**Spring Cloud, Spring Boot ve Docker kullanarak Mikro Servis Projesi Hazırlamak**

Bu makalede Spring ailesini kullanarak ve çok ayrıntıya girmeden bir mikro servis projesi hazırlayacağız. Başlamadan önce mikro servis nedir neden bu kadar yaygınlaşmaya başlamıştır kısaca ondan bahsetmek istiyorum. Daha ayrıntılı bilgi için <https://microservices.io> adresini ziyaret edebilirsiniz.

Mikro servis, bir yazılım geliştirme tekniğidir. Sadece belirli biri işten sorumlu, ölçeklenebilir, “loose-coupled” yapıya sahip servistir.

Faydalar:

* **Modüler**

Geliştirilmesi, anlaması ve test edilmesi kolay.

* **Ölçeklendirilebilir**

Birbirlerinden bağımsız süreçler olarak çalıştıklarından dolayı ölçeklendirirken daha özgür olabiliriz.

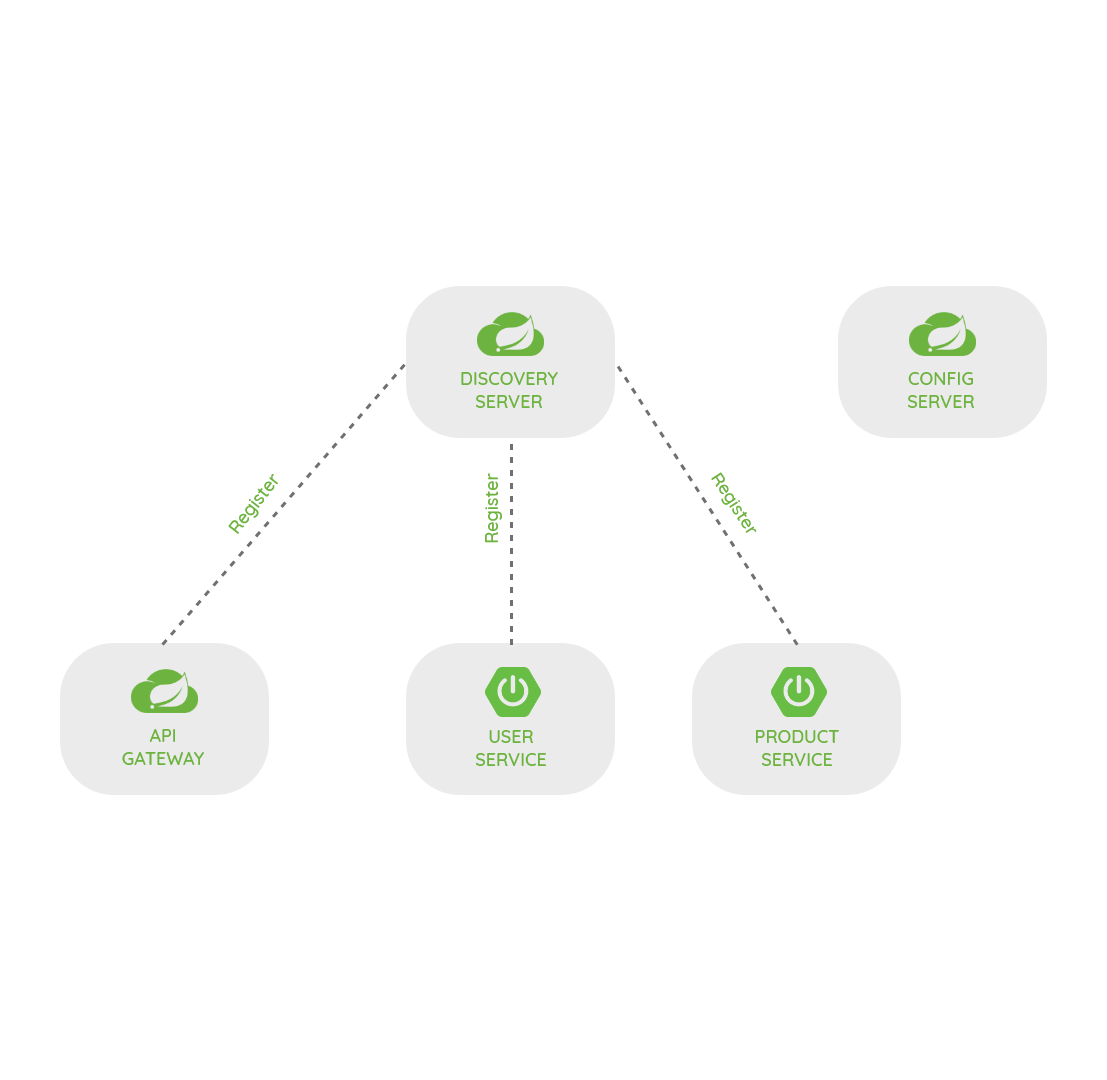
* **Kolay dağıtılabilir**

Küçük ve hızlı projeler oldukları için deploy edilmesi de daha kolay.

Şimdi projemize geçelim. Projemiz aşağıdakiler bileşenleri içermektedir. Yeri geldikçe bileşenler hakkında biraz bilgi ve projedeki görevlerini anlatmaya çalışacağım.

* Discovery Server (Spring Eureka)
* Config Center (Spring Cloud Config)
* API Gateway (Spring Cloud Netfliz - Zuul)
* User Service
* Product Service

Proje sonlandığı zaman elimizde böyle bir ekosistem olacak.



