Configuration Management

Her servisin şimdiye kadar kendi application.yml'inde tuttuğu yapılandırmayı Spring Cloud Config ile merkezileştirmek: @EnableConfigServer ile ayrı bir config-server uygulaması kurmak, bir Config Repository'nin neyi merkezileştirmeye değdiğine karar vermesi, spring.config.import ile Config Client olmak, profillerle ortam bazlı override, @RefreshScope ile yeniden başlatmadan yapılandırma yenileme, ve sırların neden düz metin olarak saklanmaması gerektiği.

Orta 24 dk

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

---

Configuration Management

Bu kurstaki her servis kendi yapılandırmasını kendi `application.yml`'inde tuttu -- order-service'in port ve datasource ayarları ("Mikroservis Yapılandırma" dersinin "Kendi `application.yml`'i: Port, Uygulama Adı ve Veritabanı" bölümüne bakınız), Eureka client ayarları ("Servis Keşfi ve Eureka" dersine bakınız), ve şimdi Resilience4j örnekleri ("Resilience4j" dersine bakınız). Bu, birkaç servis için gayet iyi -- ama yirmi servisin AYNI datasource pool-size ayarına ihtiyaç duyduğunu, ya da bir değerin HEPSİNDE AYNI ANDA değişmesi gerektiğini hayal edin. Aynı bloğu yirmi `application.yml` dosyasına kopyala-yapıştır yapmak, ve değiştiğinde yirmisini de düzenlemek, ölçeklenmiyor. Bu ders, yapılandırmayı bunun yerine merkezileştiren parçayı tanıtıyor: Spring Cloud Config.

Configuration Management Nedir?

Mikroservis anlamında yapılandırma yönetimi, yapılandırmayı onu kullanan uygulamaların İÇİNDE değil, her servisin kendi paketlenmiş kodunda tekrarlamak yerine, TEK bir merkezi yerde tutmaktır. Bir Config Server, kendisinden isimle isteyen HERHANGİ bir servise, başlangıçta (ve bu dersin göreceği gibi bazen yeniden başlatmaya bile gerek kalmadan) yapılandırmayı AĞ üzerinden sunar.

Neden Var?

Birçok serviste tekrarlanan yapılandırma iki gerçek sorun yaratır: birincisi, gerçekten PAYLAŞILAN bir değerin (bir bağlantı havuzu boyutu, bir feature flag, üçüncü taraf bir API'nin base URL'i) her serviste kendi dosyasında ayrı ayrı güncellenmesi gerekir -- birini atlamak

kolay, ve her servis, kodunu hiç etkilemeyen bir değeri almak için kendi yeniden deploy'unu ister. İkincisi, bazı yapılandırmaların ortama göre FARKLI olması gerekir (staging'de bir veritabanı URL'i, production'da başka) -- her biri için tüm dosyayı çoğaltmadan. Yapılandırmayı tek bir sunucuda merkezileştirmek, ortam bazlı override desteğiyle ("Profiller: Farklı Ortamlar İçin Farklı Yapılandırma" bölümüne bakınız), her ikisini de çözer.

Tarihçe

Spring Cloud Config, Eureka ve Zuul'la (bkz. "Servis Keşfi ve Eureka" ve "API Gateway" derslerinin "Tarihçe" bölümleri) yaklaşık 2015'te birlikte yayınlanan, Spring ekosistemine getirilen aynı Netflix-OSS-ilhamlı araç dalgasının bir parçası, orijinal Spring Cloud projelerinden biriydi. Eureka'nın aksine, hiçbir Netflix kütüphanesi üzerine kurulu değil -- her dağıtık sistemin er ya da geç karşılaştığı bir soruna Spring-native bir cevap, ve bugün hâlâ Spring Cloud içinde aktif olarak bakımı yapılıyor.

