Spring Cloud - *NETFLIX*

Spring Cloud - Netflix

- Spring Cloud Netflix, Spring Environment ve diğer Spring programlama modeli deyimlerine otomatik yapılandırma ve bağlama yoluyla Spring Boot uygulamaları için Netflix OSS entegrasyonları sağlar.
- Birkaç basit anatasyonlarla uygulamanızın içindeki ortak kalıpları hızlı bir şekilde etkinleştirebilir ve yapılandırabilir ve savaşta test edilmiş Netflix bileşenleriyle geniş dağıtılmış sistemler oluşturabilirsiniz. Sağlanan kalıplar Service Discovery'yi **(Eureka)** içerir.

Spring Cloud – Netflix Özellikleri

- **Service Discovery(Hizmet Keşfi):** Eureka örnekleri kaydedilebilir ve client'ler(istemciler-alıcılar) Spring-managed bean'lerini kullanarak örnekleri keşfedebilir
- **Service Discovery(Hizmet Keşfi):** Bildirim temelli Java yapılandırmasıyla yerleşik bir Eureka sunucusu oluşturulabilir

Spring Cloud Netflix ve Eureka Core «sınıf yolunda» olduğu sürece, **@EnableEurekaClient** içeren herhangi bir Spring Boot uygulaması **http://localhost:8761** (eureka.client.serviceUrl.defaultZone'un varsayılan değeri) üzerinden bir Eureka sunucusuyla iletişim kurmaya çalışacaktır:

```
@SpringBootApplication
@EnableEurekaClient
@RestController
public class Application {

  @RequestMapping("/")
  public String home() {
    return "Hello World";
  }

  public static void main(String[] args) {
    SpringApplication.run(Application.class, args);
  }

}
```

Kendi server'inizi çalıştırmak için **spring-cloud-starter-netflix-eureka-server** dependency ve **@EnableEurekaServer**'ı kullanın.

LEARN-ÖĞREN

1. Service Discovery: Eureka Clients (Hizmet Keşfi: Eureka Müşterileri)

Hizmet Keşfi, mikro hizmet tabanlı bir mimarinin temel ilkelerinden biridir. Her client'i(alıcıyı) veya bir tür sözleşmeyi elle yapılandırmaya çalışmak zor olabilir ve kırılgan olabilir. Eureka, Netflix Hizmet Keşif Serveri ve Client'ıdır. Server, yüksek düzeyde kullanılabilir olacak şekilde yapılandırılabilir ve dağıtılabilir, her bir server, kayıtlı hizmetler hakkındaki durumu diğerlerine kopyalar.

1.1. How to Include Eureka Client (Eureka İstemcisi Nasıl Dahil Edilir?)

Eureka Client'ını(alıcınızı) projenize dahil etmek için, **org.springframework.cloud** grup kimliğiyle ve **spring-cloud-starter-netflix-eureka-client** yapı kimliğiyle başlatıcıyı kullanın. Mevcut Spring Cloud Release Train ile derleme sisteminizi kurma hakkında ayrıntılar için Spring Cloud Project sayfasına bakın.

1.2. Registering with Eureka (Eureka ile kaydolma)

Bir müşteri Eureka'ya kaydolduğunda, kendisi hakkında ana bilgisayar, bağlantı noktası, sağlık göstergesi URL'si, ana sayfa ve diğer ayrıntılar gibi meta veriler sağlar. Eureka, bir hizmete ait her bir örnekten kalp atışı mesajları alır. Kalp atışı yapılandırılabilir bir zaman çizelgesinde başarısız olursa, örnek normalde kayıt defterinden kaldırılır.

Aşağıdaki örnek, minimal bir Eureka istemci uygulamasını göstermektedir:

```
@SpringBootApplication
@RestController
public class Application {

   @RequestMapping("/")
   public String home() {
      return "Hello world";
   }

   public static void main(String[] args) {
      new SpringApplicationBuilder(Application.class).web(true).run(args);
   }

}
```

Yukarıdaki örneğin normal bir Spring Boot uygulamasını gösterdiğine dikkat edin. **Spring-cloud-starter-netflix-eureka-client**'ı sınıf yolunda bulundurarak, uygulamanız otomatik olarak Eureka Serverine kaydolur. Aşağıdaki örnekte gösterildiği gibi, Eureka serverini bulmak için yapılandırma gereklidir:

