API Gateway

order-service ve inventory-service'in artık dış istemciler için tek bir giriş noktası üzerinden erişilebilir olması: Spring Cloud Gateway ile ayrı bir api-gateway uygulaması kurmak, rota (route) predicate/URI/filtre üçlüsü, lb:// ile Eureka üzerinden keşif tabanlı yönlendirme, özel bir GlobalFilter yazmak (kursun ilk reaktif kodu), ve correlation id gibi sistem geneli konuların neden gateway'de kalıp iş mantığının serviste kalması gerektiği.

Orta 24 dk

Son güncelleme: 23 Ağustos 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/api-gateway

---

API Gateway

Şimdiye kadar bu kurstaki mikroservislere yapılan her çağrı BAŞKA bir mikroservisten geldi -- order-service'in inventory-service'i önce sabit kodlanmış bir URL'le ("Servisler Arası İletişim" dersi), sonra Eureka üzerinden isimle çağırması ("Servis Keşfi ve Eureka" dersinin "Load-Balanced RestClient ile İsimle Çağrı Yapmak" bölümü) gibi. Ama gerçek sistemlerde DIŞARIDAN gelen istemciler de vardır -- bir tarayıcı, bir mobil uygulama -- ve bu istemciler Eureka ağının İÇİNDE çalışmaz, servis isimlerini bilmez, "orders" ile "inventory"'nin ayrı uygulamalar olduğunu bile bilmesine gerek yoktur. Bu ders, dış istemcilerle mikroservis sisteminin tamamı arasına giren parçayı tanıtıyor: API Gateway.

API Gateway Nedir?

API Gateway, bir mikroservis kümesinin ÖNÜNDE duran TEK bir giriş noktasıdır. Dış bir istemci her isteğini TEK bir adrese gönderir (gateway'e); gateway isteğe bakıp onu asıl işleyecek servise -- order-service'e, inventory-service'e, ya da ileride eklenecek herhangi bir servise -- YÖNLENDİRİR. İstemci hiçbir zaman order-service ya da inventory-service'le doğrudan konuşmaz, gerçek adreslerini bilmesine hiç gerek kalmaz.

Neden Var?

Gateway olmadan, dış bir istemcinin HER mikroservisin adresini ayrı ayrı bilmesi gerekirdi -- ve bu adres listesi bir servis her taşındığında/ölçeklendiğinde/yeniden adlandırıldığında değişirdi. Daha kötüsü, tek bir giriş noktası olmadan, sistemin TAMAMINI ilgilendiren konular (kimlik doğrulama, rate limiting, istek loglama) HER serviste AYRI AYRI yeniden yazılmak zorunda kalırdı. API Gateway iki sorunu birden çözer: istemcilerin hatırlaması gereken TEK

bir adres, ve sistem geneli konuların uygulanacağı TEK bir yer -- order-service ile inventory-service'in aynı mantığı tekrar tekrar yeniden icat etmesi yerine.

Tarihçe

Netflix, yaygın kullanılan ilk API gateway'lerinden birini -- Zuul'u -- 2013 civarında, Eureka'yı da üreten aynı iç altyapı için inşa etti ("Servis Keşfi ve Eureka" dersinin "Tarihçe" bölümüne bakınız). Zuul 1 bloklayan (blocking), istek başına bir thread kullanan, eski Servlet modeli üzerine kuruluydu. 2018'de tanıtılan Spring Cloud Gateway, Spring Cloud'un modern cevabı -- baştan itibaren Spring WebFlux (reaktif, non-blocking) üzerine kurulu, çünkü bir gateway sistemdeki HER isteğin yolunun üzerinde durur ve istek başına bir thread tutmadan çok sayıda eşzamanlı bağlantıyı işlemekten en çok fayda gören parça odur. Bu ders Spring Cloud Gateway kullanıyor.

