# Spring Boot Microservices Tutorial

## What are Microservices?

Microservices are a software architectural style in which a large application is built as a collection of small, independent services that communicate with each other over a network.

Each service is a self-contained unit of functionality that can be developed, tested, and deployed independently of the other services. This allows for more flexibility and scalability than a monolithic architecture, where all the functionality is contained in a single, large codebase.

Microservices can be written in different programming languages and use different technologies, as long as they can communicate with each other through a common API.

They are designed to be loosely coupled, meaning that changes to one service should not affect the other services. This makes it easier to update, maintain, and scale the application. Microservices architecture is best suited for large and complex applications that need to handle a high volume of traffic and need to be scaled horizontally.

## Key Components of a Microservices Architecture

Key components of a microservices architecture include:

1. **Core Services**: Each service is a self-contained unit of functionality that can be developed, tested, and deployed independently of the other services.
2. **Service registry**: A service registry is a database of all the services in the system, along with their locations and capabilities. It allows services to discover and communicate with each other.
3. **API Gateway:** An API gateway is a single entry point for all incoming requests to the microservices. It acts as a reverse proxy, routing requests to the appropriate service and handling tasks such as authentication and rate limiting.

4. **Message bus:** A message bus is a messaging system that allows services to communicate asynchronously with each other. This can be done through protocols like HTTP, RabbitMQ, or Kafka.
5. **Monitoring and logging:** Monitoring and logging are necessary to track the health of the services and troubleshoot problems.
6. **Service discovery and load balancing:** This component is responsible for discovering service instances and directing traffic to the appropriate service instances based on load and availability.
7. **Continuous integration and continuous deployment (CI/CD):** To make the development and deployment process of microservices as smooth as possible, it is recommended to use a tool such as Jenkins, TravisCI, or CircleCI to automate the process of building, testing, and deploying microservices.