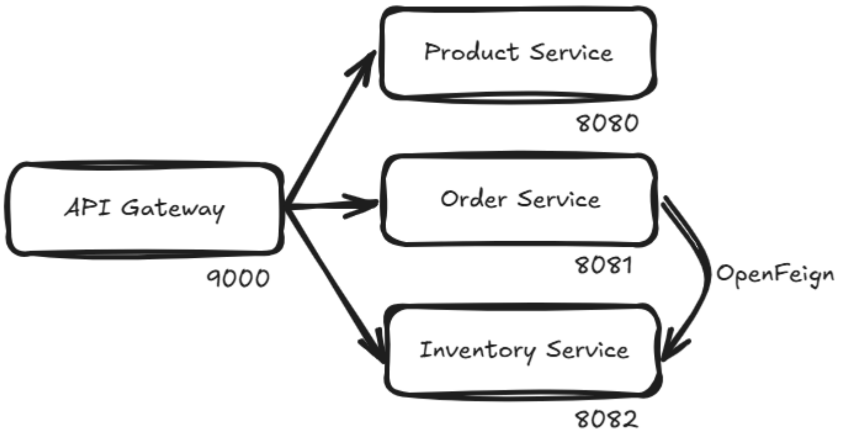
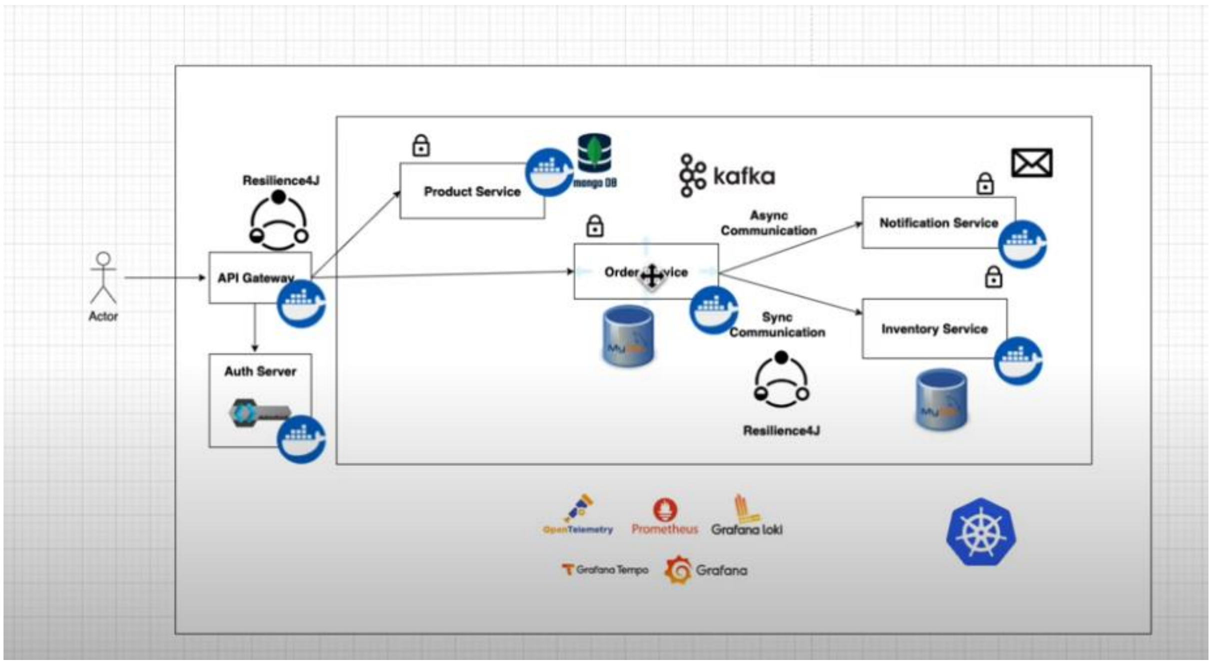
Microservices





OpenFeignClient

Communication between services – OrderService and InventoryService

```
@FeignClient(value = "inventory", url = "http://localhost:8082")
public interface InventoryClient {

    @RequestMapping(method = RequestMethod.GET, value = "/api/inventory")
    boolean isInStock(@RequestParam String skuCode, @RequestParam Integer quantity);
}
```

Now in order to use this, we can directly inject this into our @Service class and use the methods in here.