application.yml

```
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

Önceki örnekte **defaultZone**, bir tercih ifade etmeyen herhangi bir client için hizmet URL'sini sağlayan sihirli bir dize yedek değeridir (başka bir deyişle, kullanışlı bir varsayılandır).

defaultZone özelliği büyük/küçük harfe duyarlıdır ve **serviceUrl** özelliği bir **Map<String, String>** olduğundan büyük/küçük harf gerektirir. Bu nedenle, **defaultZone** özelliği, **defaultZone**'in normal Spring Boot snake-case kuralına uymaz.

Varsayılan uygulama adı (yani hizmet kimliği), sanal ana bilgisayar ve güvenli olmayan bağlantı noktası (**Environment**'ten alınan) **${spring.application.name}**, **${spring.application.name}** ve **${server.port}**, sırasıyla.

spring-cloud-starter-netflix-eureka-client'ın sınıf yolunda olması, uygulamayı hem bir Eureka "örneği" (yani kendisini kaydeder) hem de bir " client" (diğer hizmetleri bulmak için kayıt defterini sorgulayabilir) yapar. Örnek davranışı **eureka.instance.*** yapılandırma anahtarları tarafından yönlendirilir, ancak uygulamanızın **spring.application.name** için bir değere sahip olmasını sağlarsanız varsayılanlar uygundur (bu, Eureka hizmet kimliği veya VIP için varsayılandır).

Yapılandırılabilir seçenekler hakkında daha fazla ayrıntı için **EurekaInstanceConfigBean** ve **EurekaClientConfigBean**'e bakın.

Eureka Discovery Client'ı devre dışı bırakmak için **eureka.client.enabled** öğesini **false** olarak ayarlayabilirsiniz.
spring.cloud.discovery.enabled false olarak ayarlandığında da Eureka Discovery Client devre dışı bırakılır.

Spring Cloud Netflix Eureka serverinin sürümünün yol parametresi olarak belirtilmesi şu anda desteklenmemektedir. Bu, bağlam yolundaki (**eurekaServerURLContext**) sürümü ayarlayamayacağınız anlamına gelir. Bunun yerine, sürümü server URL'sine dahil edebilirsiniz (örneğin, **defaultZone: localhost:8761/eureka/v2**'yi ayarlayabilirsiniz).

1.3. Authenticating with the Eureka Server(Eureka Serveri ile Kimlik Doğrulama)

eureka.client.serviceUrl.defaultZone URL'lerinden birinin içinde gömülü kimlik bilgileri varsa, HTTP temel kimlik doğrulaması eureka istemcinize otomatik olarak eklenir (kıvrım stili, şu şekilde: **user:password@localhost:8761/eureka**). Daha karmaşık ihtiyaçlar için, **DiscoveryClientOptionalArgs** türünde bir **@Bean** oluşturabilir ve tümü istemciden sunucuya yapılan çağrılara uygulanan **ClientFilter** örneklerini buna enjekte edebilirsiniz.

Eureka sunucusu, kimlik doğrulama için istemci tarafı sertifikası gerektirdiğinde, client(alıcı) tarafı sertifikası ve güven repository, aşağıdaki örnekte gösterildiği gibi özellikler aracılığıyla yapılandırılabilir:

application.yml

```
eureka:
  client:
    tls:
      enabled: true
      key-store: <path-of-key-store>
      key-store-type: PKCS12
      key-store-password: <key-store-password>
      key-password: <key-password>
      trust-store: <path-of-trust-store>
      trust-store-type: PKCS12
      trust-store-password: <trust-store-password>
```

Eureka clien(alıcı) tarafı TLS'yi etkinleştirmek için **eureka.client.tls.enabled** öğesinin true olması gerekir. **eureka.client.tls.trust-store** atlandığında, bir JVM varsayılan güven deposu kullanılır. **eureka.client.tls.key-store-type** ve **eureka.client.tls.trust-store-type** için varsayılan değer PKCS12'dir. Parola özellikleri atlandığında, boş parola olduğu varsayılır.

Eureka HTTP client(alıcı) tarafından kullanılan RestTemplate'i özelleştirmek istiyorsanız, bir **EurekaClientHttpRequestFactorySupplier** çekirdeği oluşturmak ve bir **ClientHttpRequestFactory** örneği oluşturmak için kendi mantığınızı sağlamak isteyebilirsiniz.

Eureka'daki bir sınırlama nedeniyle, server başına temel kimlik doğrulama bilgilerini desteklemek mümkün değildir, bu nedenle yalnızca bulunan ilk küme kullanılır.

1.4. Status Page and Health Indicator (Durum Sayfası ve Sağlık Göstergesi)

Bir Eureka örneği için durum sayfası ve sistem durumu göstergeleri, bir Spring Boot Actuator uygulamasındaki yararlı uç noktaların varsayılan konumları olan sırasıyla **/info** ve **/health** şeklindedir. Varsayılan olmayan bir bağlam yolu veya server uygulaması yolu (**server.servletPath=/custom** gibi) kullanıyorsanız, bunları bir Aktüatör uygulaması için bile değiştirmeniz gerekir. Aşağıdaki örnek, iki ayar için varsayılan değerleri gösterir:

application.yml

```
eureka:
  instance:
    statusPageUrlPath: ${server.servletPath}/info
    healthCheckUrlPath: ${server.servletPath}/health
```

Bu bağlantılar, clientler(alıcılar) tarafından tüketilen meta verilerde görünür ve bazı senaryolarda uygulamanıza istek gönderip göndermemeye karar vermek için kullanılır, bu nedenle doğru olmaları yararlı olur.

Dalston'da, o yönetim bağlam yolunu değiştirirken durum ve sağlık kontrolü URL'lerinin ayarlanması da gerekiyordu. Bu gereksinim, Edgware'den başlayarak kaldırıldı.

1.5. Registering a Secure Application (Güvenli Bir Uygulama Kaydetme)

Uygulamanız HTTPS üzerinden iletişim kurulmasını istiyorsa, **EurekaInstanceConfigBean**'de iki bayrak ayarlayabilirsiniz:

- **eureka.instance.[nonSecurePortEnabled]=[false]**
- **eureka.instance.[securePortEnabled]=[true]**

Bunu yapmak, Eureka'nın güvenli iletişim için açık bir tercih gösteren örnek bilgilerini yayınlamasını sağlar. Spring Cloud **DiscoveryClient**, bu şekilde yapılandırılmış bir hizmet için her zaman https ile başlayan bir URI döndürür. Benzer şekilde, bir hizmet bu şekilde yapılandırıldığında, Eureka (**native**) örnek bilgilerinin güvenli bir durum denetimi URL'si olur.

Eureka'nın dahili olarak çalışma şekli nedeniyle, siz bunları açıkça geçersiz kılmadığınız sürece, durum ve ana sayfalar için güvenli olmayan bir URL yayınlamaya devam eder. Aşağıdaki örnekte gösterildiği gibi eureka örneği URL'lerini yapılandırmak için yer tutucuları kullanabilirsiniz:

application.yml

