach.
- Microservices are **self-contained, independent deployment modules** enabling the substitution of one microservice with another similar microservice, when the second one is not performing as per our need. It helps in taking right buy-versus-build decisions which are often the challenge for many enterprises.
- Microservices help us **build systems that are organic in nature** (Organic systems are systems that grow laterally over a period of time by adding more and more functions to it). Because microservices are all about independently manageable services – it enables to add more and more services as the need arises with minimal impact on the existing services.
- Technology changes are one of the barriers in software development. With microservices, it is possible to **change or upgrade technology for each service individually** rather than upgrading an entire application.
- As microservices **package the service runtime environment along with the service itself**, this enables having multiple versions of the service to coexist in the same environment.
- And finally, microservices also enable **smaller, focused agile teams** for development. Teams will be organized based on the boundaries of microservices.

4. Conclusion

Often the true consequences of your architectural decisions are only evident several years after you made them.

In this article, I have only listed down some positives about microservices which have been seen in many organizations in my limited knowledge. A monolithic application, backed by strong design and brilliant coders, can also prove a good decision and the product can stay long enough to support the decision.

Similarily with microservices, poor design decisions will be proved costly. They may seem to be simplifying the components, but they may add complexity in communication between them and that is harder to control and manage.

Finally, **it's good design and skilled team is what will bring win for you**. A less skilled team will always create a poor system and it's very hard to tell if microservices reduce the mess in this case or make it worse.

I will suggest starting with monolithic application design, and when you feel that it is making the system complex – try microservices to check if they make the application less complex. In this way, you will have a more informed and better decision.

Happy Learning !!

Resources:

Martin Fowler on Microservices

Related Posts:

1. [Solved] java.lang.IllegalArgumentException: taglib definition not consistent with specification version
2. SOLID Principles
3. FIRST Principles for Writing Good Unit Tests

Leave a Reply

Join the discussion

6 COMMENTS Most Voted

Saurabh Singh
March 30, 2019 1:54 pm

After clicking on hyperlink “You build it, you own it” . came across collection of poems by Aron Atkins’s sister.

Thats very nice.

0 Reply

Pirish Gandey
Reply to Saurabh Singh December 31, 2019 3:32 am

POEM? That’s an article.

1 Reply

Srikanth
February 26, 2019 11:12 pm

understood the topic.
Can you please upload the source code with the explanation.

0 Reply

Anuj Kumar Verma

Hi Lokesh,

First of all, thank you for this valuable information.As far as I understand that microservices is an architecture like web services. But I don't understand how microservices can remove the constraints of interoperability?

Can you please help to understand the working of microservices at low level?

0 Reply

Himansu

October 3, 2016 12:29 pm

Hi Lokesh,

Good work

Can you also explain popular tech stack used for building microservices?

Regards
Himansu

0 Reply

> **Atif**
>
> Reply to Himansu February 2, 2017 8:08 pm
>
> Following Himansu's question
>
> 1 Reply

Search ...

About Us

HowToDoInJava provides tutorials and how-to guides on Java and related technologies.

It also shares the best practices, algorithms & solutions, and frequently asked interview questions.

Tutorial Series

- Regex
- Maven
- Logging
- TypeScript
- Python

Meta Links

- About Us
- Advertise
- Contact Us
- Privacy Policy

Our Blogs

REST API Tutorial