**Spring Cloud**

Spring Cloud is a set of tools that helps developers build **distributed systems and microservices**.  
It provides solutions to common challenges like:

* **Service Discovery** → How do services find each other?
* **Load Balancing** → How to distribute requests across instances?
* **API Gateway / Routing** → How to manage all requests in one entry point?
* **Configuration Management** → How to manage configs for multiple services?
* **Fault Tolerance** → How to handle failures gracefully?
* **Distributed Tracing** → How to track requests across microservices?

**2. Why Use Spring Cloud?**

Building microservices introduces **complexities** that don’t exist in monolithic apps.  
Spring Cloud helps with:

* **Dynamic service registration** (services don’t need hardcoded IPs).
* **Centralized configuration** (no duplication of configs).
* **Secure and efficient communication** between services.
* **Resilient systems** (fallback, retries, circuit breakers).
* **Monitoring and tracing** of distributed services.

**3. Key Components of Spring Cloud**

Here are the main building blocks:

1. **Spring Cloud Netflix Eureka**
   * Service registry and discovery.
   * Services register themselves with Eureka.
   * Clients use Eureka to discover other services.
2. **Spring Cloud Gateway**
   * Acts as a **reverse proxy** and single entry point for clients.
   * Handles **routing, filtering, load balancing, and security**.
   * Example: Client → API Gateway → Book Service.
3. **Spring Cloud OpenFeign**
   * Simplifies service-to-service communication.
   * Instead of writing RESTtemplate code, you define an **interface**.
   * Feign automatically generates REST clients.
4. **Spring Cloud Config**
   * Provides **centralized configuration management** for all microservices.
   * Can store configs in Git or local files.
5. **Resilience4j / Hystrix (Circuit Breaker)**
   * Provides **fault tolerance**.
   * Protects systems when a service fails or responds slowly.
   * Example: Returns **fallback response** instead of crashing the system.
6. **Spring Cloud Sleuth + Zipkin**
   * Distributed tracing tools.
   * Help monitor and debug requests across multiple microservices.

Steps:

1. Setting up Eureka Server

* Dependencies: **spring-cloud-starter-netflix-eureka-server,**

**spring-cloud-starter**

* Annotations used: **@EnableEurekaServer**
* In application.properties/ app.yml file these 2 lines are must:
  + server.port = 8761(recommended)
  + spring.application.name = eurekaserver
  + **eureka.client.register-with-eureka = false**
  + **eureka.client.fetch-registery = false**

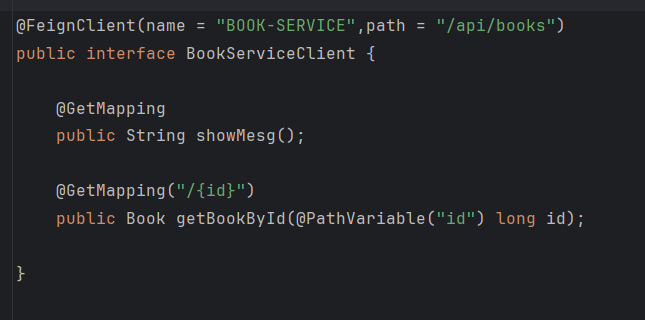
1. Setting up Client’s for registering to that server so that they can talk with each other thorough our server.

* spring-cloud-starter-netflix-eureka-client <- dependencies
* **@EnableDiscoveryClient**
* In application.properties we should mention about which server will this client gets register and the name of this client
  + server.port = 9090(….assume)
  + eureka.client.serviceUrl.defaultZone = <http://localhost:8761/eureka/>
  + spring.application.name = bookservice ..
  + **And** our normal MVC structure follows here (service class,…....)

1. To access data from other services through our server we can do it by 2 ways:
2. By using RestClient (follow then normal steps and we need to manually configure everything….. explore more).
3. By using FeignClient(Everything is automated)

Using FeignClient to communicate with other service

* Dependencies: **spring-cloud-starter-openfeign** (along with others)
* **@EnableFeignClients -> used on Main class along with @SprinbBootApp…**
* **Then we need to create an interface and write all the method definitions which are to be used in our service and annotate that class with @FeginClient: Each @FeignClient is tied to a service name (the name registered in Eureka or the URL)**



Clear names (reflect business meaning)

Correct annotations (PathVariable, RequestParam, RequestBody)

Make sure the **path matches exactly** the microservice’s controller endpoint.

Always specify parameters correctly:

DTOs for clarity

**One interface per microservice**

* Then create a service class and call this interface methods by refrencing by their name and use them in controller as you wish:
* private final BookServiceClient feignClient;
* public String showMesg() {
* return feignClient.showMesg();
* }
* public Book getBookById(long id) {
* return feignClient.getBookById(id);
* }

Feign, a declarative HTTP client that "simplifies implementation" and maintenance efforts. It means you write an interface, and Spring Cloud + OpenFeign "generate the implementation that makes actual HTTP calls". We are telling WHAT TO DO and not HOW TO DO..

API GATEWAY

* An API Gateway is like the **main entrance to your entire microservices system**.
* Instead of clients (mobile apps, web apps, etc.) calling each microservice directly, they call the gateway, and the gateway forwards the request to the correct microservice.