```
eureka:
  instance:
    statusPageUrl: https://${eureka.hostname}/info
    healthCheckUrl: https://${eureka.hostname}/health
    homePageUrl: https://${eureka.hostname}/
```

${eureka.hostname}** öğesinin yalnızca Eureka'nın sonraki sürümlerinde kullanılabilen yerel bir yer tutucu olduğunu unutmayın. Aynı şeyi Spring yer tutucularıyla da gerçekleştirebilirsiniz — örneğin, **${eureka.instance.hostName} kullanarak.

Uygulamanız bir proxy arkasında çalışıyorsa ve SSL sonlandırması proxy'deyse (örneğin, Cloud Foundry'de veya hizmet olarak diğer platformlarda çalıştırıyorsanız), proxy "iletilen" başlıklarının ele geçirildiğinden ve işlendiğinden emin olmanız gerekir. uygulama tarafından. Spring Boot uygulamasına gömülü Tomcat kapsayıcısının 'X-Forwarded-*' üstbilgileri için açık yapılandırması varsa, bu otomatik olarak gerçekleşir. Uygulamanız tarafından kendisine verilen bağlantıların yanlış olması (yanlış ana bilgisayar, bağlantı noktası veya protokol), bu yapılandırmayı yanlış yaptığınızın bir işaretidir.

1.6. Eureka’s Health Checks (Eureka'nın Sağlık Kontrolleri)

Varsayılan olarak, Eureka bir client’ın(alıcının) çalışıp çalışmadığını belirlemek için client kalp atışını kullanır. Aksi belirtilmedikçe, Discovery Client, Spring Boot Actuator'a göre uygulamanın mevcut sağlık kontrolü durumunu yaymaz. Sonuç olarak, başarılı bir kayıttan sonra, Eureka her zaman uygulamanın 'UP' durumunda olduğunu duyurur. Bu davranış, uygulama durumunun Eureka'ya yayılmasıyla sonuçlanan Eureka sağlık kontrolleri etkinleştirilerek değiştirilebilir. Sonuç olarak, diğer tüm uygulamalar 'UP' dışındaki durumlardaki uygulamalara trafik göndermez. Aşağıdaki örnek, client için sistem durumu denetimlerinin nasıl etkinleştirileceğini gösterir:

application.yml

```
eureka:
  client:
    healthcheck:
      enabled: true
```

eureka.client.healthcheck.enabled=true yalnızca **application.yml** içinde ayarlanmalıdır. Değeri **bootstrap.yml**'de ayarlamak, Eureka’da **UNKNOWN** bir durumla kaydolma gibi istenmeyen yan etkilere neden olur.

Durum denetimleri üzerinde daha fazla denetime ihtiyacınız varsa, kendi **com.netflix.appinfo.HealthCheckHandler**'ınızı uygulamayı düşünün.

1.7. Eureka Metadata for Instances and Clients

Eureka meta verilerinin nasıl çalıştığını anlamak için biraz zaman harcamaya değer, böylece onu platformunuzda anlamlı bir şekilde kullanabilirsiniz.
Ana bilgisayar adı, IP adresi, bağlantı noktası numaraları, durum sayfası ve sağlık denetimi gibi bilgiler için standart meta veriler vardır. Bunlar hizmet kayıt defterinde yayınlanır ve client'ler(alıcılar) tarafından hizmetlerle doğrudan iletişim kurmak için kullanılır.
eureka.instance.metadataMap'teki örnek kaydına ek meta veriler eklenebilir ve bu meta verilere uzak client'lardan(alıcılardan) erişilebilir. Genel olarak, ek meta veriler, client meta verilerin anlamından haberdar edilmedikçe, client'ın (alıcının) davranışını değiştirmez.
Spring Cloud'un zaten meta veri haritasına anlam atadığı, bu belgede daha sonra açıklanan birkaç özel durum vardır.

1.7.1. Using Eureka on Cloud Foundry (Eureka'yı Cloud Foundry'de Kullanma)

Cloud Foundry, aynı uygulamanın tüm örneklerinin aynı ana bilgisayar adına sahip olması için küresel bir yönlendiriciye sahiptir (benzer mimariye sahip diğer PaaS çözümleri aynı düzenlemeye sahiptir). Bu, Eureka'yı kullanmak için mutlaka bir engel değildir. Ancak, yönlendiriciyi kullanıyorsanız (platformunuzun kurulum şekline bağlı olarak önerilir veya hatta zorunludur), yönlendiriciyi kullanmaları için ana bilgisayar adını ve bağlantı noktası numaralarını (güvenli veya güvenli olmayan) açıkça ayarlamanız gerekir. Client'taki (alıcıdaki) örnekler arasında ayrım yapabilmek için örnek meta verilerini de kullanmak isteyebilirsiniz (örneğin, özel bir yük dengeleyicide). Varsayılan olarak, **eureka.instance.instanceId**, aşağıdaki örnekte gösterildiği gibi **vcap.application.instance_id**'dir:

application.yml