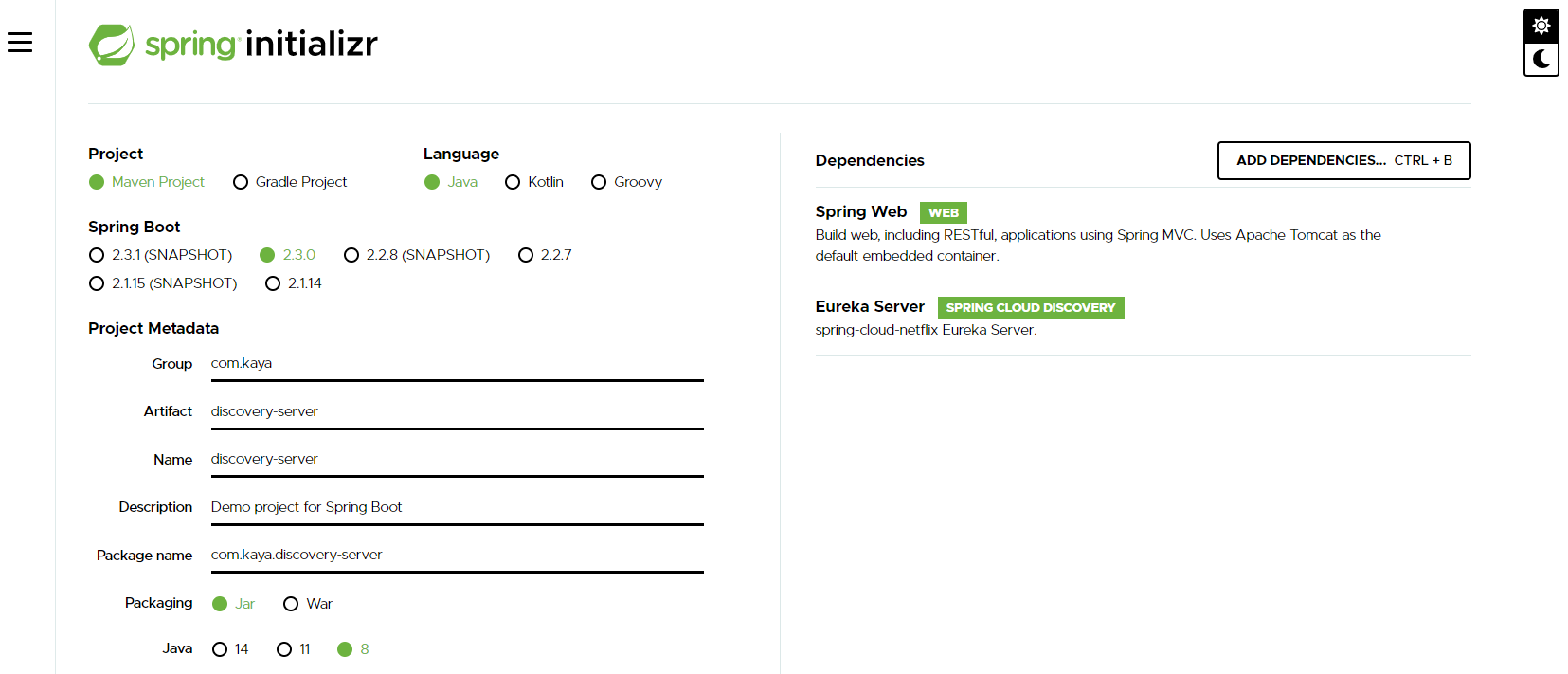
1. **Discovery Server**

Discovery Server için **Spring Eureka** projesini kullanacağız.

Discovery Server’ın amacı kendisine kayıt yapmış servislerin bilgisini tutmaktır. Bir mikro servis ekosistemde birden fazla servis ve bu servislerin birden fazla kopyası bulunabilir. Bu servislerin birbirleriyle bir noktada iletişim kurması gerekecektir. Ama bir servis başka bir servisin hangi ip ve portta çalıştığını bilemez. Discovery servis bu sorunu çözmektedir.

Çalışan her servis, Discovery Server’a kayıt olur. Ve böylelikle Discovery Server hangi servislerin ayakta olduğunu, onların ip ve port bilgileri gibi çeşitli veriyi kendinde saklar.

Spring initializr ([start.spring.io](https://start.spring.io/)) sayfasına girerek projemizi oluşturuyoruz.



Projeyi oluşturduktan sonra **application.properteis** dosyasının içine bazı ayarlar eklememiz gerekiyor.

server.port=5858

eureka.client.registerWithEureka=false

eureka.client.fetchRegistry=false

Daha sonra **DiscoveryServerApplication.java** içine girip. Class’ımıza **@EnableEurekaServer** annotation’ını eklememiz gerekiyor. Böylelikle projemizi bir Eureka Server haline getiriyoruz.