Here the methods in the open feign client class are similar to the methods in the JpaRepository where data JPA takes care of the underlying implementation, similarly here also openfeign takes care of the underlying implementation which is nothing but fetching the actual implementation of the method from the referenced microservice (inventory service).

Api-Gateway

Gateway dependency

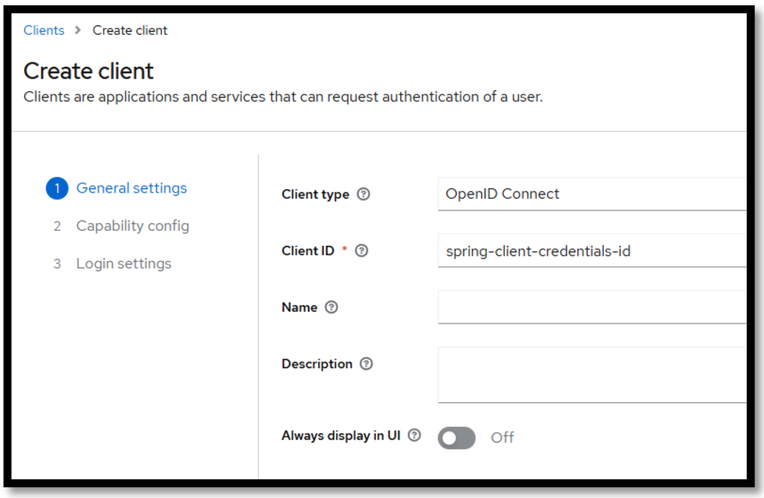
Create a Routes configuration class :

```
@Configuration
public class Routes {

    @Value("${product.service.url}")
    private String productServiceUrl;
    @Value("${order.service.url}")
    private String orderServiceUrl;
    @Value("${inventory.service.url}")
    private String inventoryServiceUrl;

    @Bean
    public RouterFunction<ServerResponse> productServiceRoute() {
        return GatewayRouterFunctions.route("product_service")
                .route(RequestPredicates.path("/api/product"),
HandlerFunctions.http(productServiceUrl))
                .build();
    }

    @Bean
    public RouterFunction<ServerResponse> orderServiceRoute() {
        return GatewayRouterFunctions.route("order_service")
                .route(RequestPredicates.path("/api/order"),
HandlerFunctions.http(orderServiceUrl))
                .build();
    }

    @Bean
    public RouterFunction<ServerResponse> inventoryServiceRoute() {
        return GatewayRouterFunctions.route("inventory_service")
                .route(RequestPredicates.path("/api/inventory"),
HandlerFunctions.http(inventoryServiceUrl))
                .build();
    }

}
```

Keycloak

First of all needs two containers, one is the keycloak itself and the other is the keycloak sql, means a mysql to maintain all the clients and realms.

1. Create a realm
2. Create a client.


Clients > Create client

Create client

Clients are applications and services that can request authentication of a user.

1 General settings
2 Capability config
3 Login settings

Client type: OpenID Connect

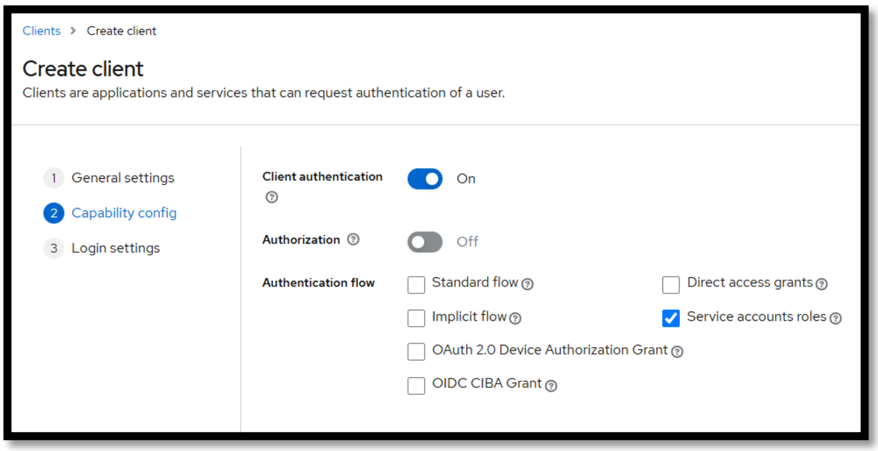
Client ID *: spring-client-credentials-id