Gateway Uygulamasını Kurmak

api-gateway KENDİ BAŞINA bir Spring Boot uygulaması -- bu kursta order-service, inventory-service ve eureka-server'ın yanında dördüncüsü. Kendi iş mantığı, kendi veritabanı yok; tek işi istekleri almak ve yönlendirmek.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

// A FOURTH Spring Boot application in this course, next to order-service,
// inventory-service, and eureka-server -- but api-gateway isn't a business
// microservice OR a registry. It's the SINGLE entry point external clients (a
// browser, a mobile app) talk to (see "What Is an API Gateway?") -- it never
// contains business logic of its own, it only ROUTES requests to the right
// internal service.
//
// No special annotation is needed here, unlike @EnableEurekaServer on
// EurekaServerApplication (see the Service Discovery & Eureka lesson's "Eureka
// Server: A Central Registry" section) -- adding the
// spring-cloud-starter-gateway dependency is what turns a plain Spring Boot app
// into a gateway; routing itself is configured, not annotated (see "Route
// Configuration: Predicates and Filters").
@SpringBootApplication
public class ApiGatewayApplication {
    public static void main(String[] args) {
        SpringApplication.run(ApiGatewayApplication.class, args);
    }
}
```

```
# api-gateway's own application.yml -- runs as its own independent process, on
# its own port, exactly like eureka-server and every business microservice in
# this course (see the Spring Boot Microservice Basics lesson's "Its Own
# `application.yml`: Port, Application Name, and Database" section).

server:
  port: 8090                          # never 8080 (learning-platform), 8081
                                       # (order-service), 8082 (inventory-service),
                                       # or 8761 (eureka-server)

spring:
  application:
    name: api-gateway                 # not just a log label -- other tools
                                       # (and eventually Observability, see the
                                       # upcoming lesson) will use this to tell
                                       # requests inside the gateway apart from
                                       # requests inside order-service/
                                       # inventory-service

eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/   # api-gateway is ALSO a
                                                       # Eureka client -- it needs
                                                       # the registry to resolve
                                                       # lb://order-service and
                                                       # lb://inventory-service
                                                       # (see "Discovery-Based
                                                       # Routing with lb://")

management:
  endpoints:
    web:
      exposure:
        include: health
```

> api-gateway, order-service ve inventory-service'in aynı nedenle Eureka client'ı OLDUĞU gibi, o da bir Eureka client'ıdır -- istekleri yönlendirebilmek için servis isimlerini gerçek adreslere çözmesi gerekir (aşağıdaki "lb:// ile Keşif Tabanlı Yönlendirme" bölümüne bakınız).

Rota Yapılandırması: Predicate'ler ve Filtreler

ROTA (route), gateway'in temel yapı taşı -- üç şeyin birlikte oluşturduğu bir bütün: bir PREDICATE (gelen bir isteğe bu rotanın uygulanıp uygulanmayacağına karar veren koşul, en yaygın olarak istek yoluyla eşleşme), bir URI (eşleşen isteğin nereye yönlendirileceği), ve isteğe bağlı bir veya daha fazla FİLTRE (istek ya da yanıt üzerinde yol boyunca uygulanan dönüşümler).

```
# Route definitions, ADDED on top of api-gateway's own application.yml (see
# ApiGatewayConfig.yml) -- this is where the gateway's actual job (matching an
# incoming request to the right internal service) is configured. A "route" is
# three things together: a PREDICATE (when does this route apply?), a URI
# (where does it forward to?), and optionally one or more FILTERS (what does it
# do to the request/response along the way?).

spring:
  cloud:
    gateway:
      routes:
        - id: orders-route
          uri: lb://order-service                # "lb://" -- NOT a real host,
                                                   # a service NAME resolved
                                                   # through Eureka, exactly
                                                   # like @LoadBalanced
                                                   # RestClient (see "Discovery-
                                                   # Based Routing with lb://")
          predicates:
            - Path=/orders/**                     # this route only matches
                                                   # requests whose path starts
                                                   # with /orders/
          filters:
            - StripPrefix=0                       # order-service's own
                                                   # controllers already expect
                                                   # paths starting with
                                                   # /orders -- StripPrefix=0
                                                   # means "forward the path
                                                   # exactly as received"