@SpringBootApplication

@EnableEurekaServer

public class DiscoveryServerApplication {

public static void main(String[] args) {

SpringApplication.run(DiscoveryServerApplication.class, args);

}

}

Son adım olarak Discovery Server’ımız dockerize edeceğiz. Böylelikle projemiz Docker’ın kurulu olduğu herhangi bir ortamda rahatlıkla ayağa kalkabilicek ve istersek Kubernetes gibi container-orchestration orkestrasyon toolarıyla server’imizi deploy edebileceğiz.

**Dockerfile** isimli bir dosya oluşturup içine ayarlarımız ekliyoruz.

FROM openjdk:8-jdk-alpine

ARG JAR\_FILE=target/\*.jar

COPY ${JAR\_FILE} app.jar

ENTRYPOINT ["java","-jar","/app.jar"]

Daha sonra aynı dizin içinde docker image’ımızı build ediyoruz.

docker build -t discovery-server-docker .

Artık elimizde projemizin bir image dosyası var. Image’leri çalıştırmanın birden fazla yolu var ancak ben bir docker-compose özelliğini kullanacağım. Böylelikle projemizdeki bütün servislerin image dosyasını oluşturduktan sonra birkaç satır komut ile projemizin tamamını ayağa kaldırabiliceğiz.