Config Server'ı Kurmak

config-server, eureka-server gibi, KENDİ BAŞINA bağımsız bir Spring Boot uygulaması -- bu kursta order-service, inventory-service, eureka-server ve api-gateway'in yanında beşincisi.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

// A FIFTH Spring Boot application in this course, next to order-service,
// inventory-service, eureka-server, and api-gateway -- config-server, like
// eureka-server, is pure infrastructure: no business logic, no domain database.
// Its only job is serving CONFIGURATION files to every other service in the
// system (see "Setting Up a Config Server").
//
// @EnableConfigServer is the ONE annotation that turns a plain Spring Boot app
// into a Config Server -- exactly the same shape as @EnableEurekaServer on
// EurekaServerApplication (see the Service Discovery & Eureka lesson's "Eureka
// Server: A Central Registry" section).
@SpringBootApplication
@EnableConfigServer
public class ConfigServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(ConfigServerApplication.class, args);
    }
}
```

```
# config-server's own application.yml -- runs as its own independent process,
# on its own port, exactly like eureka-server (see the Service Discovery &
# Eureka lesson) and api-gateway (see the API Gateway lesson).

server:
  port: 8888                          # Config Server's traditional default
                                       # port, by the same convention Eureka
                                       # follows with 8761

spring:
  application:
    name: config-server
  profiles:
    active: native                    # "native" backend: reads configuration
                                       # files from the local filesystem/
                                       # classpath, NOT from a Git repository --
                                       # the simplest backend to reason about
                                       # while learning (see "The Config
                                       # Repository: Where Configuration
                                       # Actually Lives" for the real-world
                                       # Git-backed alternative)
  cloud:
    config:
      server:
        native:
          search-locations: classpath:/config-repo/   # where the actual
                                                       # order-service.yml,
                                                       # inventory-service.yml
                                                       # etc. files live
```

> Bu ders, örneği kendi içinde tam (self-contained) tutmak için Config Server'ın bir Git deposu yerine "native" backend'ini (yerel bir dosya sistemi/classpath dizini) kullanıyor -- production'da, `spring.cloud.config.server.git.uri` 'nin gerçek bir Git deposuna işaret etmesi çok daha yaygın, çünkü yapılandırma değişikliklerine kodla aynı sürüm geçmişini ve review sürecini kazandırır.

Config Repository: Yapılandırma Aslında Nerede Yaşıyor?

Gerçek yapılandırma değerleri -- servis başına bir YAML dosyası, o servisin `spring.application.name` 'iyle eşleşecek şekilde adlandırılmış -- herhangi bir servisin kendi paketlenmiş kodundan tamamen ayrı, Config Repository'de yaşar.

```
# This file lives INSIDE config-server's own repository (at
# classpath:/config-repo/order-service.yml, matching ConfigServerConfig.yml's
# search-locations) -- NOT inside order-service itself. Config Server serves
# this content to order-service over HTTP when it asks for configuration
# named "order-service" (see "Making order-service a Config Client").
#
# Notice this is a SUBSET of what used to live entirely inside order-service's
# own application.yml (see the Spring Boot Microservice Basics lesson's "Its
# Own `application.yml`: Port, Application Name, and Database" section) --
# server.port and spring.application.name stay LOCAL to order-service (a
# service needs to know its own port before it can even ask Config Server for
# anything else), only the parts worth centralizing move here.

greeting:
  message: "Welcome to the Order Service"   # the property RefreshableGreetingController
                                             # exposes -- changing THIS file and
                                             # refreshing (see "Refreshing
                                             # Configuration Without Restarting")
                                             # changes the running service's
                                             # behavior with no restart at all

datasource:
  max-pool-size: 10                         # exactly the kind of tuning value that's
                                             # genuinely worth sharing/centralizing
                                             # across every service using the same
                                             # database technology
```

Bu dosyanın yalnızca gerçekten MERKEZİLEŞTİRMEYE değer olanı içerdiğine dikkat edin -- `server.port` ve `spring.application.name` order-service'in kendi yerel `application.yml`'inde kalır, çünkü bir servisin Config Server'a herhangi bir şey İSTEMEDEN önce kendi kimliğini ve portunu bilmesi gerekir.

order-service'i Bir Config Client Yapmak

order-service'in MEVCUT `application.yml`'ine eklenen tek bir özellik, onu başlangıçta config-server'dan yapılandırma çekip birleştirir hale getiren şey.

