EP 23 - 10 Microservices Principles and Best Practices for Java Developers

Essential Microservice principles and best practice to make most of Microservice architecture without falling in traps and compromising maintenance.

JAVINPAUL AND SOMA
OCT 21, 2023 · PAID

3 2 Share

Hello folks, if you are a senior Java developer or a junior programmer working in Microservices, and don't have much idea how to create a production-grade Microservices the article is for you. I have been doing Microservices development in the last couple of years and it's been a tough journey as I have to learn most of things the hard way.

I made a big mistake ....

when I didn't invest in my learning by joining any Microservice training courses before jumping into development and due to that my learning was slow and I had to do a lot of Google searches even for simple stuff.

That's why it makes more sense to join a couple of courses and read books and articles before you start taking work on a new technology or architecture. Even if you don't learn everything you will get ideas about a lot of things which will save a lot of your time even if you have to search.

By the way, if you are new to Microservice architecture or just want to revise key Microservice concepts and looking for resources then here are a few online courses you can join:

1. **Master Microservices with Spring Boot and Spring Cloud** [**Udemy**]
2. **Building Scalable Java Microservices with Spring Boot** [**Coursera**]
3. **Developing Microservices with Spring Boot** [**Educative**]
4. **Master Microservices with Java, Spring, Docker, Kubernetes** [**Udemy**]
5. **Grokking Microservices Design Patterns** (**DesignGuru**)

This list includes both video and text-based courses as well as project-based courses for hands-on learning, you can join one or a couple of them to revise Microservices concepts. If you need more choices, you can see the below articles:

10 Essential Microservices Design and Development Principles for Java Developers

Here is a list of 10 Microservices design principles you can keep in mind and follow while doing Microservice development. They will help you a lot in the long run even after development and deployment.

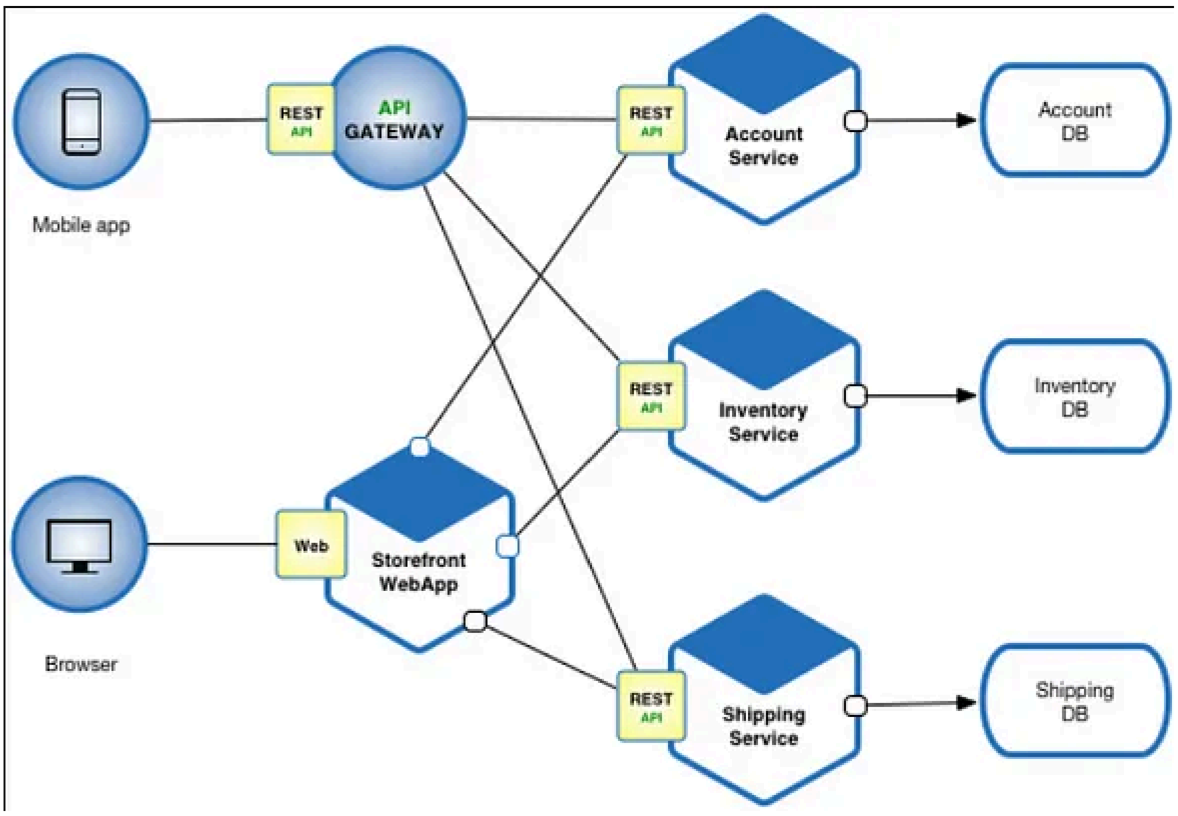
1. Single Responsibility Principle

This is not new to any Software developer, every one of us is familiar with this as part of learning the SOLID design principle but this also applies to Microservices.