docker-compose.yml dosyasını oluşturup içini dolduruyoruz

version: '3'

services:

discovery-service:

image: discovery-server-docker

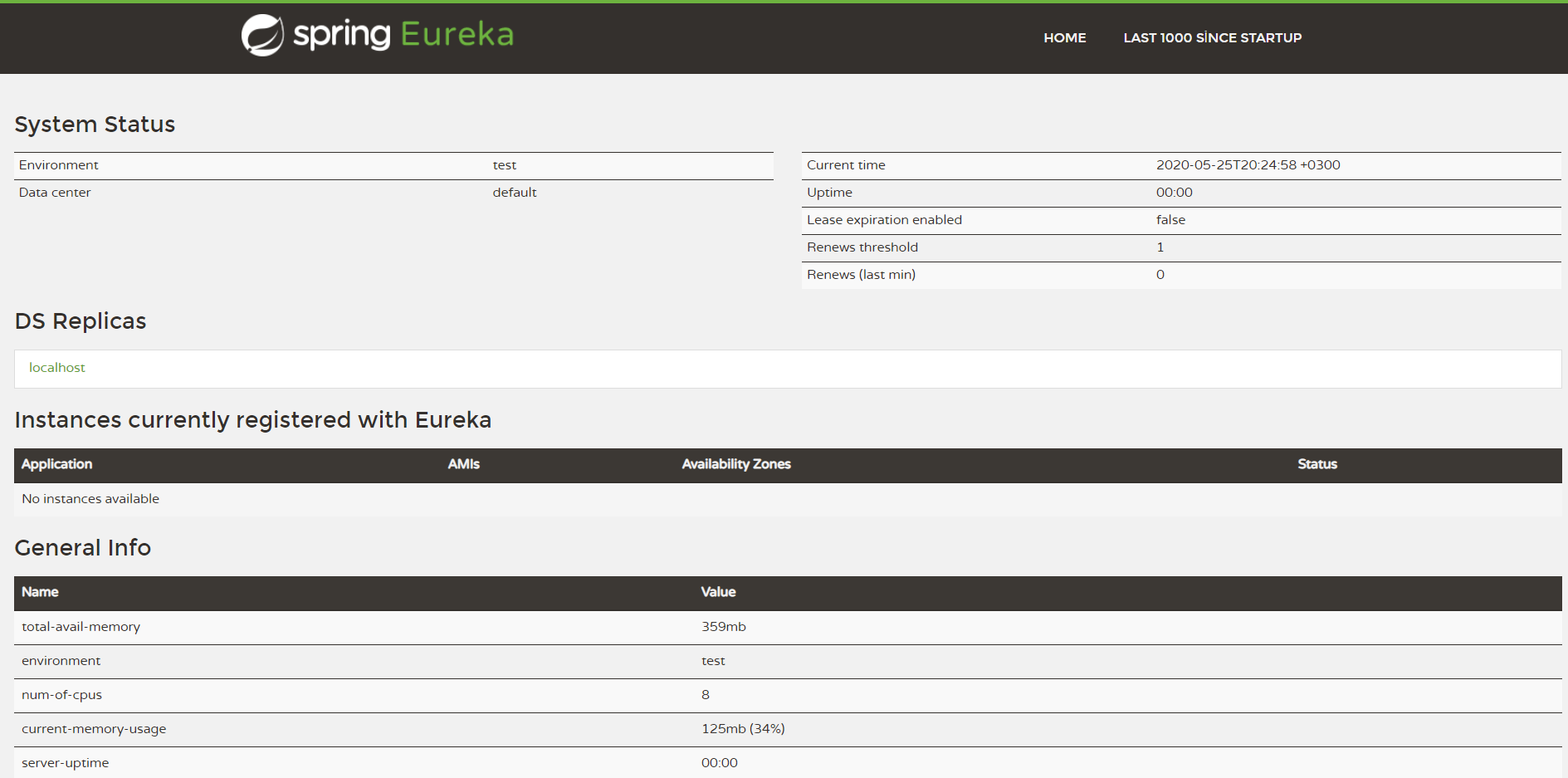
ports:

- 5858:5858

Şimdi docker-compose komutunu kullanarak projemizi ayağa kaldırırıyoruz.

docker-compose up

Sorunsuz bir şekilde çalıştırdıktan sonra <http://localhost:5858> adresinden Discovery Server arayüzüne ulaşabiliriz.



Buradaki **“Instances currently registered with Eureka”** bölümünün altında serverımıza kayıt olmuş servisleri görebiliriz.

Projemizdeki diğer servislerimizi yazmaya başladıktan sonra, burada ekranda servislerimizi göremeye başlayacağız.