PetVerse Gözlemlenebilirlik Kurulum ve Kullanım Kılavuzu

İçindekiler Tablosu

[Değişiklik Tarihçesi 2](#_Toc204615472)

[1 Amaç ve Kapsam 3](#_Toc204615473)

[1.1. Amaç 3](#_Toc204615474)

[1.2 Kapsam 3](#_Toc204615475)

[2. Kullanılan Araçlar 3](#_Toc204615476)

[3. Gereksinimler 4](#_Toc204615477)

[4. Sistem Mimarisi 4](#_Toc204615478)

[5. Kurulum Adımları — *Prometheus & Grafana için* 5](#_Toc204615479)

[5.1. Metrik Açıklamaları 5](#_Toc204615480)

[5.2. Sık Karşılaşılan Hatalar ve Çözümleri 6](#_Toc204615481)

[5.3. Dashboard Kullanımı 6](#_Toc204615482)

[5.3.1. Grafana Dashboard Ekran Görüntüleri 6](#_Toc204615483)

[Grafana Giriş Ekranı 6](#_Toc204615484)

[Grafana Ana Sayfa 7](#_Toc204615485)

[Dashboard Listesi 7](#_Toc204615486)

[HTTP Request Count Paneli 8](#_Toc204615487)

[Memory Usage Paneli 8](#_Toc204615488)

[CPU Kullanımı Paneli 9](#_Toc204615489)

[5.3.2. Yeni Dashboard Oluşturma: ActivityService Örneği 9](#_Toc204615490)

[Yeni Dashboard Oluşturma 9](#_Toc204615491)

[Görselleştirme Ekle 10](#_Toc204615492)

[Veri Kaynağı Seçimi 10](#_Toc204615493)

[Metrik Sorgusu Girme 11](#_Toc204615494)

[Panel Başlığı ve Grafik 11](#_Toc204615495)

[Dashboard Kaydetme 12](#_Toc204615496)

[5.3.3. Mevcut Dashboard'a Yeni Panel Ekleme (Bellek Kullanımı) 12](#_Toc204615497)

[Yeni Panel Ekleme 12](#_Toc204615498)

[Bellek Kullanım Metrik Sorgusu 13](#_Toc204615499)

[5.4. Prometheus Kullanımı ve Durum İzleme 13](#_Toc204615500)

[Prometheus Target Kontrolü 13](#_Toc204615501)

[Prometheus Endpoint Çıktısı 14](#_Toc204615502)

[5.4.1 Kurulumda Yapılan Kod Değişiklikleri 14](#_Toc204615503)

[application.yml Konfigürasyonu 14](#_Toc204615504)

[pom.xml Bağımlılığı 15](#_Toc204615505)

[prometheus.yml Yapılandırması 15](#_Toc204615506)

[docker-compose.yml Dosyasına Eklemeler 16](#_Toc204615507)

[6. Circuit Breaker (Resilience4j) Yapılandırması 16](#_Toc204615508)

[6.1 Circut Breaker Dashboard Görüntülüleri 16](#_Toc204615509)

[6.1.1 Başarılı Çağrı Senaryosu (Circuit Closed) 17](#_Toc204615510)

[6.1.2 Servis Hatası ve Circuit Open Durumu 18](#_Toc204615511)

[6.1.3 Recovery: Half-Open Durumu ve Yeniden Açılma 19](#_Toc204615512)

[6.2 Kurulumda Yapılan Kod Değişiklikleri 21](#_Toc204615513)

[ActivityServiceImpl.java – Fallback Mekanizmasının Eklenmesi 21](#_Toc204615514)

[application.yml eklemeler 23](#_Toc204615515)

[pom.xml dosyasına eklemeler 24](#_Toc204615516)

[7. Logback ile Loglama Yapılandırması 24](#_Toc204615517)

[7.1 Bağımlılık Tanımlaması 24](#_Toc204615518)

[7.2 logback-spring.xml Dosyası Yapılandırması 24](#_Toc204615519)

[7.3 Docker Volume Tanımlaması 25](#_Toc204615520)

[7.4 Log Formatı ve Örneği 25](#_Toc204615521)

[8.Kaynaklar ve Dökümantasyon 25](#_Toc204615522)

[9. Sonuç 25](#_Toc204615523)

# Değişiklik Tarihçesi

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Yayım No** | **Tarih** | **Değişen Sayfa/Tablo/Şekil** | **\*E,**  **S, D** | **Değişikliğin Sahibi** |
| **V1.0** | 17 Temmuz 2025 | Döküman hazırlandı | E | Yağmur BAŞOĞLU |
| **V2.0** | 25 Temmuz 2025 | Döküman güncellendi | D | Yağmur BAŞOĞLU |

\*E: Ekleme S: Silme D: Değiştirme

# 1 Amaç ve Kapsam

# 1.1. Amaç

Bu kılavuz, PetVerse mikroservis projesi kapsamında gözlemlenebilirlik araçlarının (Prometheus, Grafana, Spring Boot Actuator) nasıl kurulduğunu ve sistemin nasıl kullanılacağını detaylı şekilde açıklamak amacıyla hazırlanmıştır.

# 1.2 Kapsam

Bu kılavuz aşağıdaki bileşenlerin kurulumunu ve kullanımını kapsar:  
- Prometheus ile metrik toplama

- Grafana ile veri görselleştirme

- Spring Boot Actuator ile sağlık kontrolleri

- Docker Compose ile servislerin orkestrasyonu

- Dashboard ekran görüntüleri ile sistem izleme

- Logback kullanılarak loglama yapılandırması

- Circuit Breaker (Resilience4j) uygulaması

- API Gateway üzerinde Rate Limiting yapılandırması

# 2. Kullanılan Araçlar

* Prometheus: Metrik verilerinin toplanması için kullanılır.
* Grafana: Toplanan metriklerin görselleştirilmesi için kullanılır.
* Spring Boot Actuator: Her bir mikroservis için hazır health check ve metrik endpoint'leri sağlar.
* Docker Compose: Tüm servislerin ve gözlemlenebilirlik bileşenlerinin ayağa kaldırılması için kullanılır.
* Logback: Mikroservis loglarının oluşturulması ve yapılandırılması için kullanılır.
* Resilience4j: Mikroservisler arası çağrılarda hataya dayanıklı yapı (circuit breaker) sağlamak için kullanılır.

# 3. Gereksinimler

Kuruluma başlamadan önce aşağıdaki yazılımların sisteminizde yüklü olması gerekir:

* Docker & Docker Compose
* Java 21 & Maven
* IDE: IntelliJ IDEA veya Visual Studio Code
* Postman veya benzeri bir API test aracı
* Prometheus (binary ya da Docker container olarak)
* Grafana (binary ya da Docker container olarak)

Ayrıca eğer loglama veya resilience davranışlarını test etmek istiyorsan:

* Slack (opsiyonel, uyarı mesajı göndermek için)
* GitHub hesabı (issue notification senaryoları için, opsiyonel)

# 4. Sistem Mimarisi

PetVerse gözlemlenebilirlik yapısı, mikroservislerin sağlık, performans ve hata durumlarının merkezi olarak izlenebilmesini sağlayan bileşenlerden oluşur. Aşağıda sistemde kullanılan başlıca gözlemlenebilirlik bileşenleri ve rolleri açıklanmıştır:

* **Prometheus**: Mikroservislerin expose ettiği metrikleri /actuator/prometheus endpoint’i üzerinden düzenli aralıklarla toplar.
* **Grafana**: Prometheus verilerini kullanarak servis başına gecikme, istek sayısı, hata oranı gibi metrikleri görselleştirir.
* **Spring Boot Actuator**: Her mikroservis, /actuator/health ve /actuator/prometheus gibi endpoint’lerle sağlık ve metrik verilerini sağlar.
* **Logback**: Her servis içinde yapılan işlemler, hatalar ve önemli olaylar Logback aracılığıyla loglanır. Log formatları JSON ya da pattern tabanlı yapıdadır ve sistem tanısı için kullanılır.
* **Resilience4j Circuit Breaker**: Mikroservisler arası çağrılarda dış bağımlılıklarda hata durumlarını yönetmek için kullanılır. Devre kesici (circuit breaker) açık olduğunda başarısız çağrılar önlenir ve fallback metotları devreye girer.
* **Rate Limiting (API Gateway)**: API Gateway, gelen istekleri belirli oranlarda kısıtlayarak sistemin aşırı yüklenmesini engeller. Bu sayede DDoS gibi saldırılara veya aşırı istek yüklerine karşı servisler korunmuş olur.
* **Docker Compose**: Tüm mikroservislerin ve gözlemlenebilirlik bileşenlerinin (Prometheus, Grafana vb.) tek komutla ayağa kaldırılmasını sağlar.