Name:

Description:

Always display in UI: Off

3.


Clients > Create client

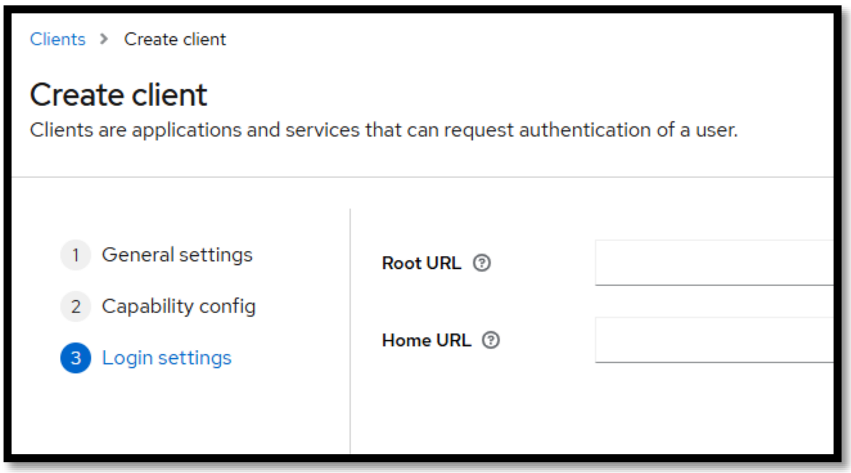
Create client

Clients are applications and services that can request authentication of a user.

1 General settings
2 Capability config
3 Login settings

Client authentication	On
Authorization	Off
Authentication flow	☐ Standard flow ☐ Direct access grants ☐ Implicit flow ✓ Service accounts roles ☐ OAuth 2.0 Device Authorization Grant ☐ OIDC CIBA Grant

4. Leave this as it is.


Clients > Create client

Create client

Clients are applications and services that can request authentication of a user.

1 General settings
2 Capability config
3 Login settings

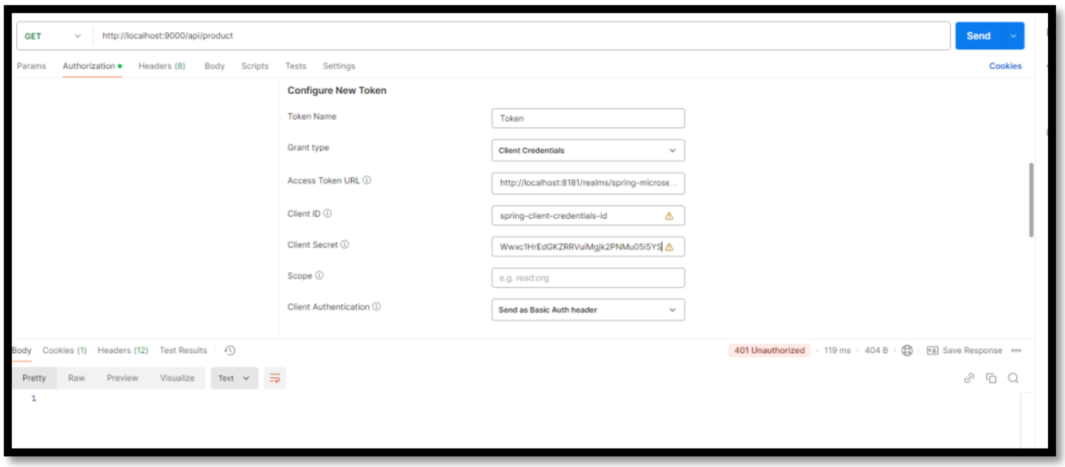
Root URL	
Home URL	

We can get our necessary things from **openid-configuration** :