As per SRP or Single Responsibility Principle, each Microservice should have a single, well-defined responsibility, and should only communicate with other microservices to accomplish tasks.

For example, one microservice could handle user authentication, while another handles payment processing. But you shouldn't create a Microservice that is doing both user authentication and Payment handling, that would be a violation of SRP.

You can see that in the following diagram, we have different services to handle different functionality like **Account Services**, **Inventory Services, and Shipping Services.**



2. Decentralized Data Management

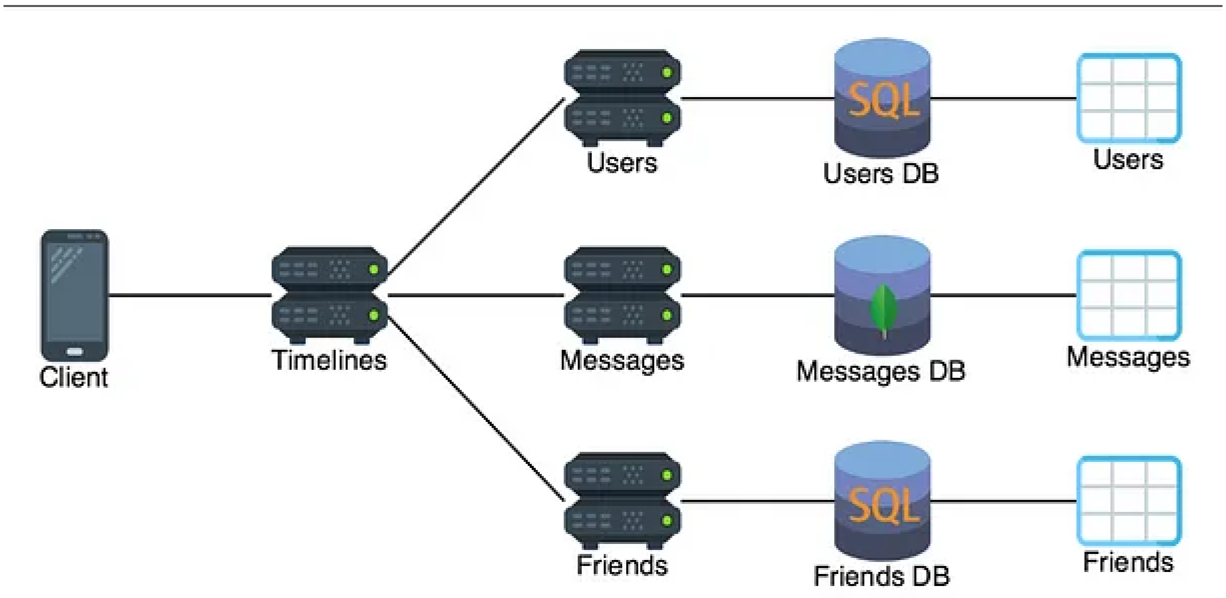
As per the Decentralized Data Management principle, each Microservice should manage its own data, without relying on other Microservice, to ensure scalability and reliability. For example, each Microservice could have its own database that it uses to store data.

Sharing a Database with other Microservices violates this principle and should be avoided as it will make it difficult to troubleshoot and can result in data inconsistency.

This design principle will help you better manage your database It is also a basis of Database per Microservice design pattern, an essential Microservice Design Principle.

If you are thinking about how could then another Microservice get access to the same data as is very much possible that another service does need it. Well, you should always create APIs for that as we going to see in the next Microservice design principle.

You can see this in the following Microservice architecture we have different databases for UserService, MessageService, and FriendService.



3. API-Driven Design

This one is my favorite Microservice design principle and it helps a lot during actually designing Microservices. As per this principle, Microservices should be designed around APIs, with each service exposing a well-defined set of APIs for communication with other services.

For example, a Microservice could expose an API for retrieving customer information, which other microservices could use to access that information.