Bu yapı sayesinde:

* Servislerin anlık durumu gözlemlenir.
* Arızalı bağımlılıklar otomatik devre dışı bırakılır.
* Servislerin davranışları (loglar ve metrikler) detaylı şekilde takip edilir.
* Anlık veya geçmişe dönük sistem yükü analizleri yapılabilir.

# 5. Kurulum Adımları — *Prometheus & Grafana için*

1. Proje Kodunu Klonlayın:  
 git clone https://github.com/kullaniciadi/petverse-observability.git  
 cd petverse-observability

2.Servisler değişikliklerden kaynaklı yapılandırılır:  
 mvn clean install -DskipTests

3. Docker Compose ile Servisleri Kapatıldı:  
 docker compose down   
  
4. Docker Compose ile Servisleri Başlatın:  
 docker compose up --build  
  
5. Servisleri Kontrol Edin:  
 - http://localhost:8081/actuator/health (ActivityService)  
 - http://localhost:8082/actuator/health (NotificationService)  
  
6. Prometheus’a Erişim:  
 - http://localhost:9090 > Targets sekmesinden servis durumu kontrol edilir.  
  
7. Grafana’ya Giriş:  
 - http://localhost:3000  
 - Kullanıcı adı: admin | Şifre: admin123

# 5.1. Metrik Açıklamaları

|  |  |
| --- | --- |
| Metrik | Açıklama |
| http\_server\_requests\_seconds\_count | HTTP isteği sayısı |
| process\_cpu\_usage | CPU kullanım oranı |
| jvm\_memory\_used\_bytes | JVM bellek kullanımı |

# 5.2. Sık Karşılaşılan Hatalar ve Çözümleri

|  |  |  |
| --- | --- | --- |
| Hata | Neden | Çözüm |
| Prometheus Target DOWN | Servise ulaşılamıyor | /actuator/prometheus açık mı kontrol edin |
| Grafana Giriş Hatası | Yanlış şifre | admin/admin veya admin123 ile deneyin |

**Not:** Prometheus servislerine ulaşamıyorsanız docker ps komutu ile container'ların aktif olup olmadığını kontrol edebilirsiniz.

# 5.3. Dashboard Kullanımı

Grafana arayüzü üzerinden ‘Dashboards > Manage’ sekmesine giderek metrik panellerine erişebilirsiniz. Zaman aralığını değiştirebilir, filtreleme yapabilir ve her servisin CPU, bellek, istek sayısı gibi değerlerini grafik olarak görüntüleyebilirsiniz.

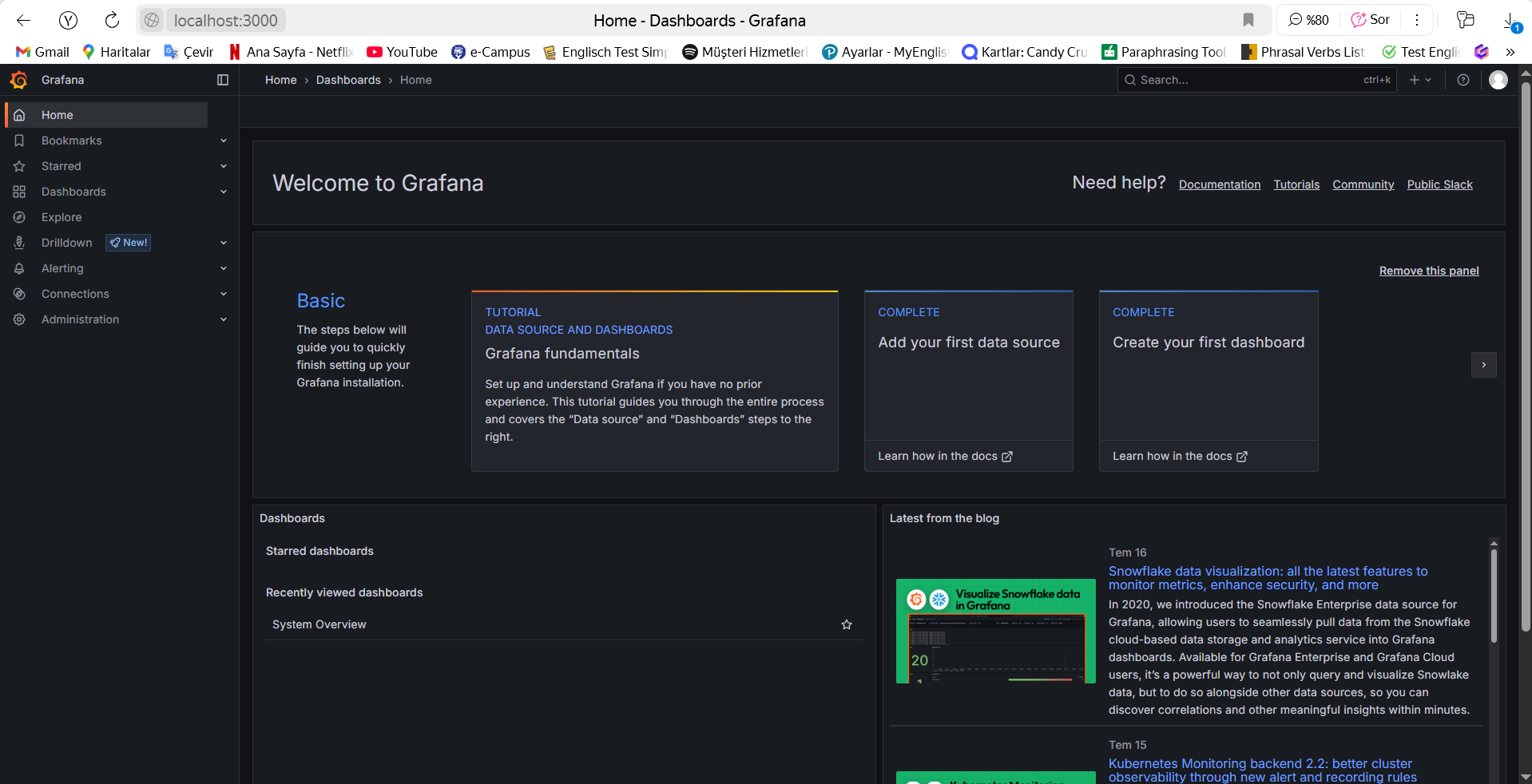
## 5.3.1. Grafana Dashboard Ekran Görüntüleri