```
# Added to order-service's own application.yml -- on top of everything from
# the Spring Boot Microservice Basics lesson, plus the eureka.client block
# from Service Discovery & Eureka. Nothing already there is REMOVED; this is
# what makes order-service ask config-server for MORE configuration on
# startup, in addition to what it already knows locally.

spring:
  config:
    import: "configserver:http://localhost:8888"   # on startup, order-service
                                                     # fetches config named after
                                                     # its own spring.application.name
                                                     # ("order-service") from this
                                                     # URL -- Spring Cloud Config
                                                     # matches the request to
                                                     # config-repo/order-service.yml
                                                     # automatically
```

Profiller: Farklı Ortamlar İçin Farklı Yapılandırma

Bir Config Repository dosyası PROFİL'e göre daha da bölünebilir -- `order-service-staging.yml` ve `order-service-production.yml`, yukarıdaki temel `order-service.yml`'in yanında, order-service hangi profille başlatıldığına bağlı olarak (örneğin `spring.profiles.active=staging`) onu override eder ya da ona ekleme yapar. Bu, Spring Boot'un yerelde zaten kullandığı AYNI profil mekanizması (bu projenin kendi `application-dev.yml` / `application-prod.yml` deseni, `application-dev.yml`'in `application.yml`'i override etmesi, tanıdık geliyorsa) -- Config Server bunu yalnızca yerel dosyalar yerine merkezi barındırılan dosyalara uyguluyor.

Yeniden Başlatmadan Yapılandırmayı Yenilemek: @RefreshScope

Varsayılan olarak, `@Value` ile enjekte edilen bir özellik yalnızca BİR KEZ, Spring bean'i oluştururken okunur -- bir Config Repository dosyasını sonradan düzenlemenin zaten çalışan bir servis üzerinde HİÇBİR etkisi olmaz. `@RefreshScope` bunu değiştirir: bir refresh tetiklendiğinde (o servisin kendi `/actuator/refresh` endpoint'ine bir `POST`), bean'in atılıp yeniden oluşturulmasına, `@Value`'larını yeniden okumasına izin verir.

```
import org.springframework.beans.factory.annotation.Value;
import org.springframework.cloud.context.config.annotation.RefreshScope;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

// Without @RefreshScope, a @Value-injected property is read EXACTLY ONCE, when
// this bean is first created -- editing config-repo/order-service.yml's
// greeting.message afterward would have NO effect on an already-running
// order-service; a full restart would be the only way to pick it up.
// @RefreshScope makes Spring throw this bean away and recreate it (re-reading
// every @Value) whenever a refresh is triggered (see "Refreshing Configuration
// Without Restarting: @RefreshScope") -- the endpoint below always reflects
// whatever config-server is currently serving, without order-service ever
// stopping.
@RestController
@RefreshScope
class RefreshableGreetingController {

    @Value("${greeting.message}")
    private String greetingMessage;