        - id: inventory-route
          uri: lb://inventory-service
          predicates:
            - Path=/inventory/**
          filters:
            - StripPrefix=0
```

lb:// ile Keşif Tabanlı Yönlendirme

Yukarıdaki rota yapılandırmasındaki `lb://order-service` URI'sine dikkat edin -- bu, "Servis Keşfi ve Eureka" dersindeki `@LoadBalanced RestClient`'ın `http://inventory-service`'i çağırmasıyla (dersin "Load-Balanced RestClient ile İsimle Çağrı Yapmak" bölümüne bakınız) AYNI fikir: `lb://` gerçek bir protokol değil, Spring Cloud Gateway'e `order-service`'i Eureka üzerinden çözmesini ve o isim altında o anda kayıtlı kaç örnek varsa aralarında yük dengelemesi yapmasını söyler. Gateway'in order-service'in gerçek host ve portunu hiçbir yerde sabit kodlamasına hiç gerek kalmaz.

Özel Bir Filtre Yazmak

`StripPrefix` (yukarıda kullanıldı) gibi hazır filtreler yaygın durumları karşılar, ama bir gateway `GlobalFilter` üzerinden her istekte özel kod da çalıştırabilir. Bu, kursun İLK

REAKTİF kodu -- Spring Cloud Gateway, order-service ve inventory-service'in controller'larının kullandığı bloklayan Spring MVC'nin aksine Spring WebFlux üzerinde çalışır.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.cloud.gateway.filter.GatewayFilterChain;
import org.springframework.cloud.gateway.filter.GlobalFilter;
import org.springframework.core.Ordered;
import org.springframework.stereotype.Component;
import org.springframework.web.server.ServerWebExchange;
import reactor.core.publisher.Mono;

// Spring Cloud Gateway is built on Spring WebFlux (reactive), NOT the same
// blocking Spring MVC used by order-service/inventory-service's controllers --
// this is the FIRST reactive code in this course. A GlobalFilter runs for
// EVERY route, unlike a filter attached to one specific route (see
// GatewayRoutesConfig.yml's per-route "filters:" list) -- implementing
// Ordered lets several global filters run in a defined sequence.
//
// The chain.filter(exchange) call is what actually forwards the request
// toward its destination (order-service or inventory-service, picked by the
// matching route) -- code BEFORE that call runs before forwarding, code
// inside .then(...) runs AFTER the downstream response comes back. Nothing
// here blocks a thread waiting for that response, unlike a plain
// @RestController method.
@Component
class RequestLoggingGlobalFilter implements GlobalFilter, Ordered {

    private static final Logger log = LoggerFactory.getLogger(RequestLoggingGlobalFilter.
class);

    @Override
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
        long startedAt = System.currentTimeMillis();
        String method = exchange.getRequest().getMethod().name();
        String path = exchange.getRequest().getPath().value();

        return chain.filter(exchange)
                .then(Mono.fromRunnable(() -> {
                    long tookMillis = System.currentTimeMillis() - startedAt;
                    int status = exchange.getResponse().getStatusCode() != null
                            ? exchange.getResponse().getStatusCode().value()
                            : 0;
                    log.info("{} {} -> {} ({} ms)", method, path, status, tookMillis);
                }));
    }

    @Override
    public int getOrder() {
        return Ordered.LOWEST_PRECEDENCE;   // run last among global filters --
                                             // logs the FINAL outcome, after any
                                             // other filter has already run
    }
}
```

> Bir `GlobalFilter`'ın `filter(...)` metodu `Mono<Void>` döndürmeli ve çağıran thread'i ASLA BLOKLAMAMALI (JDBC çağrısı yok, `Thread.sleep` yok, bloklayan I/O yok) -- Spring WebFlux, tüm eşzamanlı istekler arasında paylaşılan küçük, sabit sayıda thread çalıştırır; bunlardan birini bile bloklamak ilgisiz istekleri de durdurur.