### Grafana Giriş Ekranı



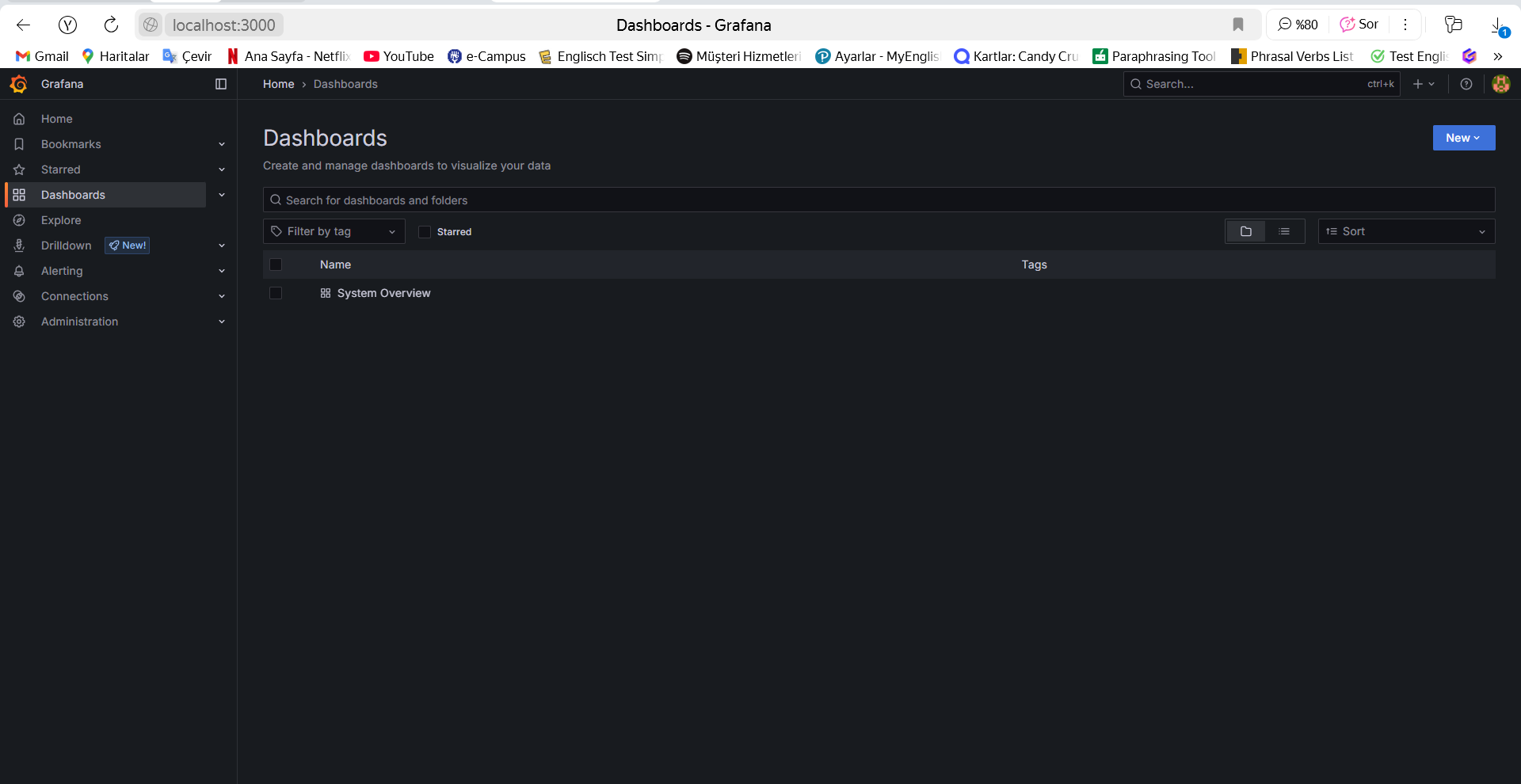
Kullanıcı adı ve şifre ile giriş yapılır. (Varsayılan: admin/admin123)

### Grafana Ana Sayfa



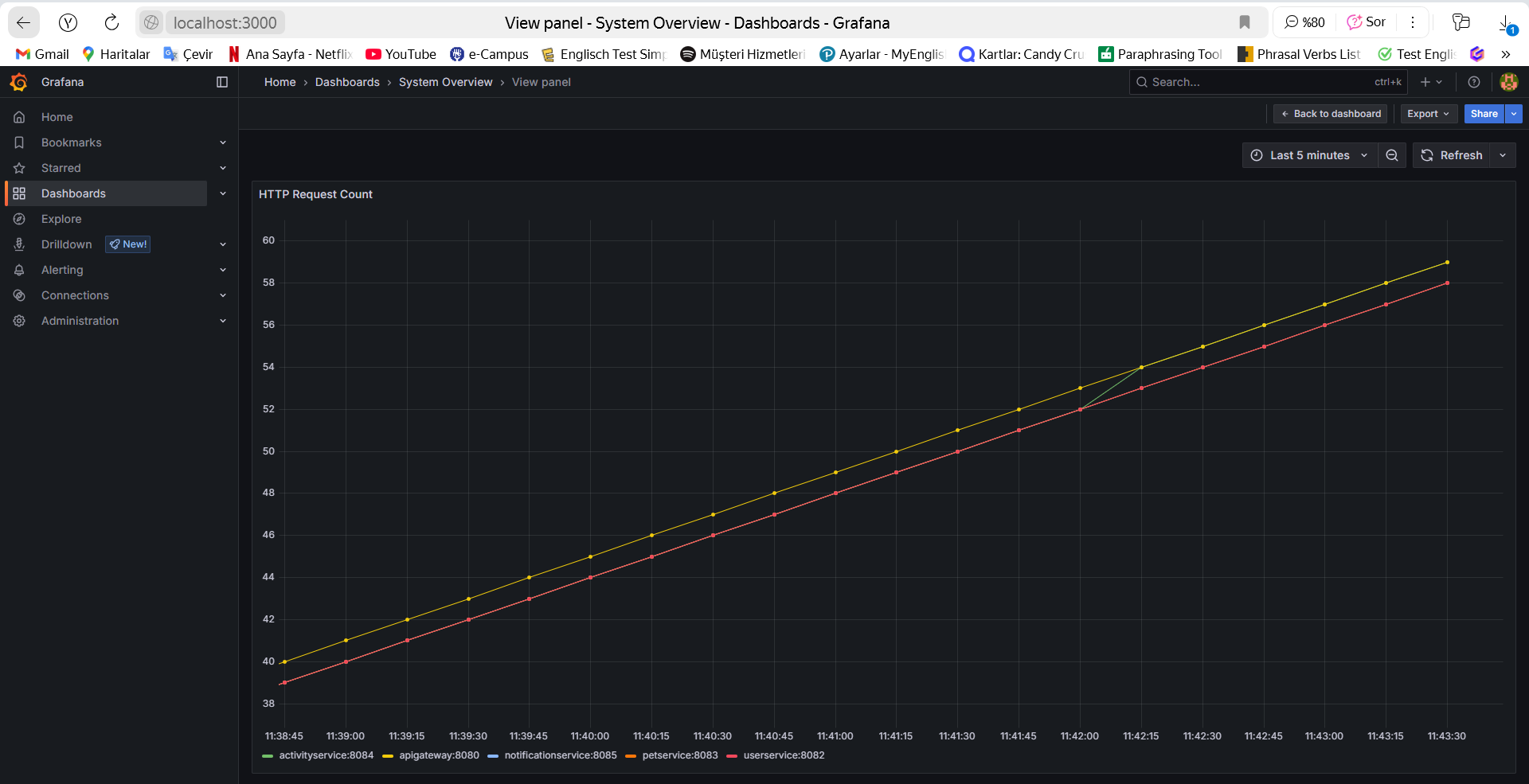
Grafana arayüzüne giriş sonrası temel ekran görünümü.