And add this to application.properties of api-gateway service :

```
spring.security.oauth2.resourceserver.jwt.issuer-
uri=http://localhost:8181/realms/spring-microservices-security-realm
```

Now we access an endpoint through the api-gateway :


GET http://localhost:9000/api/product

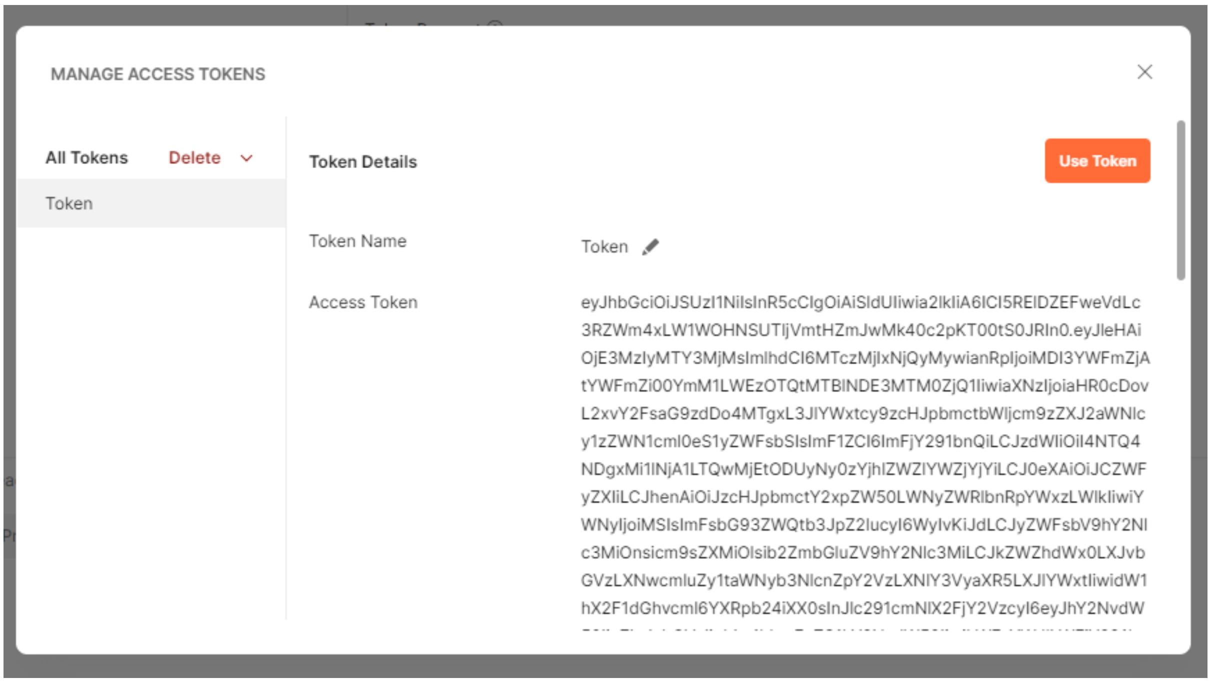
Configure New Token

Token Name	Token
Grant type	Client Credentials
Access Token URL	http://localhost:8181/realms/spring-microse...
Client ID	spring-client-credentials-id
Client Secret	Wwxc1HrEdGKZRRVuiMgjk2PNMu05i5YS
Scope	e.g. read:org
Client Authentication	Send as Basic Auth header

401 Unauthorized · 119 ms · 404 B

```
1
```

We will get :