Sistem Geneli Konular Nereye Ait?

Gateway, hangi servisin isteği sonunda işleyeceğinden BAĞIMSIZ olarak HER isteği ilgilendiren konular için doğal bir yer -- örneğin bir correlation id atamak, böylece dış bir istek ileride birden fazla iç servis üzerinden izlenebilir.

```
import org.springframework.cloud.gateway.filter.GatewayFilterChain;
import org.springframework.cloud.gateway.filter.GlobalFilter;
import org.springframework.core.Ordered;
import org.springframework.http.server.reactive.ServerHttpRequest;
import org.springframework.stereotype.Component;
import org.springframework.web.server.ServerWebExchange;
import reactor.core.publisher.Mono;

import java.util.UUID;

// A cross-cutting concern that genuinely BELONGS at the gateway (see "Where
// Cross-Cutting Concerns Belong"): every request entering the system gets a
// correlation id -- one shared value that order-service, inventory-service,
// and any future service can log alongside their own messages, making it
// possible to trace ONE external request across MULTIPLE internal services.
// This filter does NOT invent request tracing itself -- it only ensures the
// header exists as early as possible; an upcoming Observability lesson covers
// actually propagating and using it downstream.
@Component
class CorrelationIdGatewayFilter implements GlobalFilter, Ordered {

    private static final String CORRELATION_ID_HEADER = "X-Correlation-Id";

    @Override
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
        ServerHttpRequest request = exchange.getRequest();
        if (request.getHeaders().containsKey(CORRELATION_ID_HEADER)) {
            // A caller (or a previous hop) already set one -- keep it AS IS, don't
            // overwrite an id a client may already be tracking.
            return chain.filter(exchange);
        }

        String correlationId = UUID.randomUUID().toString();
        ServerHttpRequest mutatedRequest = request.mutate()
                .header(CORRELATION_ID_HEADER, correlationId)
                .build();

        return chain.filter(exchange.mutate().request(mutatedRequest).build());
    }

    @Override
    public int getOrder() {
        return Ordered.HIGHEST_PRECEDENCE;   // run FIRST -- every later filter
                                              // (including RequestLoggingGlobalFilter)
                                              // and every downstream service should
                                              // see the header already set
    }
}
```

> Bu ders yalnızca correlation id'yi ATIYOR -- onu order-service ile inventory-service arasındaki giden çağrılara GERÇEKTEN aktarmak ve log satırlarını birbirine bağlamak için kullanmak, yakında gelecek Observability dersinin konusu.

Bir Gateway'in YAPMAMASI Gerekenler

Gateway zaten her isteği gördüğü için içine iş mantığı koymak cazip gelebilir -- ama bu, iş kurallarını onları SAHİPLENEN servislerden ÇIKARIP hiçbir domain bilgisi olmayan bir altyapı parçasına TAŞIR. Bir gateway yönlendirmeli, ve gerçekten İSTEĞİN kendisiyle ilgili konuları (kimlik doğrulama, rate limiting, loglama, correlation id'ler) uygulamalı -- örneğin bir siparişin geçerli olup olmadığına karar VERMEMELİ. O karar, tıpkı öncekiler gibi, order-service'e ait.

Best Practices

- **Servisi İSİMLE (`lb://servis-adı`) yönlendir, asla sabit kodlanmış bir host:port ile değil** -- `@LoadBalanced RestClient` ile aynı gerekçe ("Servis Keşfi ve Eureka" dersine bakınız).
- **Filtreleri istek/yanıt konularına odaklı tut** (loglama, correlation id'ler, header'lar) -- iş kararına benzeyen her şeyi sahibi olan servise geri it.
- **api-gateway'i de kendi `spring.application.name`'iyle Eureka'ya kaydet** -- gateway'in kendisine keşif üzerinden hiçbir şey yönlendirilmese bile, onu registry'de diğer her servisin yanında görünür tutar.
- **Global filtreleri `Ordered` ile bilinçli olarak sırala** -- bir correlation id atayan filtre, onu kullanan bir loglama filtresinden ÖNCE çalışmalı.