### Dashboard Listesi



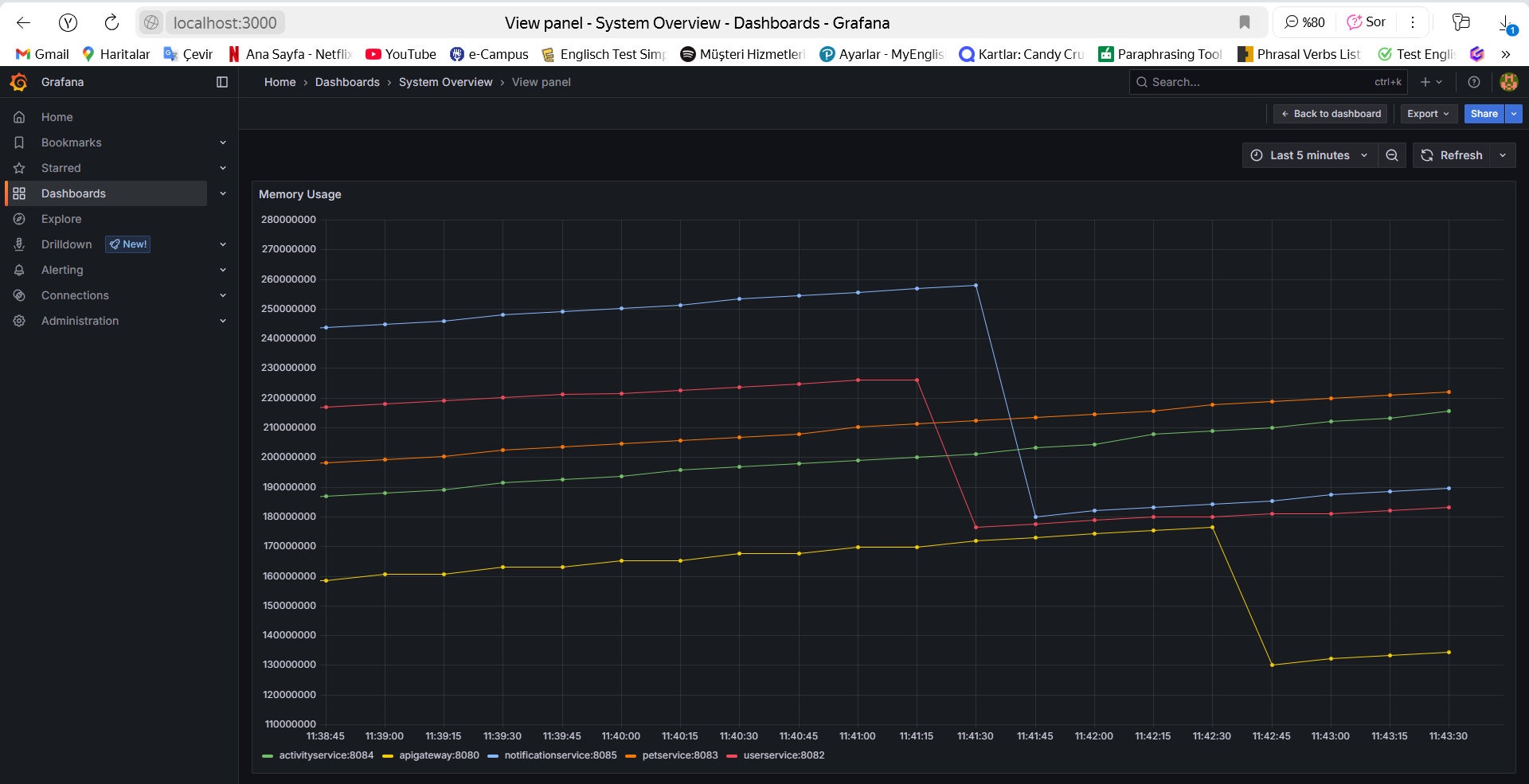
‘System Overview’ isimli dashboard burada listelenir.

### HTTP Request Count Paneli



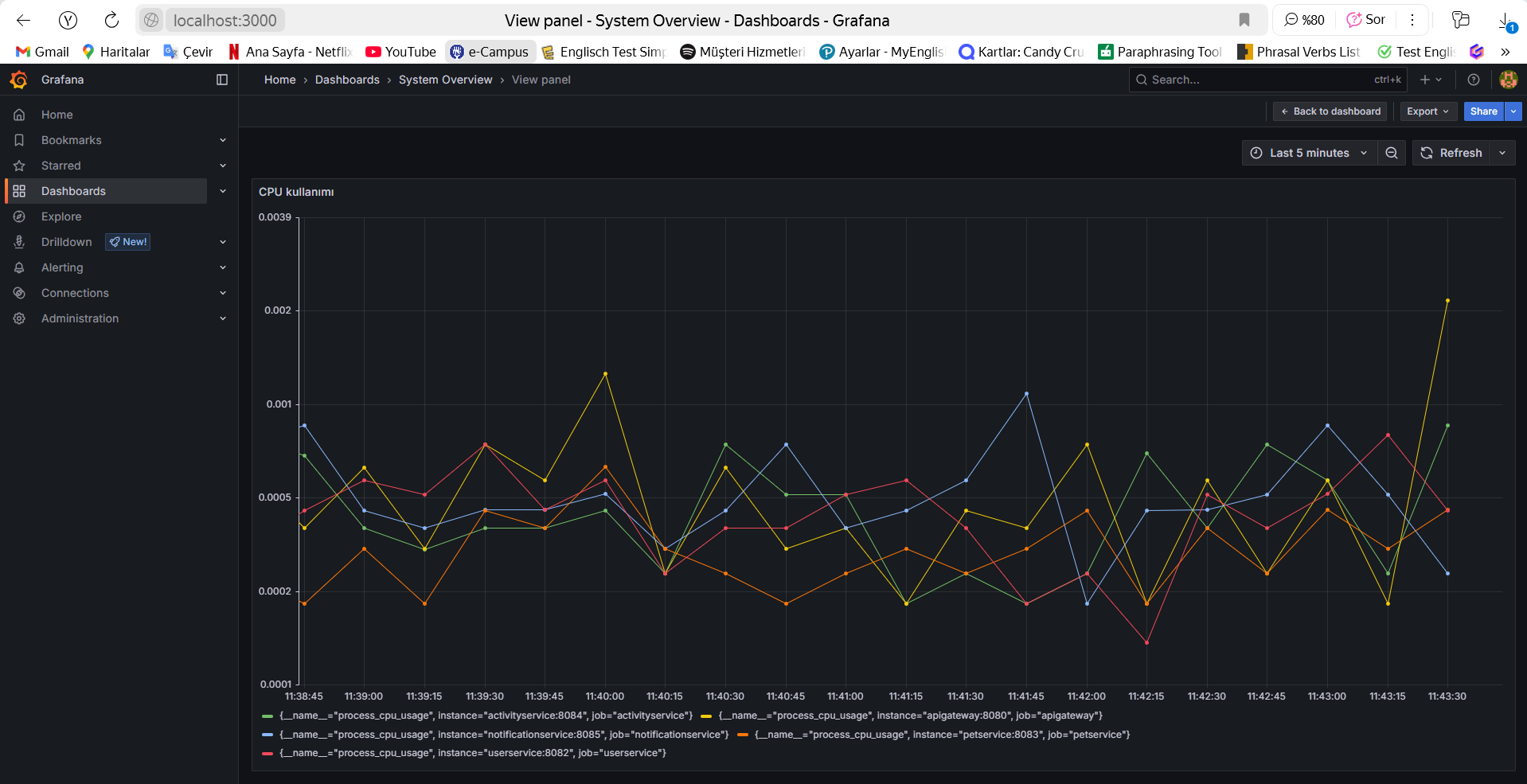
Her bir mikroservisin HTTP istek artışı zaman içinde takip edilir.

### Memory Usage Paneli



Mikroservislerin JVM bellek kullanımı zaman serisi olarak gösterilir.

### CPU Kullanımı Paneli



Tüm servislerin CPU kullanım eğrileri karşılaştırmalı olarak izlenebilir.