This principle also goes along with the previous design principle which advocates that Microservices should not share databases. You must create APIs for other services which need access to the data and this will then drive your own Microservice design.

4. Statelessness

As per this principle, Microservices should be stateless, meaning that they should not maintain any client-specific state between requests.

For example, if a microservice handles a user's shopping cart, it should not store any information about the user's cart between requests but should retrieve the cart information from a database each time it processes a request.

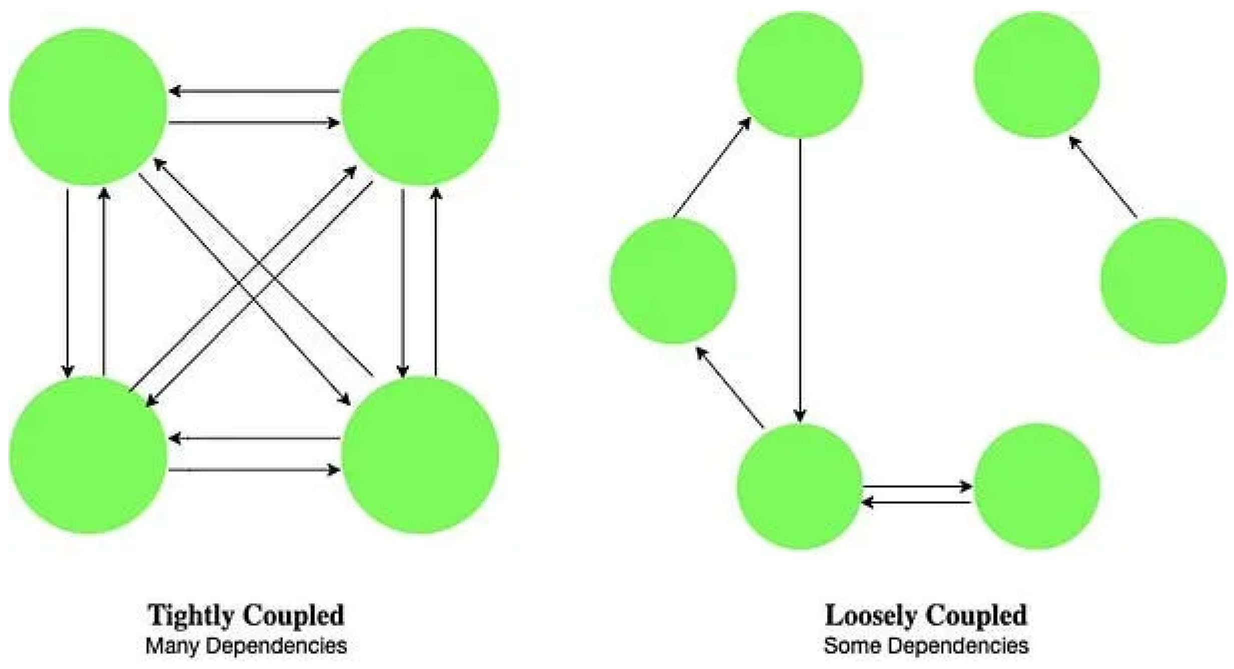
If I could give just one advice of from my years of experience in designing Java services then I would say to keep it stateless as long as possible. By not Managing the state in your service, you will avoid a lot of problems that come due to that and you can greatly benefit from things like Caching for performance improvement.

5. Loose Coupling

This is another Software design principle that also applies to Microservices. In Object-oriented design, we aim to lose a couple of our classes, packages, and modules and we can apply the same rule with Microservices.

As per this principle, Microservices should be loosely coupled, meaning that **they should not have a tight dependency on each other, to ensure scalability and ease of deployment.**

For example, if one microservice is down, the others should still be able to function normally. Here is a diagram which illustrates tightly coupled and loosely coupled Microservices:



6. Smart Endpoints and Dumb Pipes

As per this principle, the data-processing logic should be located in the Microservices themselves, rather than in a centralized hub, to ensure scalability and reliability. If you are wondering what endpoints and Pipes means here, endpoints are APIs or URLs while pipes are queues and database.

The pattern suggests that endpoints, such as user interfaces or APIs, should be designed to be smart, handling all presentation and business logic, *while pipes, such as queues or databases,* should be designed to be dumb, performing only simple data transfer and storage functions.