MANAGE ACCESS TOKENS

All Tokens Delete

Token

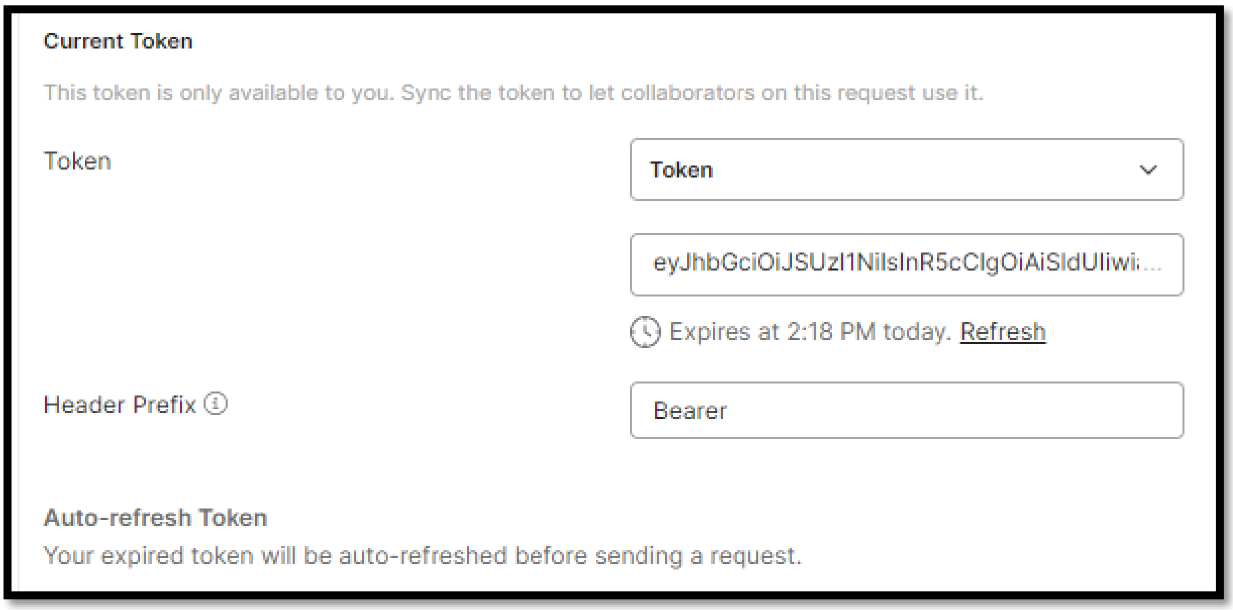
Token Details

Token Name: Token

Access Token:

```
eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICI5REIDZEFweVdLc
3RZWm4xLW1WOHNSUTljVmtHZmJwMk40c2pKT00tS0JRIn0.eyJleHAi
OjE3MzIyMTY3MjMsImlhdCI6MTczMjIxNjQyMywianRpIjoiMDI3YWFmZjA
tYWFmZi00YmM1LWEzOTQtMTBINDE3MTM0ZjQ1IiwiaXNzIjoiaHR0cDov
L2xvY2FsaG9zdDo4MTgxL3JlYWxtcy9zcHJpbmctbWljcm9zZXJ2aWNlc
y1zZWN1cml0eS1yZWFsbSIsImF1ZCI6ImFjY291bnQiLCJzdWIiOiI4NTQ4
NDgxMi1lNjA1LTQwMjEtODUyNy0zYjhlZWZlYWZjYjYiLCJ0eXAiOiJCZWF
yZXIiLCJhenAiOiJzcHJpbmctY2xpZW50LWNyZWRlbnRpYWxzLWlkIiwiY
WNyIjoiMSIsImFsbG93ZWQtb3JpZ2lucyI6WyIvKiJdLCJyZWFsbV9hY2Nl
c3MiOnsicm9sZXMiOlsib2ZmbGluZV9hY2Nlc3MiLCJkZWZhdWx0LXJvb
GVzLXNwcmluZy1taWNyb3NlcnZpY2VzLXNlY3VyaXR5LXJlYWxtIiwidW1
hX2F1dGhvcml6YXRpb24iXX0sInJlc291cmNlX2FjY2VzcyI6eyJhY2NvdW
```

When we click on ‘Use Token’, the token will be automatically added as a bearer token.