## 5.3.2. Yeni Dashboard Oluşturma: ActivityService Örneği

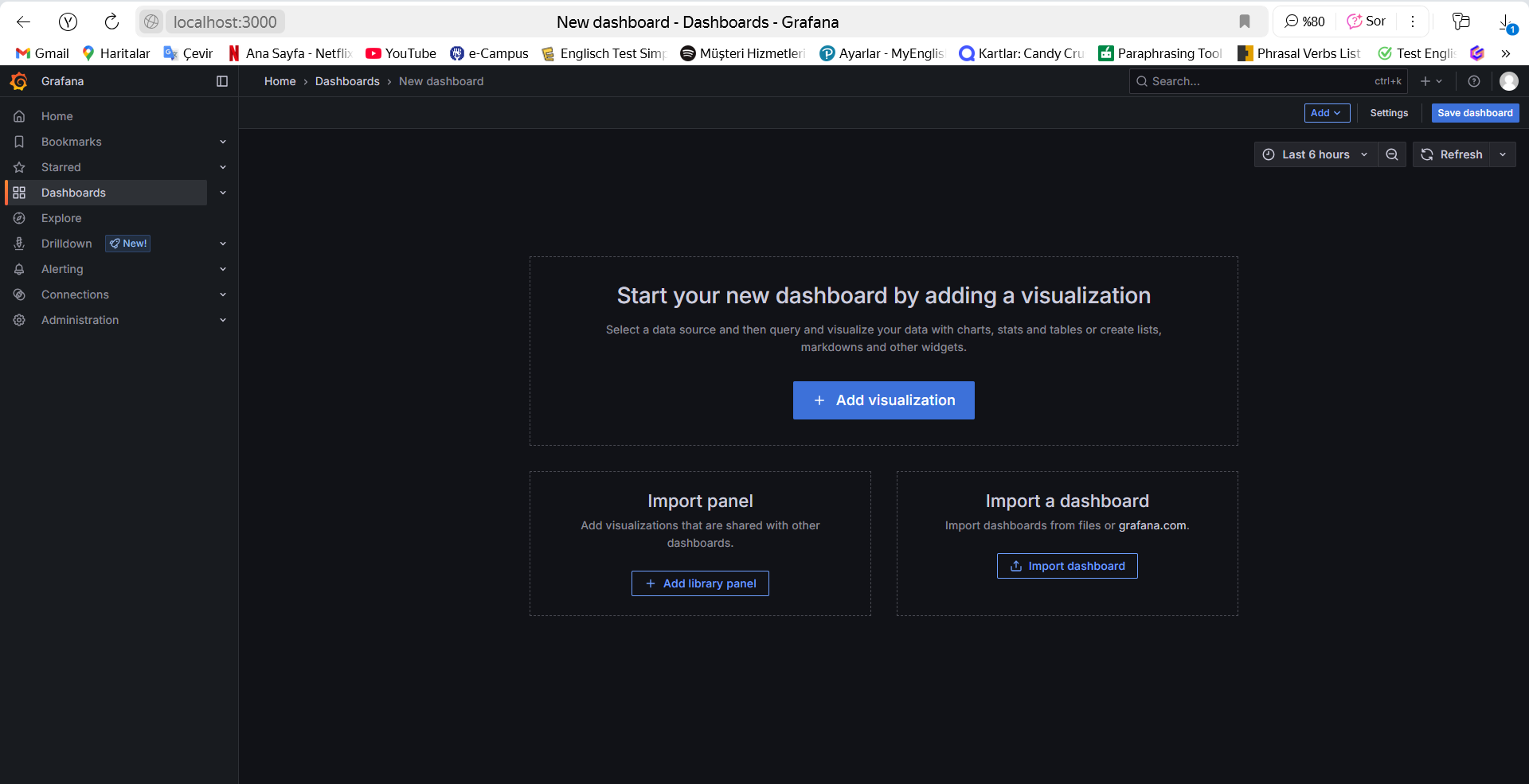
Bu bölümde Grafana arayüzü kullanılarak ActivityService için özel bir dashboard oluşturma süreci adım adım gösterilmektedir. Her adım ilgili ekran görüntüsü ile desteklenmiştir.

### Yeni Dashboard Oluşturma



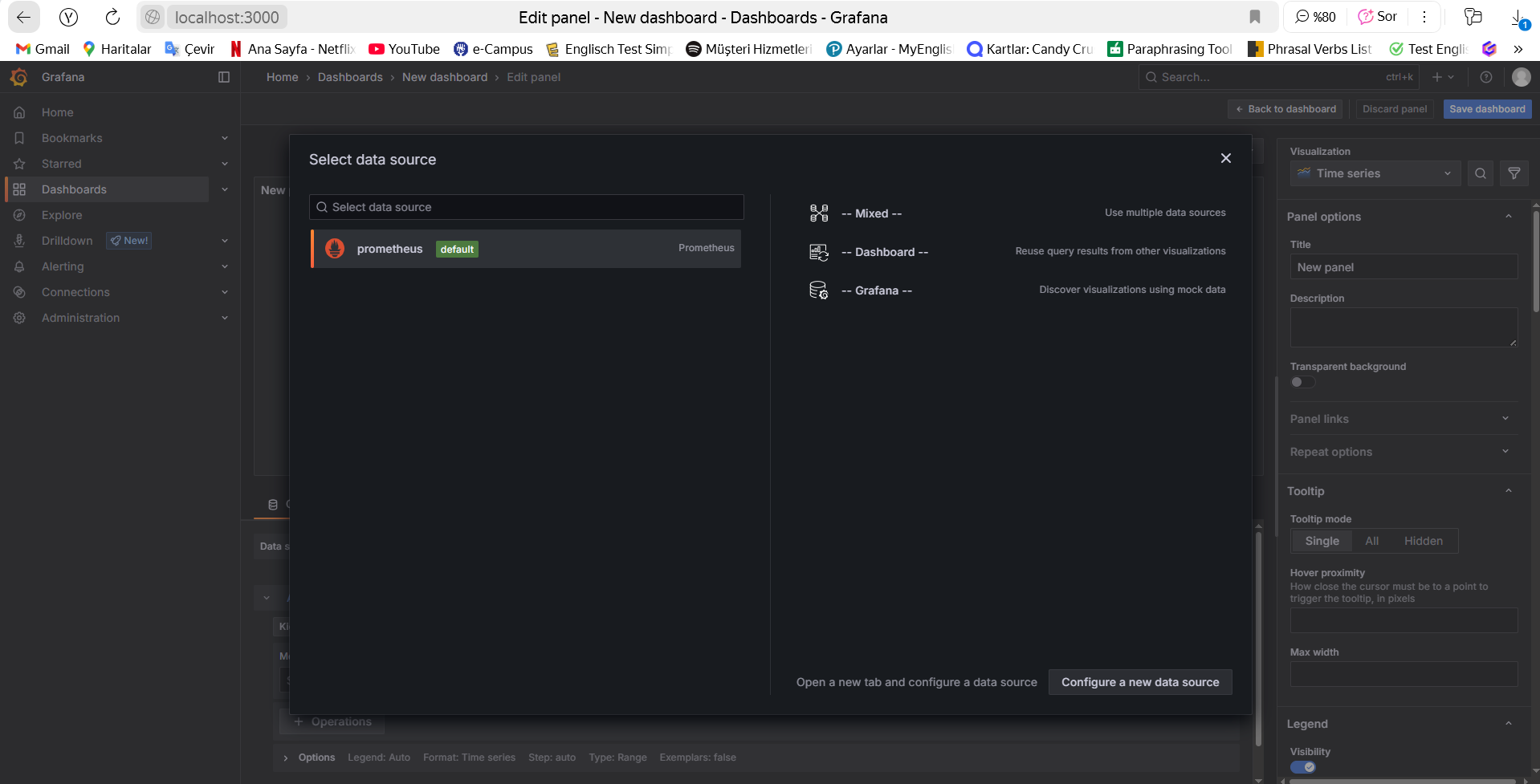
Grafana ana ekranında sağ üstten 'New' > 'New dashboard' seçeneği ile yeni bir dashboard başlatılır.

