Les microservices

Projet en un bloc = monolithique
Projet multi module = microservices, structures actuels

Permet de rendre l'appli scalable = capable de s'étendre sur plusieurs ressources

problèmes des app monolithiques :

- adaptabiliité
- réactivité
- robustesse
- problème de scalabilité (dévolution du code, une modification entraine la modification de tout le projet et de refaire tous les tests etc..)

possibilité d'utiliser un langage différent par microservices

archi microservice :
archi ou on décide de découpé le programme en petits programmes. Chaque programme aura ses propres couches et db. Chaque microservice est un micro projet autonome. Moyen de découpé une problématique en plusieurs petits bloques.

Un micro servcie est une partie du code qui est autonome et qui peut interagir avec les autres microservices. une méthode qui consiste à découper les demandes des clients en plusieurs petites fonctionnalités

Avantages :

- chaque microservice peut être dev dans un langage différents
- couplage faible puisque chaque microservice est séparé physiquement dans des projets différents
- facilité des test et des déploiements

**4 types de micro services principaux :**

micro services métiers = micro services ayant pour bu de répondre aux exigences du client

Servcie d'enregristrement / registration service = servcie obligatoir, endoit ou tous les services sont enregistrés afin qu'ils puissent être utilisable

Service de configuration/ configuration service = selon les technologies utilisés, ce service peut être utilisé, moyen de centraliser la configuration de chaques microservices

Service Gateway/ Service Proxy = microservice servant de point d'entré au client. un seul point d'entré.

Services optionnels et particuliers :

Monitoring service (surveillences des microservices) et external service (intégration des services externes, ex: rapideapi.com)

Chaque micro service est déterminé par

- son nom
- son adresse ip
- son numéro de port

Chaque micro service aura une dépendance associé pour sa fonctionnalité (gateway, configuration, …)

serice métier : spring boot java avec springdata ou springsatarest
registration: eureka
proxy/ gateway : spring cloud
externe : rapideApi
monitoring : Hystrix

orchestration : mise en oeuvre des microservices entres eux et fonctionne dans un ordre précis.

Eureka

- rajouter l'anotation dans la classe main
- configurer le fichier application.properties

```
server.port=8761
# on met a false car pas besoin de s'enregistrer, il se connait lui même
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

Service métier :

- définition de la db

```
# Configuration pour le bdd
spring.datasource.url=jdbc:mysql://localhost:3306/customer-service?serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.jpa.database-platform = org.hibernate.dialect.MySQL5Dialect
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto = create
```

- configuration du microservice

```
# Configuration pour le microservice
spring.application.name = customer-service
server.port = 8081
spring.cloud.discovery.enabled = true
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
```

Lorsqu'un micro service est saturé, la gateway dirige la demande en fonction de la saturation des mêmes microservices.
Pour lancer plusieurs fois le même service, il faut définir un nouveau:

- commenter #server.port = 8081 dans application.properties

run as> run configuration> arguments> prgram arguments :
--server.port=82

gateway

- conf du fichier application.properties

```
spring.application.name = gateway-service
server.port = 8888
spring.cloud.discovery.enabled = true
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
```

pour la gestions des différentes routes :
static (en cas de changement de port, le modifier manuellement) :

- yml dans ressources (application.yml)

```
# configuration statique par fichier de configuration
spring:
  cloud:
    gateway:
      routes:
        - id : r1
          uri : http://localhost:8081
          predicates:
            - Path= /customers/**
```

- code

```
// conf static sans load balancing donc sans repartition d'instance
@Bean
        RouteLocator staticRoutes(RouteLocatorBuilder builder) {
                return builder.routes()
                                .route(r ->
r.path("/customers/**").uri("http://localhost:8081/").id("r1"))
                                .route(r ->
r.path("/products/**").uri("http://localhost:8082/").id("r2"))
                                .build();
}

        // Deuxième méthode conf static
@Bean
        RouteLocator staticRoutes(RouteLocatorBuilder builder) {
                return builder.routes()
```

```
                .route(r ->
r.path("/customers/**").uri("lb://CUSTOMER-SERVICE").id("r1"))
                .route(r ->
r.path("/products/**").uri("lb://INVENTORY-SERVICE").id("r2"))
                .route(r ->
r.path("/bills/**").uri("lb://BILLING-SERVICE").id("r3"))
                .route(r ->
r.path("/productItems/**").uri("lb://BILLING-SERVICE").id("r4"))
                .build();
        }
        // version dynamique
@Bean
        DiscoveryClientRouteDefinitionLocator
dynamicRoutes(ReactiveDiscoveryClient rdc,
                DiscoveryLocatorProperties ldp) {
            return new DiscoveryClientRouteDefinitionLocator(rdc,ldp);
        }
```

+ rajouter le nom du service dans l'url : ex http://localhost:8888/LOCATION-SERVICE/locations/getAll

ordre de lancement :

- eureka
- micro service metier (d'abord les micro services indépendants)
- gateway

Dependance des micros services :

- services métiers
  - spring boot dev tools
  - spring boot actuator
  - spring data jpa
  - mysql server
  - eureka discovery client
  - spring web
- gateway
  - spring boot actuator
  - hystrix [maintenance]
  - erekar discovery client
  - gateway
- eureka
  - eureka server
- hystrix
  - dans la main @EnableHystrixDashboard
  - dans application.properties server.port = 9999

- dépendance hystrix dashboard
- dans application.properties des microservice à monitorer :
  - management.endpoints.web.exposure.include=hystrix.stream
  - hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds=2000
  - @EnableHystrix dans le main
  - http://localhost:8888/actuator/hystrix.stream permet de check si ça fonctionne dans le microservcie gateway par exemple (port 8888)
- http://localhost:9999/hystrix => ajouter l'adresse du service monitoré ex: http://localhost:8888/actuator/hystrix.stream

Méthode :

- servcies métiers:

1- créer un micro service inventory-service
1bis- configurer le fichier application.properties
2- domaine créer une classe Produit (id, name, price, description)
3- service : addProduit(Produit p), List<Produit> findAll(), Produit getProduit(Long id)
4- écrire le controller correspondant
5- configurer le microservice et le déployer dans le microservice annuaire
6- au niveau du microservice getway, rajouter le chemin vers ce nouveau micro service
7- tester depuis le client qu'on peut bien atteindre le micro service

Service métier avec des relations :

- openFeign permet de rechercher une structure d'une classe dans un autre servcie
- **@Transient** permet de ne pas enregistrer en db (l'inverse de @Persist)
- les dependances :
  - spring boot dev tools
  - spring boot actuator
  - spring data jpa
  - mysql server
  - eureka discovery client
  - spring web
  - OpenFeign
  - Spring HATEOAS
- **@EnableFeignClients** dans la class main cette anotation permet de rechercher les informations dans les autres microservices
- Création des interfaces dans la couches service avec l'anotation
  **@FeignClient(name="NOM-MICROSERVICE")**
  Dans l'interface :
  **@GetMapping("/customers/getAll/{id}")**
  **public Customer findCustomerById(@PathVariable(name="id")Long id);**
  On créé une classe Customer avec uniquement les Customer pour avoir la structure.

- **@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)** : permet d'éviter de faire une boucle infinie. En effet une classe A a une collection d'objet B. A OneToMany de B et B ManyeToONe de A.