Current Token

This token is only available to you. Sync the token to let collaborators on this request use it.

Token: Token

eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwi...

Expires at 2:18 PM today. Refresh

Header Prefix: Bearer

Auto-refresh Token

Your expired token will be auto-refreshed before sending a request.

Swagger for Documentation

OpenAPI Specification – Swagger – provides tools and libraries

springdoc-openapi

https://springdoc.org/#getting-started

2 dependencies to be added – webmvc openapi and ui

```
springdoc.swagger-ui.path=/swagger-ui.html
```

Aggregating swagger routes in api-gateway :

1st step :

Add these to application.properties of api-gateway :

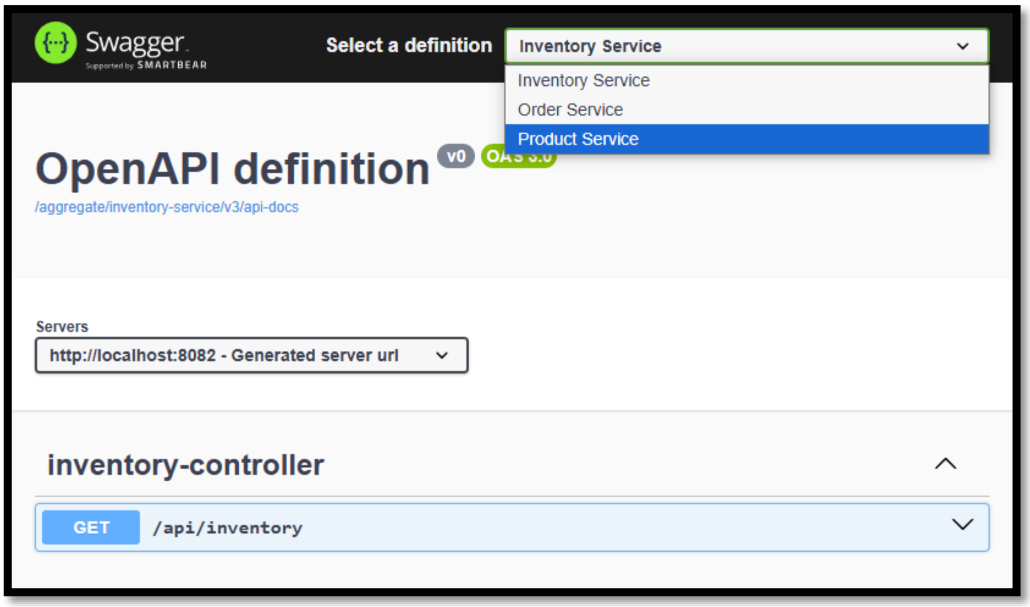
```
#Swagger
springdoc.swagger-ui.path=/swagger-ui.html
springdoc.api-docs.path=/api-docs
springdoc.swagger-ui.urls[0].name=Product Service
springdoc.swagger-ui.urls[0].url=/aggregate/product-service/v3/api-docs
springdoc.swagger-ui.urls[1].name=Order Service
springdoc.swagger-ui.urls[1].url=/aggregate/order-service/v3/api-docs
springdoc.swagger-ui.urls[2].name=Inventory Service
springdoc.swagger-ui.urls[2].url=/aggregate/inventory-service/v3/api-docs
```

Add these to Routes class :

```
@Bean
public RouterFunction<ServerResponse> productServiceSwaggerRoute(){
    return GatewayRouterFunctions.route("product_service_swagger")
            .route(RequestPredicates.path("/aggregate/product-service/v3/api-docs"),
HandlerFunctions.http(productServiceUrl))
            .filter(setPath("/api-docs"))
            .build();
}

@Bean
public RouterFunction<ServerResponse> orderServiceSwaggerRoute(){
    return GatewayRouterFunctions.route("order_service_swagger")
            .route(RequestPredicates.path("/aggregate/order-service/v3/api-docs"),
HandlerFunctions.http(orderServiceUrl))
            .filter(setPath("/api-docs"))
            .build();
}

@Bean
public RouterFunction<ServerResponse> inventoryServiceSwaggerRoute(){
    return GatewayRouterFunctions.route("inventory_service_swagger")
            .route(RequestPredicates.path("/aggregate/inventory-service/v3/api-docs"),
HandlerFunctions.http(inventoryServiceUrl))
            .filter(setPath("/api-docs"))
            .build();
}
```