### Görselleştirme Ekle



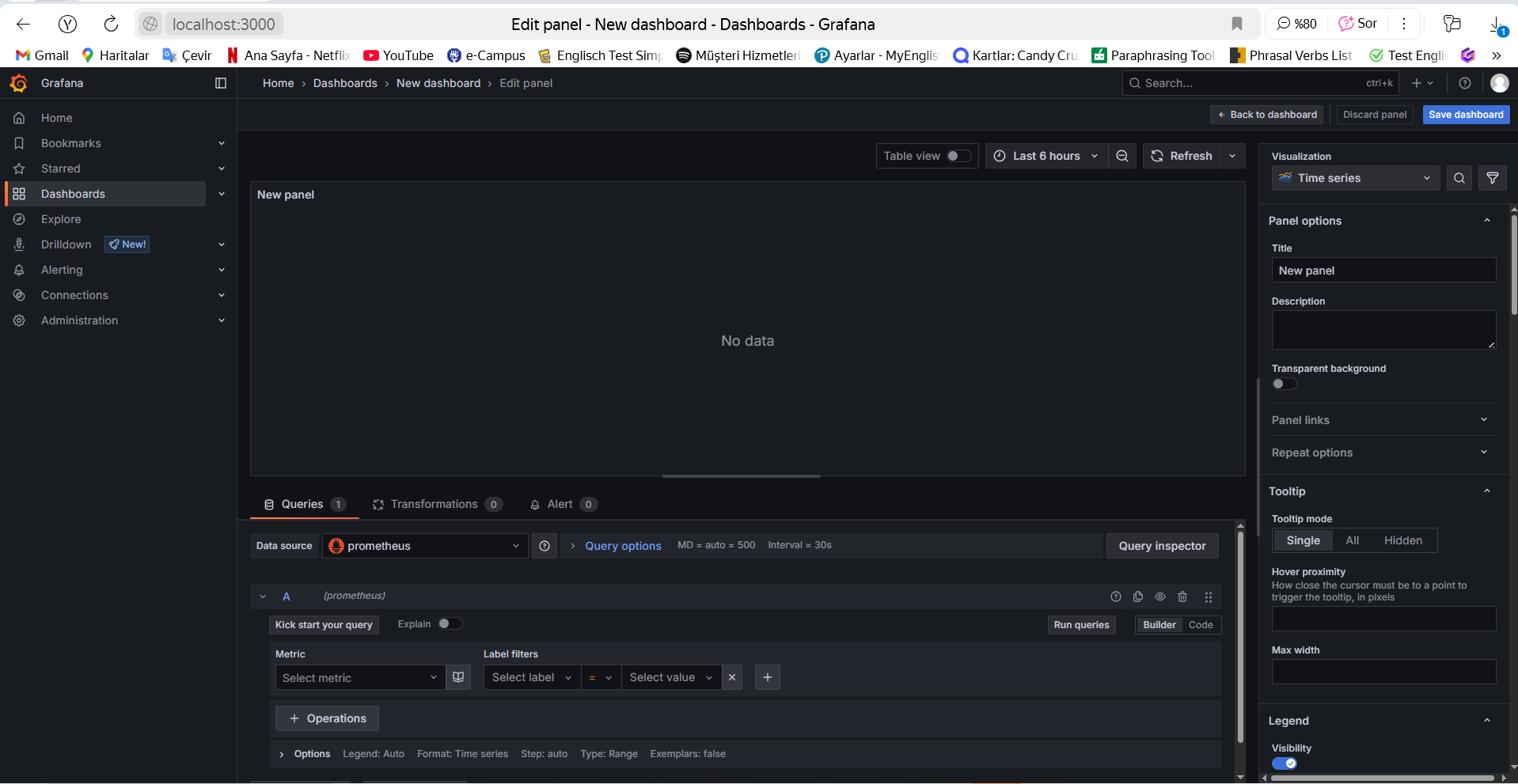
'Add visualization' butonuna tıklanarak yeni panel oluşturma ekranına geçilir.

### Veri Kaynağı Seçimi



Prometheus veri kaynağı olarak seçilir.

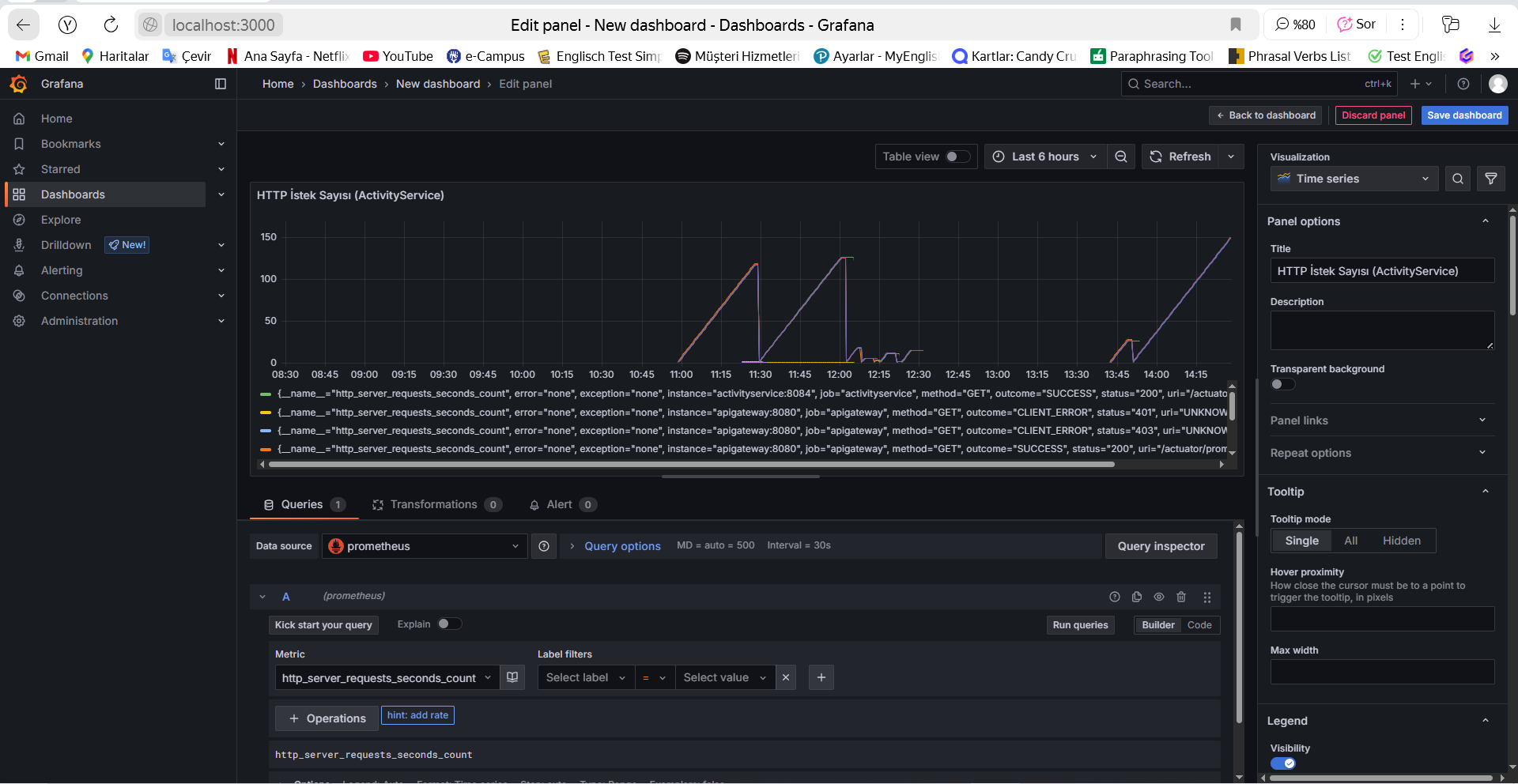
### Metrik Sorgusu Girme



Prometheus sorgu alanında `http\_server\_requests\_seconds\_count{job="activityservice"}` gibi bir sorgu yazılır ve grafik yapılandırılır.

