**MICROSERVICES**

**What is microservice?**Microservices is an architectural style used to build and organize software applications as a collection of small, independent, and loosely coupled services.

Instead of building a single large and monolithic application, microservices break the application into smaller, self-contained components, each responsible for a specific business capability.  
In simplest terms, think of microservices as building a big puzzle by assembling many small pieces together. Each puzzle piece (microservice) does its own unique job, and when you put all the pieces together, you create the complete picture (the whole application).

**Characteristics of microservice**

**Independence**: Each microservice is independent and can be developed, deployed, and scaled(scale up/down create/delete multiple instances for load balancing) separately. Changes to one microservice don't impact others, allowing teams to work autonomously.

**Loose Coupling**: Microservices communicate with each other through well-defined interfaces (usually APIs). This loose coupling means they can be replaced, updated, or scaled independently without affecting other parts of the application.

**Single Responsibility**: Each microservice has a single and specific responsibility or business capability. For example, one microservice might handle user authentication, another might process payments, and so on.

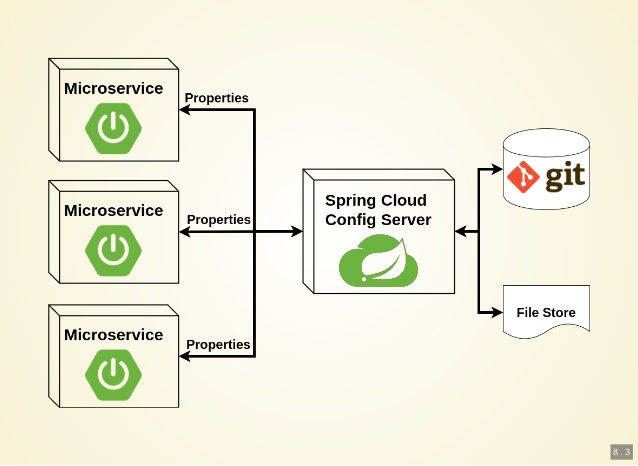
**Technology Diversity**: Microservices allow different technologies and programming languages to be used for different services, based on what's most suitable for each task.

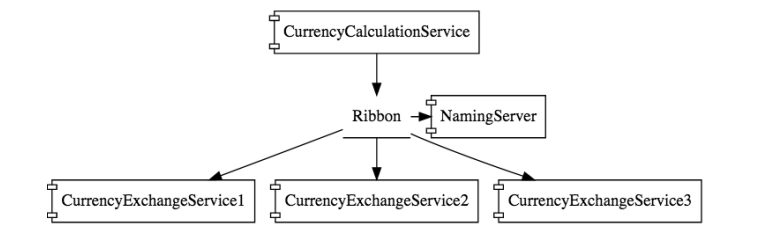
**Scalability**: Since each microservice can be scaled independently, it allows for more efficient use of resources and better handling of varying loads on different parts of the application.

**Resilience**: If one microservice fails, it doesn't bring down the entire application. Other microservices can continue to function independently.

**Challenges in microservices?  
  
BOUNDED CONTEXT:** We don’t know the exact boundaries of a microservice. What all business logic should be included. **CONFIGURATION MANAGEMENT:** 100s of microservices and lot of envs so there is tons of configurations req. Hwo to manage them. How to simply that.  **DYNAMIC SCALE UP AND SCALE DOWN:** dynamically distribute load on different instances **VISIBLITY:** we need to see whats happening behind the scenes with microservices. A simple req may involve 10 microservices. How to determine which microservice was the cause of the defect. How do I know if all the microservices are up and running? **PACK OF CARDS:** How do I prevent one microservice being down, taking down the entire application? How do I build fault tolerance into my microservices?

**INTRODUCTION TO SPRING CLOUD**Spring cloud provides solution to the above discussed problems  
**CONFIGURATION MANAGEMENT:** Spring cloud config server provides an approach where you can store all the diff config of all the diff envs of all the microservices in a git repo



**DYNAMIC SCALE UP AND SCALE DOWN:** dynamically distribute load on different instances  
In the above eg we can see that there are 3 instances for the currencyExchangeService for load balancing. We use a Naming server which provides dynamic relationship between the CurrencyExchangeService and the instances. So when currencyExchangeService asks for the instances, NamingServer provides url for all the instances. We will use Ribbon for client side load balancing. Ribbon will make sure that the load is equally distributed among the existing instances that it gets from the namingServer. We will also use Feing as a mehcanism to write simple RESTful clients

**VISIBLITY:** we need to see whats happening behind the scenes with microservices. A simple req may involve 10 microservices. How to determine which microservice was the cause of the defect. How do I know if all the microservices are up and running?Zipkin Distributed Tracing: We will use Spring cloud Sleuth to assign a id to a request across multiple components. And we will use Zipkin Distributed Tracing to trace the req across the multilple components  
Netflix API gateway. Some microservices such as logging/sercurity/analytics have common features. You don’t need to execute these common features in all the microservices. API gateway provide solution to this problem  
  
**FAULT TOLERANCE:** We will use Hystrix to perform fault tolerance. If the server is down, Hystrix helps use to configure a default response

MIRCROSERVICES V2