    // GET /greeting -- returns whatever greeting.message currently is. Change
    // OrderServiceExternalConfig.yml, POST to order-service's own
    // /actuator/refresh, and call this again: the response changes, with
    // order-service never having restarted.
    @GetMapping("/greeting")
    String greeting() {
        return greetingMessage;
    }
}
```

> Tek bir `/actuator/refresh` çağrısı yalnızca ÇAĞRILDIĞI TEK servisi yeniler -- birçok servisle, hepsini elle tetiklemek, dosyaları elle düzenlemekten çok daha iyi ölçeklenmez. Spring Cloud Bus (bir refresh olayını paylaşılan bir message broker üzerinden tüm servislere BİRDEN yayınlamak) bunu çözer, ama bu dersin kapsamı dışında -- bir sistemde birkaçtan fazla servis olduğunda var olduğunu bilmekte fayda var.

Sırlar (Secrets): Config Server'ın Düz Metin Olarak SAKLAMAMASI Gerekenler

Her şey bir Config Repository dosyasına düz metin olarak ait değil -- bir veritabanı şifresi, ortama göre değiştiği anlamda bir yapılandırmadır, ama onu şifrelenmemiş bir dosyada (özel bir Git deposunda bile) saklamak gerçek bir güvenlik riski. Spring Cloud Config tek tek değerleri şifrelemeyi destekler, ve özel sır (secret) araçları (en yaygın eşleşme HashiCorp Vault) tam olarak bunun için var -- bu ders, gerçekten hassas değer için `${ORDERS_DB_PASSWORD}`'ü ("Mikroservis Yapılandırma" dersine bakınız) bir ortam değişkeni olarak kullanmaya devam ediyor, ve yalnızca sır OLMAYAN yapılandırmayı merkezileştiriyor.

Best Practices

- **Gerçekten PAYLAŞILAN ya da koddan bağımsız değişen değerleri merkezileştir** (havuz boyutları, feature flag'ler, üçüncü taraf URL'leri) -- bir servisin kendi kimliğini (port, uygulama adı) kendi yerel `application.yml`'inde bırak.
- **Ortama özgü override'lar için profilleri kullan**, elle neredeyse birebir aynı ayrı dosyalar tutmak yerine.
- **Canlı güncellemeye ihtiyaç duyan bean'leri BİLİNÇLİ OLARAK `@RefreshScope` ile işaretle, HER YERE değil** -- refresh yapmanın gerçek bir maliyeti var (bean yeniden oluşturma), ve çoğu bean buna hiç ihtiyaç duymaz.
- **Sırları hiçbir zaman bir Config Repository dosyasına düz metin olarak koyma** -- şifreleme ya da özel bir sır aracı kullan, tıpkı bu dersin `ORDERS_DB_PASSWORD`'ü merkezileştirmek yerine bir ortam değişkeni olarak tutmaya devam etmesi gibi.

Yaygın Hatalar

- **`server.port` ya da `spring.application.name`'i merkezileştirmek.** Bir servisin, Config Server'a herhangi bir şey İSTEMEDEN önce, ikisine de YEREL olarak ihtiyacı var.
- **Bir Config Repository dosyasını düzenleyip çalışan bir servisin bunu HEMEN almasını beklemek.** `@RefreshScope` ve açık bir refresh çağrısı (ya da Spring Cloud Bus) olmadan, bir sonraki yeniden başlatmaya kadar hiçbir şey değişmez.
- **Bir veritabanı şifresini ya da API anahtarını bir Config Repository dosyasına düz metin olarak saklamak.** Özel bir depoda bile olsa, bu gerçek bir kimlik bilgisi sızıntısı bekliyor demek -- yukarıdaki "Sırlar" bölümüne bakınız.
- **`@RefreshScope`'u "ne olur ne olmaz" diye her bean'e uygulamak.** Bean oluşturmaya gerçek bir ek yük bindirir ve pratikte çalışma zamanında hiç değişmeyen bean'ler için bir bean'in yaşam döngüsü hakkında düşünmeyi karmaşıklaştırır.

Özet, Cheat Sheet ve Terimler Sözlüğü

Configuration management, yapılandırmayı onu kullanan servislerin DIŞINDA merkezileştirir. Spring Cloud Config'in Config Server'ı (`@EnableConfigServer`), servis başına YAML dosyalarını bir Config Repository'den (production'da bir Git deposu, burada yerel bir dizin) sunar; bir servis, tek bir `spring.config.import` özelliğiyle bir Config Client'a dönüşür. Profiller, bir Config Repository dosyasının ortam başına override edilmesine izin verir -- bu projenin kendi `application-dev.yml` / `application-prod.yml`'inin yerelde zaten çalıştığı AYNI şekilde. `@RefreshScope`, bir bean'in `/actuator/refresh` tarafından tetiklenen, yeniden başlatma olmadan yeni yapılandırmayı almasını sağlar. Sırlar hiçbir zaman bir Config Repository dosyasına düz metin olarak ait değildir.

Hızlı referans:

```
@SpringBootApplication
@EnableConfigServer
public class ConfigServerApplication { ... }   // kendi başına bir Spring Boot
                                                // uygulaması, yalnızca yapılandırma sunar

// order-service'in application.yml'i
// spring.config.import: "configserver:http://localhost:8888"

@RestController
@RefreshScope                                  // refresh'te @Value'ı yeniden okur,
class SomeController {                         // yalnızca başlangıçta değil
    @Value("${greeting.message}")
    private String greetingMessage;
}
```

Terimler Sözlüğü

Config Server — `@EnableConfigServer` ile etkinleştirilen, servis başına yapılandırmayı ağ üzerinden sunan bir Spring Boot uygulaması.

Config Repository — Gerçek yapılandırma dosyalarının yaşadığı yer -- production'da bir Git deposu, bu dersin örneğinde yerel bir dizin.

Config Client — Başlangıçta bir Config Server'dan `spring.config.import` üzerinden yapılandırma çekip birleştiren bir servis.

Profil (Profile) — Bir temel yapılandırma dosyasını override eden ya da genişleten, adlandırılmış bir yapılandırma varyantı (ör. staging, production).

`@RefreshScope` — Bir bean'in `@Value` ile enjekte edilen özelliklerinin, yalnızca başlangıçta değil, bir refresh tetiklendiğinde yeniden okunmasını sağlayan bir annotation.