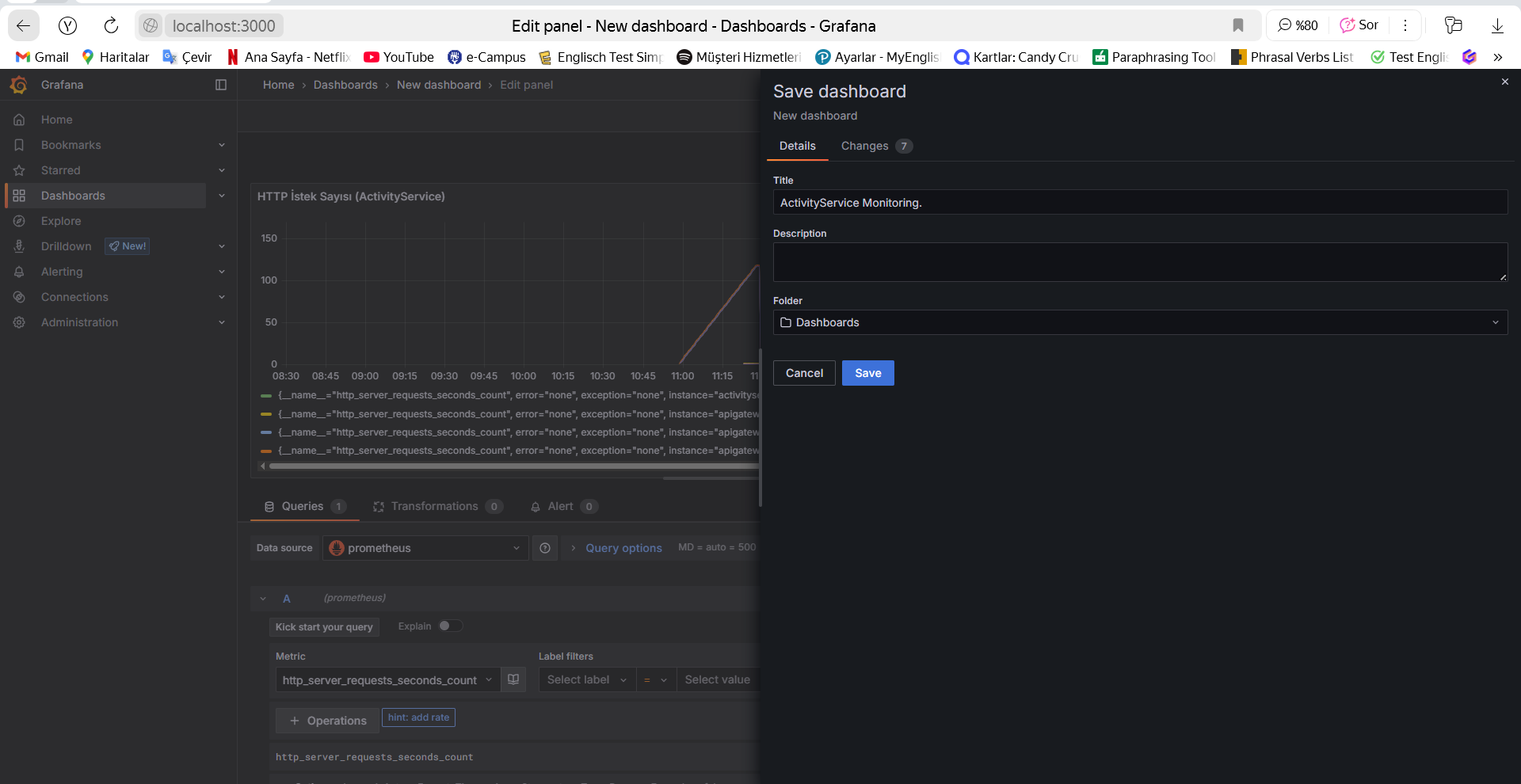
Aşağıdaki adımlar, oluşturulan panelin nasıl yapılandırıldığını ve kaydedildiğini gösterir:

### Panel Başlığı ve Grafik



Grafikte HTTP isteklerinin zaman serisi olarak gösterimi sağlanır. Panel başlığı: 'HTTP İstek Sayısı (ActivityService)'

### Dashboard Kaydetme

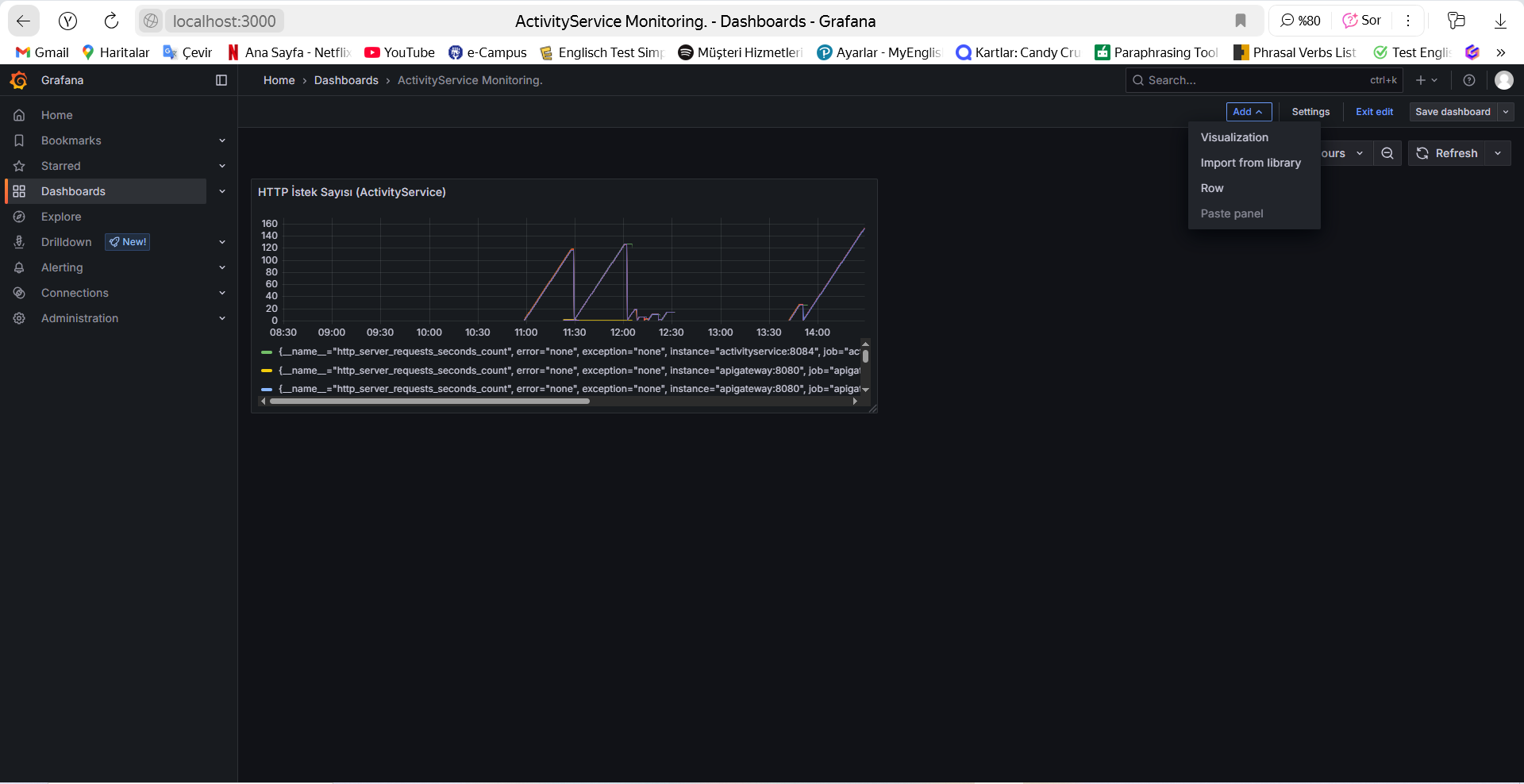


Dashboard 'ActivityService Monitoring' adıyla kaydedilir ve panel içine yerleştirilir.