Yaygın Hatalar

- **İş mantığını (doğrulama, hesaplama) doğrudan bir gateway filtresinin İÇİNE koymak.** O mantık, her isteğin geçtiği paylaşılan altyapıya değil, sahibi olan mikroservise ait.
- **Thread'i BLOKLAYAN bir `GlobalFilter` yazmak** (bir JDBC çağrısı, `Thread.sleep`) -- Spring WebFlux'in thread modeli, aynı hatayı bloklayan bir Spring MVC controller'da yapmaktan ÇOK daha yıkıcı hale getiriyor.
- **Bir rotanın `uri`'sinde alt akıştaki servisin host:port'unu sabit kodlamak**, `lb://servis-adı` kullanmak yerine -- Service Discovery'nin çözmeye çalıştığı sorunu birebir yeniden yaratır.
- **Gateway'in birden fazla servisin yanıtını TEK bir yanıtta birleştirmesini beklemek.** Sade Spring Cloud Gateway TEK bir isteği TEK bir servise yönlendirir; birden fazla çağrıyı tek bir yanıtta birleştirmek farklı bir desendir (genellikle Backend for Frontend olarak anılır), bu dersin kapsamı dışında.

Özet, Cheat Sheet ve Terimler Sözlüğü

API Gateway, dış istemcilerin tek tek mikroservisler yerine konuştuğu tek bir giriş noktasıdır. Spring Cloud Gateway rotaları predicate'lerden (bir rotanın ne zaman uygulandığı), bir

URI'den (nereye yönlendirdiği, genellikle Eureka üzerinden çözülen `lb://servis-adı`) ve filtrelerden (yol boyunca uygulanan dönüşümler) oluşur -- hem `StripPrefix` gibi hazır filtreler hem de Spring WebFlux üzerinde çalışan ve ASLA bloklamaması gereken özel `GlobalFilter`'lar. Gateway, istek seviyesindeki, sistem geneli konular (correlation id'ler, loglama) için doğru yerdir -- iş mantığı için ASLA, o her zaman sahibi olan serviste kalır.

Hızlı referans:

```
@SpringBootApplication
public class ApiGatewayApplication { ... }   // kendi başına bir Spring Boot
                                              // uygulaması, kendi iş mantığı yok

// application.yml
// spring.cloud.gateway.routes:
//   - id: orders-route
//     uri: lb://order-service               // servis ADI, Eureka üzerinden çözülür
//     predicates:
//       - Path=/orders/**
//     filters:
//       - StripPrefix=0

@Component
class SomeGlobalFilter implements GlobalFilter, Ordered {
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
        return chain.filter(exchange);       // isteği ileriye yönlendirir
    }
}
```

Terimler Sözlüğü

API Gateway — Dış istekleri doğru iç mikroservise yönlendiren tek bir giriş noktası.

Rota (Route) — Gateway'in temel yapı taşı: bir predicate, bir hedef URI, ve isteğe bağlı filtreler.

Predicate — Gelen bir isteğe bir rotanın uygulanıp uygulanmayacağına karar veren koşul (en yaygın olarak bir yol/path deseni).

GlobalFilter — Gateway'den geçen HER istekte çalışan, Spring WebFlux üzerinde reaktif olarak yazılan özel kod.

`lb://` — Spring Cloud Gateway'e sabit bir adres yerine bir servis ismini Eureka üzerinden çözmesini ve örnekleri arasında yük dengelemesi yapmasını söyleyen sahte bir protokol.