We have also given all permissions for swagger related urls in our SecurityConfig, so now we can access the swagger from the api-gateway

http://localhost:9000/swagger.ui/index.html



OpenAPI definition

/aggregate/inventory-service/v3/api-docs

Servers

http://localhost:8082 - Generated server url

inventory-controller

GET /api/inventory

OpenFeign client to RestClient migration:

```
public interface InventoryClient {

    @GetExchange("/api/inventory")
    boolean isInStock(@RequestParam String skuCode, @RequestParam Integer quantity);
}
```

Adding a RestClientConfig :

```
@Configuration
public class RestClientConfig {

    @Value("${inventory.service.url}")
    private String inventoryServiceUrl;

    @Bean
    public InventoryClient inventoryClient() {
        RestClient restClient = RestClient.builder()
                .baseUrl(inventoryServiceUrl)
                .build();
        var restClientAdapter = RestClientAdapter.create(restClient);
        var httpServiceProxyFactory =
HttpServiceProxyFactory.builderFor(restClientAdapter).build();
        return httpServiceProxyFactory.createClient(InventoryClient.class);
    }

}
```

Some refactoring :

Remove the EnableFeignClient annotation from the root class of order microservice

Remove the dependency from pom.xml

Circuit Breaker

Circuit breaker follows a principle similar to the circuit system in our homes.

We use Resilience4j for this which is a light-weight fault tolerance model.

We need 2 dependencies for this :

1. Circuit-breaker resilience4j
2. Actuator

Actuator is used because resilience relies on the actuator endpoints mostly the health endpoint for useful insights on the implemented circuit breaker.

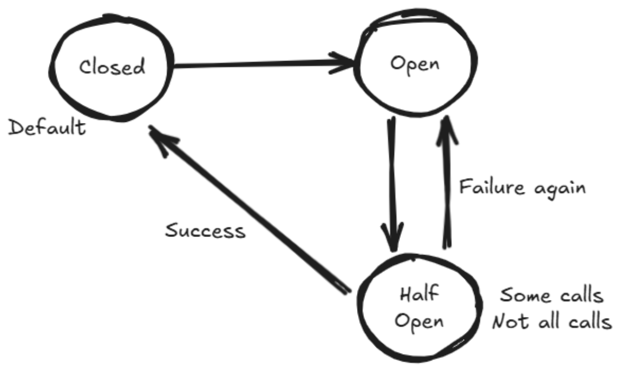
Important :

The properties related to actuator and resilience which we define in application.properties are very important, because a lot of things, mostly numbers, are configured here :

```
# Actuator Endpoints
management.health.circuitbreakers.enabled=true
management.endpoints.web.exposure.include=*
management.endpoint.health.show-details=always
```

```
#Resilinece4j Properties
resilience4j.circuitbreaker.configs.default.registerHealthIndicator=true
resilience4j.circuitbreaker.configs.default.slidingWindowType=COUNT_BASED
resilience4j.circuitbreaker.configs.default.slidingWindowSize=10
resilience4j.circuitbreaker.configs.default.failureRateThreshold=50
resilience4j.circuitbreaker.configs.default.waitDurationInOpenState=5s
resilience4j.circuitbreaker.configs.default.permittedNumberOfCallsInHalfOpenState=3
resilience4j.circuitbreaker.configs.default.automaticTransitionFromOpenToHalfOpenEnabled=true
resilience4j.circuitbreaker.configs.default.minimum-number-of-calls=5

#Resilience4J Timeout Properties
resilience4j.timelimiter.configs.default.timeout-duration=3s

#Resilience4J Retry Properties
resilience4j.retry.configs.default.max-attempts=3
resilience4j.retry.configs.default.wait-duration=2s
```

