

Spring Cloud Config



Spring Cloud Config

- Konfigürasyon ayarlarının **dağıtık ortamda harici biçimde yönetilmesini** sağlar
- Hem **istemci** hem de **sunucu** tarafı için kabiliyetler sunar
- Default durumda backend storage olarak **“git”** kullanılır

Backend Repo Türleri

- Spring Cloud Config aşağıdaki **backend store türlerini** desteklemektedir
 - File based repository (git,svn)
 - Native (doğrudan file sistemdeki dosyalar)
 - Vault
 - JDBC
- İstenirse **birden fazla farklı backend repository'de** aynı anda (composite) kullanılabilir

Sunucu Tarafı

pom.xml



```
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-config-server</artifactId>
  </dependency>
</dependencies>
```

Sunucu Tarafı

- Spring Cloud Bus üzerinden **AMQP** ile konfigürasyondaki değişiklikleri servislere **broadcast** etmek de mümkündür

pom.xml

```
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-config-monitor</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-bus-amqp</artifactId>
  </dependency>
</dependencies>
```

Sunucu Tarafı

application.properties



```
spring.cloud.config.server.git.uri=http://gitserver/global-config
spring.application.name=configserver
```

Sunucu Tarafı

```
@EnableConfigServer  
@SpringBootApplication  
public class ConfigserviceApplication {  
    ...  
}
```

Sunucu Tarafı

```
public static void main(String[] args) {

    SpringApplication app = new SpringApplication(ConfigServerApplication.class);
    app.addInitializers(
        new ApplicationContextInitializer<ConfigurableApplicationContext>() {

            @Override
            public void initialize(ConfigurableApplicationContext applicationContext) {
                ProtocolResolver resolver = new ProtocolResolver() {

                    private FileSystemResourceLoader resourceLoader =
                        new FileSystemResourceLoader();

                    @Override
                    public Resource resolve(String location,
                        ResourceLoader resourceLoader) {
                        if(location != null && location.startsWith("//")) {
                            return this.resourceLoader.getResource(location);
                        }

                        return null;
                    }
                };
                applicationContext.addProtocolResolver(resolver);
            }
        });
    app.run(args);
}
```



Eğer git uri olarak file sys'den bir lokasyon kullanmak isterseniz uri property'sinin başına // ekleyip buradaki ApplicationContextInitializer özelleştirmesini yapmanız gerekir

İstemci Tarafı

pom.xml



```
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-config</artifactId>
  </dependency>
</dependencies>
```

İstemci Tarafı (Doğrudan URI Erişimi ile)



bootstrap.properties



```
spring.cloud.config.name=configclient  
spring.cloud.config.uri=http://www.configserver.com/
```



Backend git store'da configclient.properties isimli dosya içerisinde bu client'a ait property'ler yönetilecektir

İstemci Tarafı (Service Discovery ile)



bootstrap.properties



```
spring.cloud.config.name=configclient  
spring.cloud.config.discovery.service-id=configserver  
spring.cloud.config.discovery.enabled=true
```



Backend git store'da configclient.properties isimli dosya içerisinde bu client'a ait property'ler yönetilecektir

Property Değerlerine Web Erişimi

- Config Server'daki her bir servisin property değerleri **<service-id>/<profile-id>** şeklinde bir URI ile erişilebilir

http://localhost:8888/configclient/dev

Profil adıdır

client'ın spring.cloud.config.name ile tanımladığı isimdir

Property'lerinin Paylaşılması

- Dosya tabanlı (git, svn, native) repository kullanımında config server'da aşağıdaki resource'lardaki tanımlar **bütün istemciler tarafından** paylaşılır
 - application.properties
 - application.yml
 - application-*.properties...
- Bu tür dosyalarda istemcilere ait **global default property değerleri** yönetilebilir

Property Değerlerinin Encrypted Saklanması

- Hassas property değerlerinin saklanmasında **simetrik veya asimetrik key** yöntemi ile **encryption** kullanılabilir
- Simetrik key kullanımı daha pratiktir

bootstrap.properties
↓
encrypt.key=secret

Property Değerlerinin Encrypted Saklanması

- Password gibi property değerleri repo içerisinde **encrypted** tanımlanabilir

spring.datasource.password: {cipher}FKSAJDFFGY0S8F7GLHAKERGFHLSAJ

- Bu değerler istemcilere gönderilmeden evvel otomatik olarak **decrypt** edilirler

Encrypt/Decrypt Endpoint'leri

- Config Server'in enc/dec kabiliyetine **/encrypt** ve **/decrypt** endpoint'leri üzerinden de erişmek mümkündür

```
$ curl localhost:8888/encrypt -d mysecret  
682bc583f4641835fa2db009355293665d2647dade3375c0ee201de2a49f7bda
```

```
$ curl localhost:8888/decrypt -d 682bc583f4641835fa2db009355293665d2647dade3375c0ee201de2a49f7bda  
mysecret
```


İletişim

- **Harezmi** Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- <http://www.java-egitimleri.com>
- info@java-egitimleri.com