👉 Think of it like a reception desk in a big office:

* Visitors don’t go to each department directly.
* They go to reception, and the receptionist directs them to the right department.

**Without a gateway:**

* The client has to know addresses of all microservices (user-service, order-service, payment-service).
* The client has to handle load balancing, security, retries, etc.
* If microservice URLs change, the client also must change.

**With a gateway:**

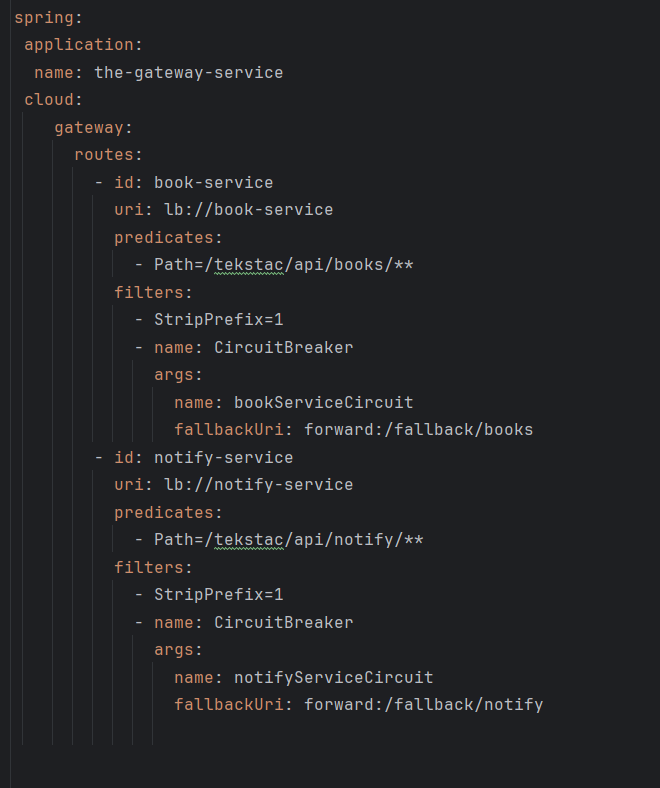
* Client only talks to one endpoint (the gateway).
* Gateway takes care of routing, security, load balancing, logging, monitoring, etc.
* Microservices can scale, move, or change without affecting clients.

**Main Functions:**

* 1. Routing
  2. Load Balancing
  3. Security
  4. Centralized Logging & Monitoring
  5. Rate Limiting
  6. Request Transformation
  7. Fallback / Resilience

We should register this as a eureka client in the same server.

server:  
 port: 9999  
   
eureka:  
 client:  
 serviceUrl:  
 defaultZone: http://localhost:8761/eureka/

****

**spring.application.name**

* This is **the name of the gateway itself** (as registered in Eureka, if discovery is enabled).
* Convention:
  + Use kebab-case (lowercase, words separated by hyphens).
  + Include -gateway or -service suffix.
* Example:  
  ✅ the-gateway-service  
  ✅ api-gateway  
  ✅ order-gateway

👉 Avoid spaces, camelCase, or uppercase.

**2. routes.id**

* This is a **unique identifier** for the route (used internally for monitoring/logging).
* Convention:
  + Use **service name in kebab-case** (match the microservice you are routing to).
  + Keep it short, meaningful, and consistent.
* Example:  
  ✅ book-service  
  ✅ notify-service  
  ✅ order-service

👉 Don’t write vague IDs like route1, test-route, etc.

**3. uri**

* This points to the target microservice.
* Convention:
  + Always use **lb://SERVICE-NAME** if using Eureka/Discovery.
  + SERVICE-NAME should match spring.application.name of that service.
  + Use uppercase for the service name when referencing inside lb:// (common convention).

Example:

uri: lb://BOOK-SERVICE

uri: lb://NOTIFY-SERVICE

**4. predicates.Path**

* This defines the **incoming request path pattern**.
* Convention:
  + Start with a common prefix for all services (/api/ or /tekstac/api/ etc.).
  + Use **plural nouns** for resource paths (REST best practice).
  + Use lowercase and hyphens if multiple words.

Examples:  
✅ /api/books/\*\*  
✅ /api/notify/\*\*  
✅ /api/order-items/\*\*

avoid mixing plural/singular (/book/\*\* vs /books/\*\*). Always prefer plural.

**🔹 5. filters**

Filters modify the request before sending to the service.

**a) StripPrefix**

* Convention: Always document clearly **how many path parts are stripped**.
* Example:  
  - StripPrefix=1 → Removes /tekstac before forwarding.
* ✅ Keep consistent across services. If you strip 1 for book-service, strip 1 for others too, unless absolutely needed.

**CircuitBreaker**

* Convention:
  + name: use the **service name + Circuit**.  
    ✅ bookServiceCircuit  
    ✅ notifyServiceCircuit
  + fallbackUri: point to a consistent fallback endpoint under /fallback/{service}.  
    ✅ /fallback/books  
    ✅ /fallback/notify