```
eureka:
  instance:
    hostname: ${vcap.application.uris[0]}
    nonSecurePort: 80
```

Cloud Foundry bulut sunucunuzda güvenlik kurallarının ayarlanma şekline bağlı olarak, doğrudan hizmetten hizmete çağrılar için ana makine sanal makinesinin IP adresini kaydedebilir ve kullanabilirsiniz. Bu özellik, Pivotal Web Hizmetleri'nde (PWS) henüz mevcut değildir.

1.7.2. Using Eureka on AWS (AWS'de Eureka'yı kullanma)

Uygulamanın bir AWS Cloud'un dağıtılması planlanıyorsa, Eureka bulut sunucusu AWS'den haberdar olacak şekilde yapılandırılmalıdır. Bunu, **EurekaInstanceConfigBean**'i aşağıdaki gibi özelleştirerek yapabilirsiniz:

application.yml

```
@Bean
@Profile("!default")
public EurekaInstanceConfigBean eurekaInstanceConfig(InetUtils
inetUtils) {
  EurekaInstanceConfigBean bean = new
EurekaInstanceConfigBean(inetUtils);
  AmazonInfo info =
AmazonInfo.Builder.newBuilder().autoBuild("eureka");
  bean.setDataCenterInfo(info);
  return bean;
}
```

1.7.3. Changing the Eureka Instance ID (Eureka Örnek Kimliğini Değiştirme)

Bir vanilya Netflix Eureka örneği, ana bilgisayar adına eşit bir kimlikle kaydedilir (yani, ana bilgisayar başına yalnızca bir hizmet vardır). Spring Cloud Eureka, aşağıdaki gibi tanımlanan mantıklı bir varsayılan sağlar:
${spring.cloud.client.hostname}:${spring.application.name}:${spring.application.instance_id:${server.port}}

Bir örnek **myhost:myappname:8080**'dir.
Spring Cloud kullanarak, aşağıdaki örnekte gösterildiği gibi **eureka.instance.instanceId** içinde benzersiz bir tanımlayıcı sağlayarak bu değeri geçersiz kılabilirsiniz:

application.yml

```
eureka:
  instance:
    instanceId: ${spring.application.name}:${vcap.application.instance_id:${spring.application.instance_id:${random.value}}}
```

Önceki örnekte gösterilen meta veriler ve localhost'ta dağıtılan birden çok hizmet örneğiyle, örneği benzersiz kılmak için buraya rastgele değer eklenir. Cloud Foundry'de **vcap.application.instance_id**, Spring Boot uygulamasında otomatik olarak doldurulur, bu nedenle rastgele değere gerek yoktur.

1.8. Using the EurekaClient (EurekaClient'ı kullanma)

Keşif istemcisi olan bir uygulamanız olduğunda, bunu Eureka Server'den hizmet örneklerini keşfetmek için kullanabilirsiniz. Bunu yapmanın bir yolu, aşağıdaki örnekte gösterildiği gibi yerel **com.netflix.discovery.EurekaClient**'i (Spring Cloud **DiscoveryClient**'in aksine) kullanmaktır:

```
@Autowired
private EurekaClient discoveryClient;

public String serviceUrl() {
    InstanceInfo instance =
discoveryClient.getNextServerFromEureka("STORES", false);
    return instance.getHomePageUrl();
}
```

EurekaClient'i bir **@PostConstruct** metodunda veya **@Scheduled** metodunda (veya **ApplicationContext**'in henüz başlatılmamış olabileceği herhangi bir yerde) kullanmayın. Bir **SmartLifecycle**'da (**phase=0** ile) başlatılır, bu nedenle mevcut olduğuna en erken güvenebileceğiniz daha yüksek bir aşamaya sahip başka bir **SmartLifecycle**'dadır.

1.8.1. EurekaClient with Jersey (Jersey ile EurekaClient)

Varsayılan olarak, EurekaClient, HTTP iletişimi için Spring'in **RestTemplate**'ini kullanır. Bunun yerine Jersey'i kullanmak istiyorsanız, Jersey bağımlılıklarını sınıf yolunuza eklemeniz gerekir. Aşağıdaki örnek, eklemeniz gereken bağımlılıkları gösterir:

```
<dependency>
    <groupId>com.sun.jersey</groupId>
    <artifactId>jersey-client</artifactId>
</dependency>
<dependency>
    <groupId>com.sun.jersey</groupId>
    <artifactId>jersey-core</artifactId>
</dependency>
<dependency>
    <groupId>com.sun.jersey.contribs</groupId>
    <artifactId>jersey-apache-client4</artifactId>
</dependency>
```

1.9. Alternatives to the Native Netflix EurekaClient

Ham Netflix **EurekaClient** kullanmanıza gerek yoktur. Ayrıca, genellikle bir tür sargının arkasında kullanmak daha uygundur. Spring Cloud, fiziksel URL'ler yerine mantıksal Eureka hizmet tanımlayıcıları (VIP'ler) aracılığıyla Feign (bir REST client(alıcı) oluşturucu) ve Spring **RestTemplate** desteğine sahiptir.

Aşağıdaki örnekte gösterildiği gibi, keşif client'leri için basit bir API (Netflix'e özgü değildir) sağlayan **org.springframework.cloud.client.discovery.DiscoveryClient**'i de kullanabilirsiniz:

```
@Autowired
private DiscoveryClient
discoveryClient;

public String serviceUrl() {
    List<ServiceInstance> list =
discoveryClient.getInstances("STORE
S");
    if (list != null && list.size() > 0 ) {
        return list.get(0).getUri();
    }
    return null;
}
```

1.10. Why Is It so Slow to Register a Service? (Bir Hizmeti Kaydetmek Neden Bu Kadar Yavaş?)

Örnek olmak, varsayılan süresi 30 saniye olan kayıt defterine (client'ın(alıcının) **serviceUrl**'si aracılığıyla) periyodik bir sinyal göndermeyi de içerir. Örnek, server ve client'ının tümü yerel önbelleklerinde aynı meta verilere sahip olana kadar (bu nedenle 3 kalp atışı sürebilir) bir hizmet client'lar tarafından keşfedilemez. **eureka.instance.leaseRenewalIntervalInSeconds** ayarını yaparak dönemi değiştirebilirsiniz. 30'dan daha düşük bir değere ayarlamak, client'ları(alıcıları) diğer hizmetlere bağlama sürecini hızlandırır. Üretimde, serverdeki kiralama yenileme süresi hakkında varsayımlarda bulunan dahili hesaplamalar nedeniyle varsayılana bağlı kalmak muhtemelen daha iyidir.

1.11. Zones (Bölgeler)

Eureka Client'leri birden çok bölgeye dağıttıysanız, başka bir bölgedeki hizmetleri denemeden önce bu istemcilerin aynı bölgedeki hizmetleri kullanmasını tercih edebilirsiniz. Bunu ayarlamak için Eureka istemcilerinizi doğru şekilde yapılandırmanız gerekir.
İlk olarak, her bölgeye dağıtılmış Eureka sunucularınız olduğundan ve bunların birbirinin eşi olduğundan emin olmanız gerekir. Daha fazla bilgi için zones ve regions bölümüne bakın.
Ardından, hizmetinizin hangi bölgede olduğunu Eureka'ya söylemeniz gerekiyor. Bunu **metadataMap** özelliğini kullanarak yapabilirsiniz.
Örneğin, **Service 1** hem **Zone 1** hem de **Zone 2**'ye dağıtılmışsa, **Service 1**'de aşağıdaki Eureka özelliklerini ayarlamanız gerekir:

Service 1 in Zone 1 (Bölge 1'de Hizmet 1)

```
eureka.instance.metadataMap.zone = zone1
eureka.client.preferSameZoneEureka = true
```

Service 1 in Zone 2 (Bölge 2'de Hizmet 1)

```
eureka.instance.metadataMap.zone = zone2
eureka.client.preferSameZoneEureka = true
```

1.12. Refreshing Eureka Clients

Varsayılan olarak, **EurekaClient** çekirdeği yenilenebilir, yani Eureka Client özellikleri değiştirilebilir ve yenilenebilir. Bir yenileme gerçekleştiğinde, client'ların(alıcıların) Eureka serverindeki (sunucusundaki) kaydı silinecektir ve belirli bir hizmetin tüm örneklerinin kullanılamadığı kısa bir süre olabilir. Bunu ortadan kaldırmanın bir yolu, Eureka Client'larını yenileme özelliğini devre dışı bırakmaktır. Bunu yapmak için **eureka.client.refresh.enable=false** ayarlayın.

1.13. Using Eureka with Spring Cloud LoadBalancer

Spring Cloud LoadBalancer **ZonePreferenceServiceInstanceListSupplier** için destek sunuyoruz. Eureka örneği meta verilerinden (**eureka.instance.metadataMap.zone**) **Zone** değeri, hizmet örneklerini bölgeye göre filtrelemek için kullanılan **spring-cloud-loadbalancer-zone** özelliğinin değerini ayarlamak için kullanılır.

Bu eksikse ve **spring.cloud.loadbalancer.eureka.aboutZoneFromHostname** bayrağı **true** olarak ayarlanmışsa, bölge için bir proxy olarak sunucu ana bilgisayar adındaki alan adını kullanabilir.

Başka bir bölge verisi kaynağı yoksa, istemci konfigürasyonuna göre (örnek konfigürasyonunun aksine) bir tahmin yapılır. Bölge adından bölge listesine bir harita olan **eureka.client.availabilityZones**'u alırız ve örneğin kendi bölgesi için ilk bölgeyi çıkarırız (yani, varsayılan olarak "us-east-1" olan **eureka.client.region**), yerel Netflix ile uyumluluk için).

2. Service Discovery: Eureka Server

Bu bölümde bir Eureka Serveri'nin nasıl kurulacağı açıklanmaktadır.

2.1. How to Include Eureka Server (Eureka Serveri Nasıl Dahil Edilir)

Eureka Server'ı projenize dahil etmek için, **org.springframework.cloud** grup kimliğiyle ve **spring-cloud-starter-netflix-eureka-server** yapı kimliğiyle başlatıcıyı kullanın. Mevcut Spring Cloud Release Train ile derleme sisteminizi kurma hakkında ayrıntılar için Spring Cloud Project sayfasına bakın.

Projeniz şablon motoru olarak Thymeleaf kullanıyorsa, Eureka serverinin Freemarker şablonları doğru yüklenmeyebilir. Bu durumda şablon yükleyiciyi manuel olarak yapılandırmak gerekir:

application.yml