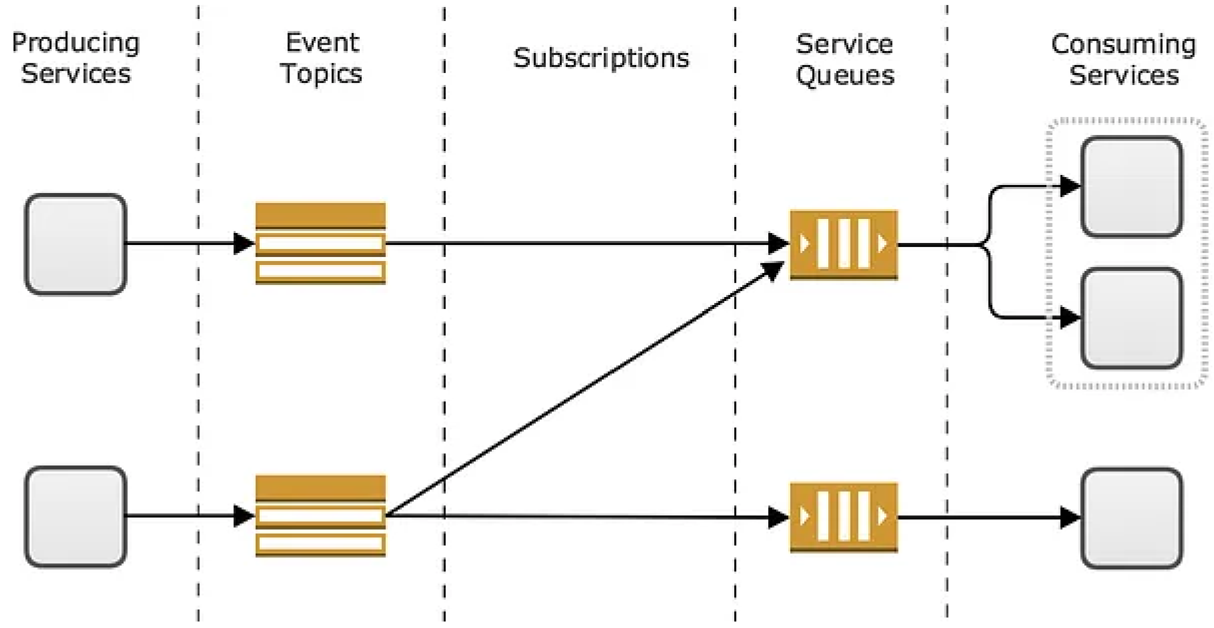
For example, a microservice could be responsible for processing customer orders, rather than having a centralized hub handle all order processing.

The key benefit of using this pattern is that it enables development teams to create applications with loosely coupled components that can be developed and deployed independently, allowing for better scalability and easier maintenance.

This pattern also helps to simplify the development process, as developers can focus on implementing specific features in their smart endpoints rather than worrying about the underlying communication and data storage systems.

However, one drawback of this pattern is that it can make debugging more difficult, as errors may occur in multiple components. Additionally, smart endpoints may require more resources and processing power, which can increase the cost of running the application.

Here is a nice diagram that illustrates this principle:



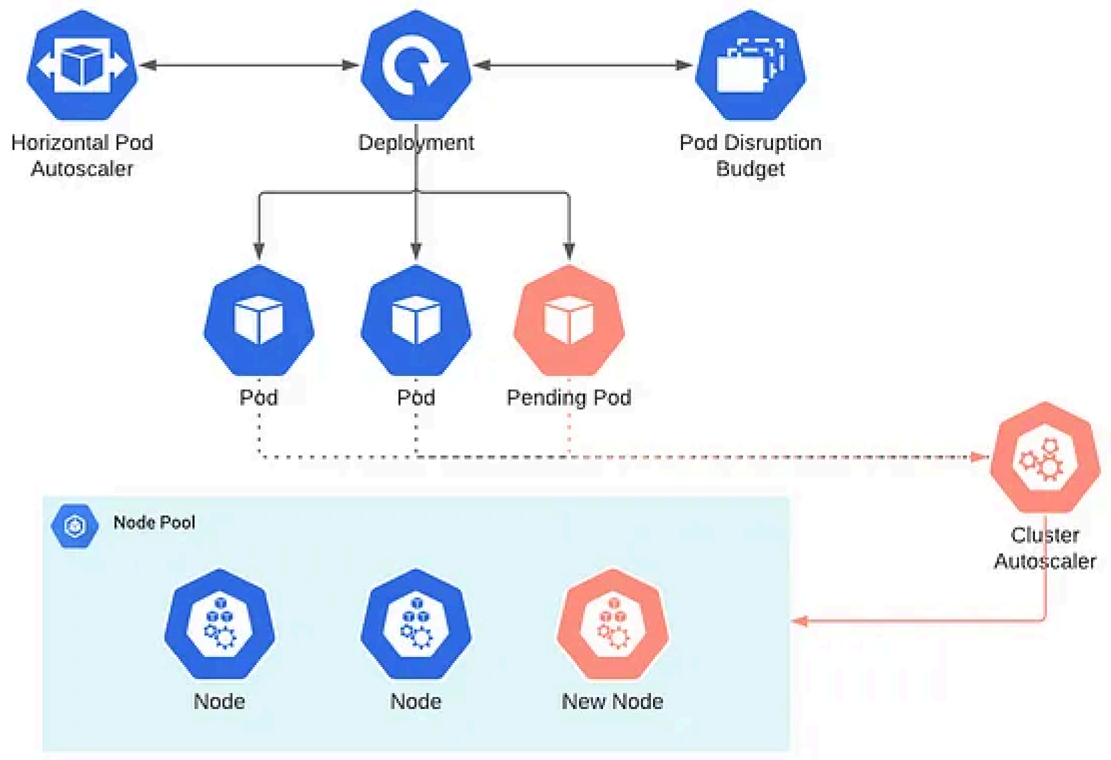
7. Auto-Scaling

When we think about Microservices, we think about Scaling and this is the principle that takes care of that. As per this best practice and principle, each Microservices should be designed to automatically scale up or down in response to changes in demand, to ensure that the application remains responsive and available.

For example, if the number of users accessing a microservice increases, that microservice could automatically spin up additional instances to handle the increased demand.

In the real world, tools like Kubernetes can automatically scale up and scale down by creating new instances of your Microservices and destroying them once they are no longer required.

If your Microservices is not designed for automatic scaling then it wouldn't take full benefit of Cloud and tools like Kubernetes, so you must ensure that they are designed for scaling. If you are wondering how to achieve that, then stay tuned, I will share more tips on how to ensure the scalability of your Microservices in the next article.



8. Monitoring and Logging

One of the major problems with working on a project that has hundreds of Microservices as debugging is really hard. It's hard to find the cause of why a request failed because logs are scattered.

You may see user authentication errors thinking that something is wrong with user credentials but it is because of timeout on the service that does that, you will only come to know the true reason by looking at logs of multiple services.

As per this principle. Microservices should have robust monitoring and logging mechanisms in place to help diagnose issues and track performance.

For example, each microservice could log information about its performance and usage, which could be used to identify and diagnose issues. This will help you surely you in the long run and day to day-to-day support work.

9. Continuously Deployment and Integration

As per this principle, Any Microservices should be continuously deployable, meaning that they should be updated frequently with small, incremental changes like bug fixes, small enhancements etc.

For example, a microservice could be updated to fix a bug or add a new feature, without affecting the rest of the application.

Continuous deployment is achieved through a combination of techniques such as automation of build and deployment processes, testing, and integration with other tools such as version control systems, issue tracking systems, and monitoring tools.

By automating the deployment process, teams can ensure that new changes are deployed quickly and consistently, with minimal human intervention.