Different states of circuit-breaker :



Next we add a fallback route, holding the logic to handle a circuit broken situation :

```
@Bean
public RouterFunction<ServerResponse> fallbackRoute() {
        return route("fallbackRoute")
                .GET("/fallbackRoute", request -> ServerResponse.status(HttpStatus.SERVICE_UNAVAILABLE)
                        .body("Service Unavailable, please try again later"))
                .build();
    }
```

Next we add circuit breaker as a filter to the present routes in api-gateway:

```
.filter(CircuitBreakerFilterFunctions.circuitBreaker("productServiceCircuitBreaker",
                        URI.create("forward:/fallbackRoute")))
```

Testing :

```
"circuitBreakers": {
          "status": "UP",
          "details": {
                "productServiceCircuitBreaker": {
                    "status": "UP",
                    "details": {
                        "failureRate": "-1.0%",
                        "failureRateThreshold": "50.0%",
                        "slowCallRate": "-1.0%",
                        "slowCallRateThreshold": "100.0%",
                        "bufferedCalls": 2,
                        "slowCalls": 0,
                        "slowFailedCalls": 0,
                        "failedCalls": 1,
                        "notPermittedCalls": 0,
                        "state": "CLOSED"
                    }
                }
            }
        },
```

After 5 failed requests :

```
"circuitBreakers": {
            "status": "UNKNOWN",
            "details": {
                "productServiceCircuitBreaker": {
                    "status": "CIRCUIT_HALF_OPEN",
                    "details": {
                        "failureRate": "-1.0%",
                        "failureRateThreshold": "50.0%",
                        "slowCallRate": "-1.0%",
                        "slowCallRateThreshold": "100.0%",
                        "bufferedCalls": 0,
                        "slowCalls": 0,
                        "slowFailedCalls": 0,
                        "failedCalls": 0,
                        "notPermittedCalls": 0,
                        "state": "HALF_OPEN"
                    }
                }
            }
        },
```

Now again when ProductService is up and running, after 3 successful requests, state will go back to CLOSED state.

Next since we have a connection between OrderService and InventoryService, therefore we need to apply circuit-breaker here :

So, we add the similar properties to application.properties here as well, but with a slight change, which is :

```
resilience4j.circuitbreaker.instances.inventory.event-consumer-buffer-size=10
```

This inventory is the name given to the InventoryClient defined in this service :

```
public interface InventoryClient {

    Logger log = LoggerFactory.getLogger(InventoryClient.class);
    @GetExchange("/api/inventory")
    @CircuitBreaker(name = "inventory", fallbackMethod = "fallbackMethod")
    @Retry(name = "inventory")
    boolean isInStock(@RequestParam String skuCode, @RequestParam Integer quantity);

    default boolean fallbackMethod(String code, Integer quantity, Throwable throwable) {
        log.info("Cannot get inventory for skucode {}, failure reason: {}", code, throwable.getMessage());
        return false;
    }
}
```

Here we are defining a fallback method, the fallback logic can be improved though.

Also, we are adding a **timelimiter** by making a few additions to the RestClientConfig :

```
@Bean
public InventoryClient inventoryClient() {
    RestClient restClient = RestClient.builder()
            .baseUrl(inventoryServiceUrl)
            .requestFactory(getClientRequestFactory())
            .build();
    var restClientAdapter = RestClientAdapter.create(restClient);
    var httpServiceProxyFactory = HttpServiceProxyFactory.builderFor(restClientAdapter).build();
    return httpServiceProxyFactory.createClient(InventoryClient.class);
}

private ClientHttpRequestFactory getClientRequestFactory() {
    ClientHttpRequestFactorySettings clientHttpRequestFactorySettings =
                   ClientHttpRequestFactorySettings.DEFAULTS
            .withConnectTimeout(Duration.ofSeconds(3))
            .withReadTimeout(Duration.ofSeconds(3));
    return ClientHttpRequestFactories.get(clientHttpRequestFactorySettings);
}
```

H

h

h

h

h