```
spring:
  freemarker:
    template-loader-path:
classpath:/templates/
    prefer-file-system-access: false
```

2.2. How to Run a Eureka Server (Eureka Serveri Nasıl Çalıştırılır)

Aşağıdaki örnek, minimal bir Eureka serverini göstermektedir:

```
@SpringBootApplication
@EnableEurekaServer
public class Application {

    public static void main(String[] args) {
        new SpringApplicationBuilder(Application.class).web(true).run(args);
    }

}
```

Serverin, **/eureka/*** altındaki normal Eureka işlevselliği için bir UI ve HTTP API uç noktalarına sahip bir ana sayfası vardır. Aşağıdaki bağlantılarda bazı Eureka arka plan okumaları vardır: akı kondansatörü ve google grup tartışması.

Spring-cloud-starter-netflix-eureka-server'a bağlı olarak Gradle'ın bağımlılık çözümleme kuralları ve bir ana bomba özelliğinin olmaması nedeniyle, uygulama başlatılırken hatalara neden olabilir. Bu sorunu çözmek için Spring Boot Gradle eklentisini ekleyin ve Spring cloud starter ana bombasını aşağıdaki gibi içe aktarın:

build.gradle

```
buildscript {
  dependencies {
    classpath("org.springframework.boot:spring-boot-gradle-plugin:{spring-boot-docs-version}")
  }
}

apply plugin: "spring-boot"

dependencyManagement {
  imports {
    mavenBom "org.springframework.cloud:spring-cloud-dependencies:{spring-cloud-version}"
  }
}
```

2.3. defaultOpenForTrafficCount and its effect on EurekaServer warmup time

Spring Cloud Eureka serverinde Netflix Eureka'nın **waitTimeInMsWhenSyncEmpty** ayarı başlangıçta dikkate alınmaz. Isınma süresini etkinleştirmek için **eureka.server.defaultOpenForTrafficCount=0** olarak ayarlayın.

2.4. High Availability, Zones and Regions

Eureka serverinin bir arka uç deposu yoktur, ancak kayıt defterindeki hizmet örneklerinin tümü, kayıtlarını güncel tutmak için sinyal göndermek zorundadır (böylece bu, bellekte yapılabilir). Client'lar(alıcılar) ayrıca Eureka kayıtlarının bir bellek içi önbelleğine sahiptir (böylece bir hizmete yönelik her istek için kayıt defterine gitmeleri gerekmez).

Varsayılan olarak, her Eureka serveri aynı zamanda bir Eureka client'idir(alıcısıdır) ve bir eş bulmak için (en az bir) hizmet URL'si gerektirir. Bunu sağlamazsanız, hizmet çalışır ve çalışır, ancak günlüklerinizi peer'e(eşe) kayıt olamama konusunda çok fazla gürültü ile doldurur.

2.5. Standalone Mode (Bağımsız Mod)

İki önbelleğin (client ve server) ve kalp atışlarının birleşimi, onu canlı tutan bir tür monitör veya esnek çalışma zamanı (Cloud Foundry gibi) olduğu sürece bağımsız bir Eureka serverini arızaya karşı oldukça dayanıklı kılar. Bağımsız modda, emsallerine ulaşmayı denemeye ve başarısızlığa uğramaması için client(alıcı) tarafı davranışını kapatmayı tercih edebilirsiniz. Aşağıdaki örnek, client(alıcı) tarafı davranışının nasıl kapatılacağını gösterir:

application.yml (Standalone Eureka Server)

```
server:
  port: 8761

eureka:
  instance:
    hostname: localhost
  client:
    registerWithEureka: false
    fetchRegistry: false
    serviceUrl:
      defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
```

serviceUrl'nin yerel örnekle aynı ana bilgisayarı gösterdiğine dikkat edin.

2.6. Peer Awareness (Akran Farkındalığı)

Eureka, birden çok örnek çalıştırarak ve onlardan birbirlerine kaydolmalarını isteyerek daha esnek ve kullanılabilir hale getirilebilir. Aslında, bu varsayılan davranıştır, bu nedenle çalışması için yapmanız gereken tek şey, aşağıdaki örnekte gösterildiği gibi bir eşe geçerli bir **serviceUrl** eklemektir:

application.yml (Two Peer Aware Eureka Servers)

```
---
spring:
  profiles: peer1
eureka:
  instance:
    hostname: peer1
  client:
    serviceUrl:
      defaultZone: https://peer2/eureka/