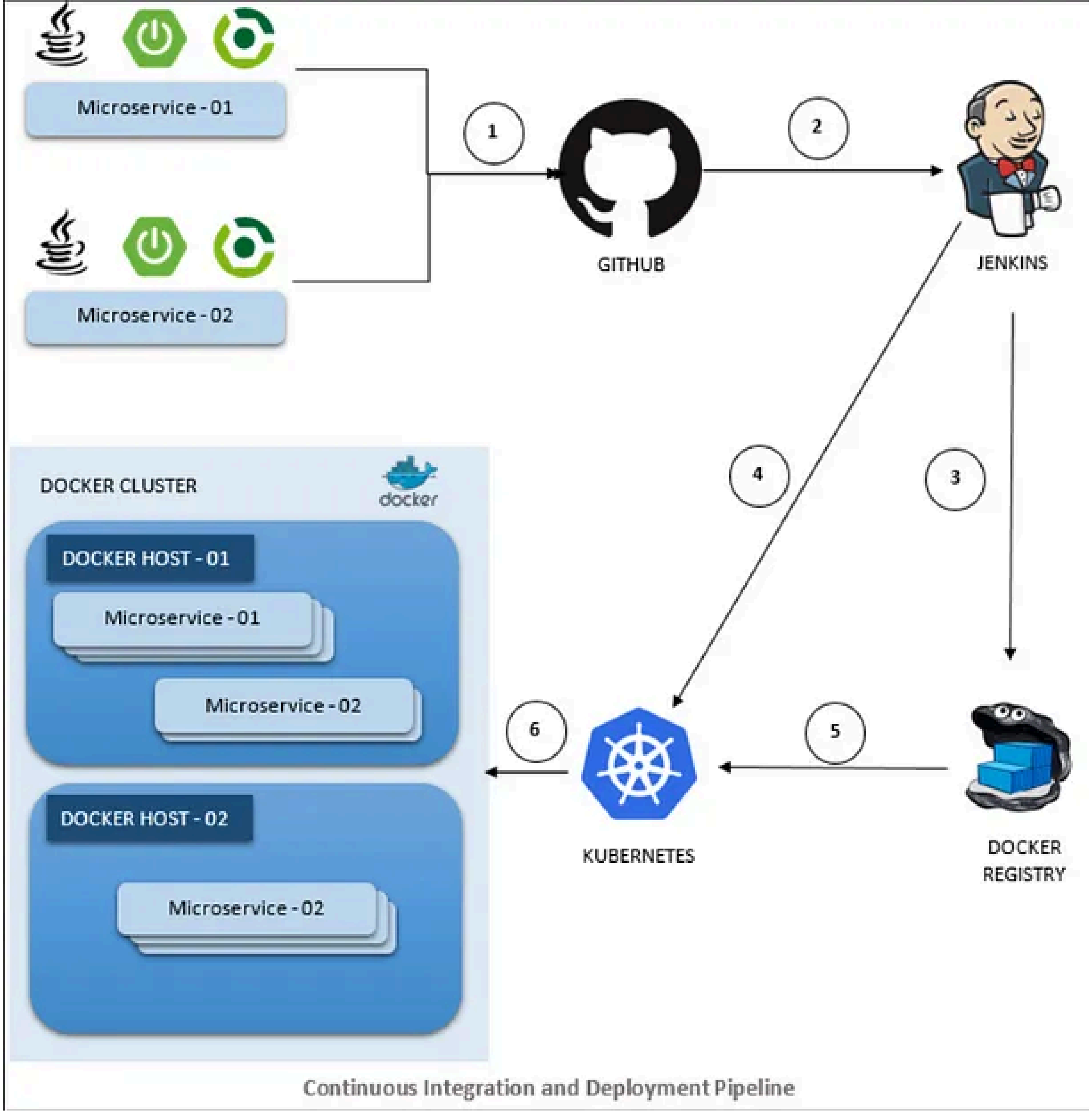
This practice also helps to reduce downtime and minimize the risk of errors, as new changes are thoroughly tested before they are deployed to production. Furthermore, it allows organizations to release new features and bug fixes more frequently, which can result in increased innovation and faster time to market.

10. Infrastructure Automation

This is something for DevOps but as a developer, we should also be familiar with this. As per this principle, the deployment and management of microservices should be automated, to ensure consistency and efficiency.

For example, you can use tools like Docker or Kubernetes to automate the deployment and scaling of Microservices. This is actually the standard practice now across the firms and it ensures that the deployment process is consistent and efficient.

You can create Jenkins pipelines for automatic deployment and you can also use tools like Terraform to create the environment automatically. Here is an example pipeline for automatic deployment of Microservices.



That's all about the **essential Microservices design principles every Java developer should know.** By following these design principles, you can build scalable and reliable microservices applications that can meet the demands of their customers.

They will not only help you to build robust and scalable Microservices but also save time while debugging, monitoring, troubleshooting, and maintaining Microservices, which is a major challenge in the ever-growing space of Microservices.

By the way, if you are new to Microservice architecture or just want to revise key Microservice concepts and looking for resources then here are a few online courses you can join:

1. Master Microservices with Spring Boot and Spring Cloud [**Udemy**]
2. Building Scalable Java Microservices with Spring Boot [**Coursera**]
3. Developing Microservices with Spring Boot [**Educative**]
4. Master Microservices with Java, Spring, Docker, Kubernetes [**Udemy**]
5. Grokking Microservices Design Patterns (**DesignGuru**)

This list includes both video and text-based courses as well as project-based courses for hands-on learning, you can join one or a couple of them to revise Microservices concepts. If you need more choices, you can see the below articles:

3 Likes · 2 Restacks

← Previous

Next →

A guest post by

Soma

Java and React Developer

Subscribe to Soma

Comments

Write a comment...