---
spring:
  profiles: peer2
eureka:
  instance:
    hostname: peer2
  client:
    serviceUrl:
      defaultZone: https://peer1/eureka/
```

Önceki örnekte, aynı serveri farklı Spring profillerinde çalıştırarak iki ana bilgisayarda (**peer1**-eş1 ve **peer2**-eş2) çalıştırmak için kullanılabilecek bir YAML dosyamız var. Bu yapılandırmayı, ana bilgisayar adlarını çözümlemek için **/etc/hosts** öğesini değiştirerek tek bir ana bilgisayarda (bunu üretimde yapmanın pek bir değeri yoktur) peer(eş) farkındalığını test etmek için kullanabilirsiniz. Aslında, kendi ana bilgisayar adını bilen bir makinede çalışıyorsanız **eureka.instance.hostname** gerekli değildir (varsayılan olarak **java.net.InetAddress** kullanılarak aranır).

Bir sisteme birden fazla eş ekleyebilirsiniz ve hepsi en az bir uç ile birbirine bağlı olduğu sürece kayıtları kendi aralarında senkronize eder. Eşler fiziksel olarak ayrılırsa (bir veri merkezi içinde veya birden fazla veri merkezi arasında), sistem prensipte "bölünmüş beyin" tipi arızalardan kurtulabilir. Bir sisteme birden fazla eş ekleyebilirsiniz ve hepsi birbirine doğrudan bağlı olduğu sürece, kayıtları kendi aralarında senkronize edeceklerdir.

application.yml (Three Peer Aware Eureka Servers)

```
eureka:
  client:
    serviceUrl:
      defaultZone: https://peer1/eureka/,http://peer2/eureka/,http://peer3/eureka/

---
spring:
  profiles: peer1
eureka:
  instance:
    hostname: peer1

---
spring:
  profiles: peer2
eureka:
  instance:
    hostname: peer2

---
spring:
  profiles: peer3
eureka:
  instance:
    hostname: peer3
```

2.7. When to Prefer IP Address (IP Adresi Ne Zaman Tercih Edilir?)

Bazı durumlarda, Eureka'nın ana bilgisayar adı yerine hizmetlerin IP adreslerinin reklamını yapması tercih edilir. **eureka.instance.preferIpAddress** öğesini **true** olarak ayarlayın ve uygulama eureka'ya kaydolduğunda, ana bilgisayar adı yerine IP adresini kullanır.

Eğer ana bilgisayar adı Java tarafından belirlenemezse, IP adresi Eureka'ya gönderilir. Ana bilgisayar adını ayarlamanın tek açık yolu, **eureka.instance.hostname** özelliğini ayarlamaktır. Ana bilgisayar adınızı çalışma zamanında bir ortam değişkeni kullanarak ayarlayabilirsiniz; örneğin, **eureka.instance.hostname=${HOST_NAME}**.

2.8. Securing The Eureka Server (Eureka Sunucusunun Güvenliğini Sağlama)

spring-boot-starter-security aracılığıyla serverinizin sınıf yoluna Spring Security ekleyerek Eureka serverinizi güvenli hale getirebilirsiniz. Spring Security sınıf yolundayken varsayılan olarak, uygulamaya yapılan her istekte geçerli bir CSRF belirtecinin gönderilmesini gerektirir. Eureka client'ları(alıcıları) genellikle geçerli bir siteler arası istek sahteciliği (CSRF) belirtecine sahip olmayacaktır, bu gereksinimi **/eureka/**** uç noktaları için devre dışı bırakmanız gerekecektir. Örneğin:

```
@EnableWebSecurity
class WebSecurityConfig extends WebSecurityConfigurerAdapter {

   @Override
   protected void configure(HttpSecurity http) throws Exception {
      http.csrf().ignoringAntMatchers("/eureka/**");
      super.configure(http);
   }
}
```

CSRF hakkında daha fazla bilgi için Spring Security belgelerine bakın.
Spring Cloud Samples repository'sinde bir demo Eureka Server'i bulunabilir.

2.9. JDK 11 Support

Eureka serverinizin bağlı olduğu JAXB modülleri JDK 11'de kaldırılmıştır. Bir Eureka sunucusu çalıştırırken JDK 11 kullanmayı düşünüyorsanız, bu bağımlılıkları POM veya Gradle dosyanıza eklemelisiniz.

```
<dependency>
  <groupId>org.glassfish.jaxb</groupId>
  <artifactId>jaxb-runtime</artifactId>
</dependency>
```

3. Configuration properties

Spring Cloud Netflix ile ilgili tüm yapılandırma özelliklerinin listesini görmek için lütfen Ek sayfasını kontrol edin. (Bir sonraki sayfada bu ek içeriğinden bahsedilmektedir.)

3. Configuration properties

EK: Common application properties

application.properties dosyanızın içinde, **application.yml** dosyanızın içinde veya komut satırı anahtarları olarak çeşitli özellikler belirtilebilir. Bu ek, yaygın Spring Cloud Netflix özelliklerinin bir listesini ve bunları tüketen temel sınıflara referanslar sağlar. Özellik katkıları, sınıf yolunuzdaki ekjar dosyalarından gelebilir, bu nedenle bunu kapsamlı bir liste olarak düşünmemelisiniz. Ayrıca, kendi özelliklerinizi tanımlayabilirsiniz.

Name	Default	Description
eureka.client.eureka-connection-idle-timeout-seconds	30	eureka sunucusuna yapılan HTTP bağlantılarının kapatılmadan önce ne kadar süre (saniye olarak) boşta kalabileceğini gösterir. AWS ortamında, güvenlik duvarı bağlantı bilgilerini bir kaç dakika sonra temizlediğinden, bağlantıyı belirsizlik içinde bıraktığından, değerlerin 30 saniye veya daha az olması önerilir.
eureka.client.eureka-server-connect-timeout-seconds	5	eureka sunucusuna bir bağlantının zaman aşımına uğraması gerekmeden önce ne kadar bekleneceğini (saniye olarak) gösterir. İstemcideki bağlantıların org.apache.http.client.HttpClient tarafından havuza alındığını ve bu ayarın gerçek bağlantı oluşturmayı ve ayrıca havuzdan bağlantı almak için bekleme süresini etkilediğini unutmayın.
eureka.client.eureka-server-d-n-s-name		Eureka sunucularının listesini almak için sorgulanacak DNS adını alır. Sözleşme serviceUrl'leri uygulayarak hizmet url'lerini döndürürse bu bilgi gerekli değildir. DNS mekanizması, useDnsForFetchingServiceUrls true olarak ayarlandığında ve eureka istemcisi, değişen eureka sunucularını dinamik olarak getirebilmesi için DNS'nin belirli bir şekilde yapılandırılmasını beklediğinde kullanılır. Değişiklikler çalışma zamanında etkilidir.
eureka.client.eureka-server-port		eureka sunucularının listesi DNS'den geldiğinde eureka sunucusuyla iletişim kurmak için hizmet url'sini oluşturmak için kullanılacak bağlantı noktasını alır. Sözleşme eurekaServerServiceUrls(String) hizmet url'lerini döndürürse bu bilgi gerekli değildir. DNS mekanizması, useDnsForFetchingServiceUrls true olarak ayarlandığında ve eureka istemcisi, değişen eureka sunucularını dinamik olarak getirebilmesi için DNS'nin belirli bir şekilde yapılandırılmasını beklediğinde kullanılır. Değişiklikler çalışma zamanında etkilidir.
eureka.client.eureka-server-read-timeout-seconds	8	eureka sunucusundan bir okumanın zaman aşımına uğraması gerekmeden önce ne kadar bekleneceğini (saniye olarak) gösterir.
eureka.client.eureka-server-total-connections	200	eureka istemcisinden tüm eureka sunucularına izin verilen toplam bağlantı sayısını alır.
eureka.client.eureka-server-total-connections-per-host	50	eureka istemcisinden bir eureka sunucusu ana bilgisayarına izin verilen toplam bağlantı sayısını alır.
eureka.client.eureka-server-u-r-l-context		eureka sunucularının listesi DNS'den geldiğinde eureka sunucusuyla iletişim kurmak için hizmet url'sini oluşturmak için kullanılacak URL bağlamını alır. Sözleşme, hizmet url'lerini eurekaServerServiceUrls'den döndürürse bu bilgi gerekli değildir. DNS mekanizması, useDnsForFetchingServiceUrls true olarak ayarlandığında ve eureka istemcisi, değişen eureka sunucularını dinamik olarak getirebilmesi için DNS'nin belirli bir şekilde yapılandırılmasını beklediğinde kullanılır. Değişiklikler çalışma zamanında etkilidir.
eureka.client.eureka-service-url-poll-interval-seconds	0	eureka sunucu bilgilerindeki değişiklikler için ne sıklıkta (saniye cinsinden) yoklama yapılacağını belirtir. Eureka sunucuları eklenebilir veya kaldırılabilir ve bu ayar, eureka istemcilerinin bunu ne kadar sürede bilmesi gerektiğini kontrol eder.
eureka.client.prefer-same-zone-eureka	true	Bu örneğin gecikme ve/veya başka bir nedenle aynı bölgede eureka sunucusunu kullanmaya çalışıp çalışmayacağını belirtir. İdeal olarak eureka istemcileri aynı bölgedeki sunucularla konuşacak şekilde yapılandırılır Değişiklikler, registerFetchIntervalSeconds tarafından belirtildiği gibi sonraki kayıt defteri getirme döngüsünde çalışma zamanında etkilidir
eureka.client.register-with-eureka	true	Bu örneğin başkaları tarafından keşfedilmesi için bilgilerini eureka sunucusuna kaydetmesi gerekip gerekmediğini gösterir. Bazı durumlarda, örneklerinizin keşfedilmesini istemezsiniz, oysa sadece diğer örnekleri keşfetmek istersiniz.
eureka.server.peer-eureka-nodes-update-interval-ms	0	
eureka.server.peer-eureka-status-refresh-time-interval-ms	0	
ribbon.eureka.enabled	true	Ribbon ile Eureka kullanımına olanak sağlar.
spring.cloud.loadbalancer.eureka.approximate-zone-from-hostname	false	Ana makine adından bölge değerini almaya çalışmanız gerekip gerekmediğini belirlemek için kullanılır.