## 5.3.3. Mevcut Dashboard'a Yeni Panel Ekleme (Bellek Kullanımı)

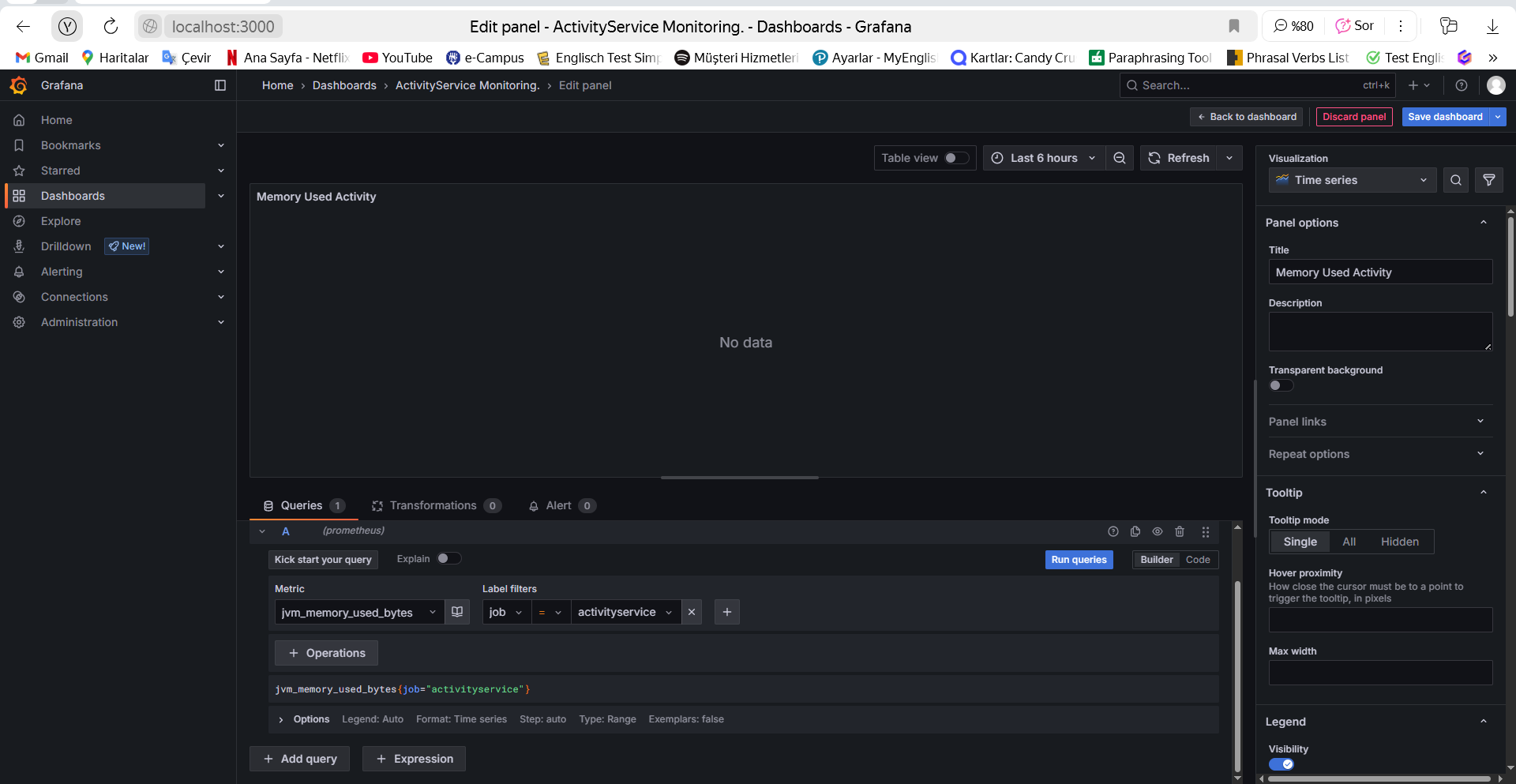
Bu bölümde, oluşturulmuş bir dashboard içerisine ActivityService'e ait yeni bir metrik paneli (bellek kullanımı) nasıl ekleneceği adım adım açıklanmaktadır. Bu işlem sayesinde aynı dashboard üzerinden farklı metrikleri birlikte takip etmek mümkündür.

### Yeni Panel Ekleme



Mevcut dashboard içerisinden 'Add' > 'Visualization' seçilerek yeni bir panel ekleme işlemi başlatılır.

### Bellek Kullanım Metrik Sorgusu

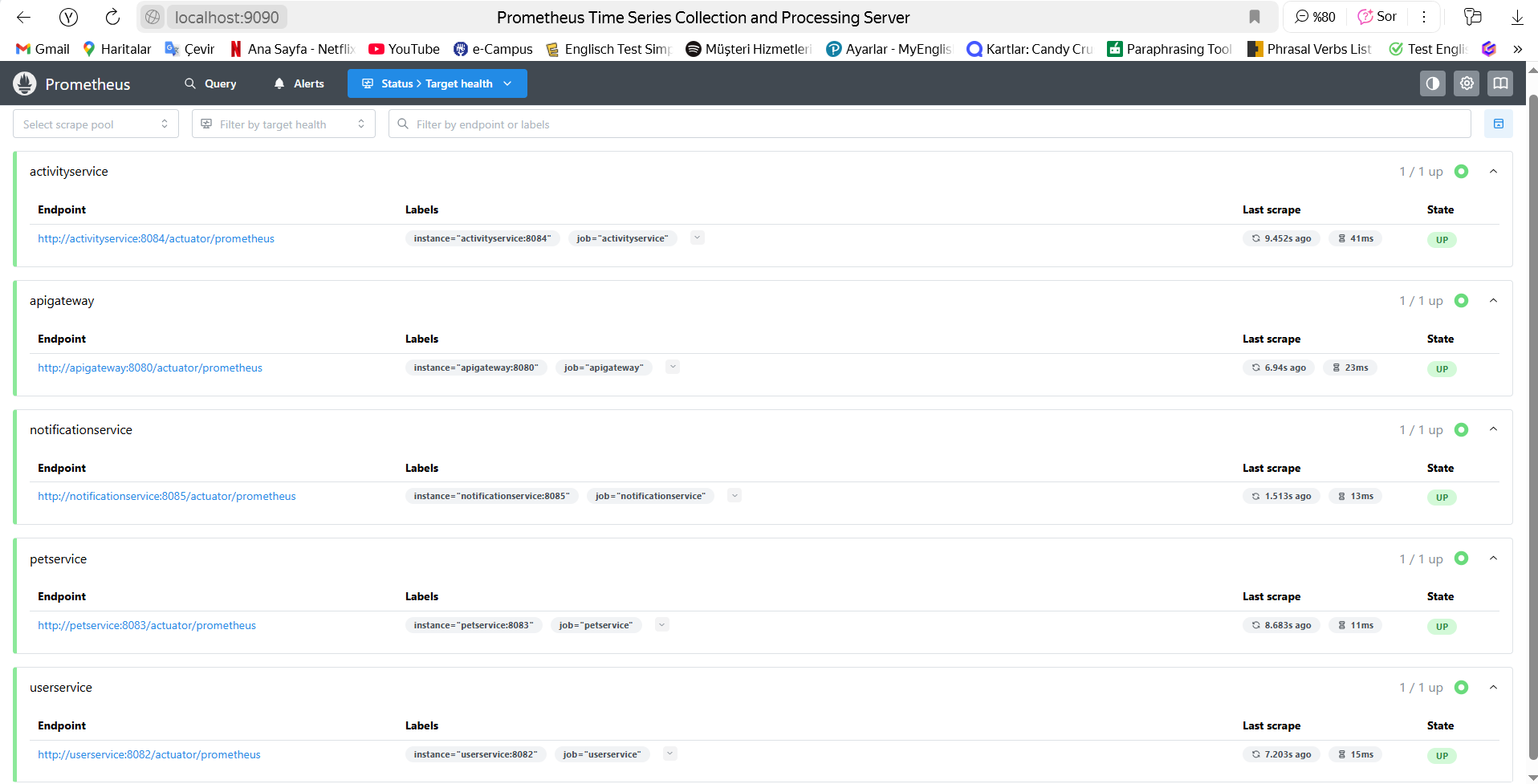


Sorgu alanına `jvm\_memory\_used\_bytes{job="activityservice"}` yazılarak bellek kullanım metrikleri çekilir. Panel başlığı olarak 'Memory Used Activity' belirlenmiştir. Görselleştirme tipi olarak zaman serisi (time series) kullanılmıştır. Kaydetme işlemleri yukarda anlatıldığı şekilde tamamlanmaktadır.

# 5.4. Prometheus Kullanımı ve Durum İzleme

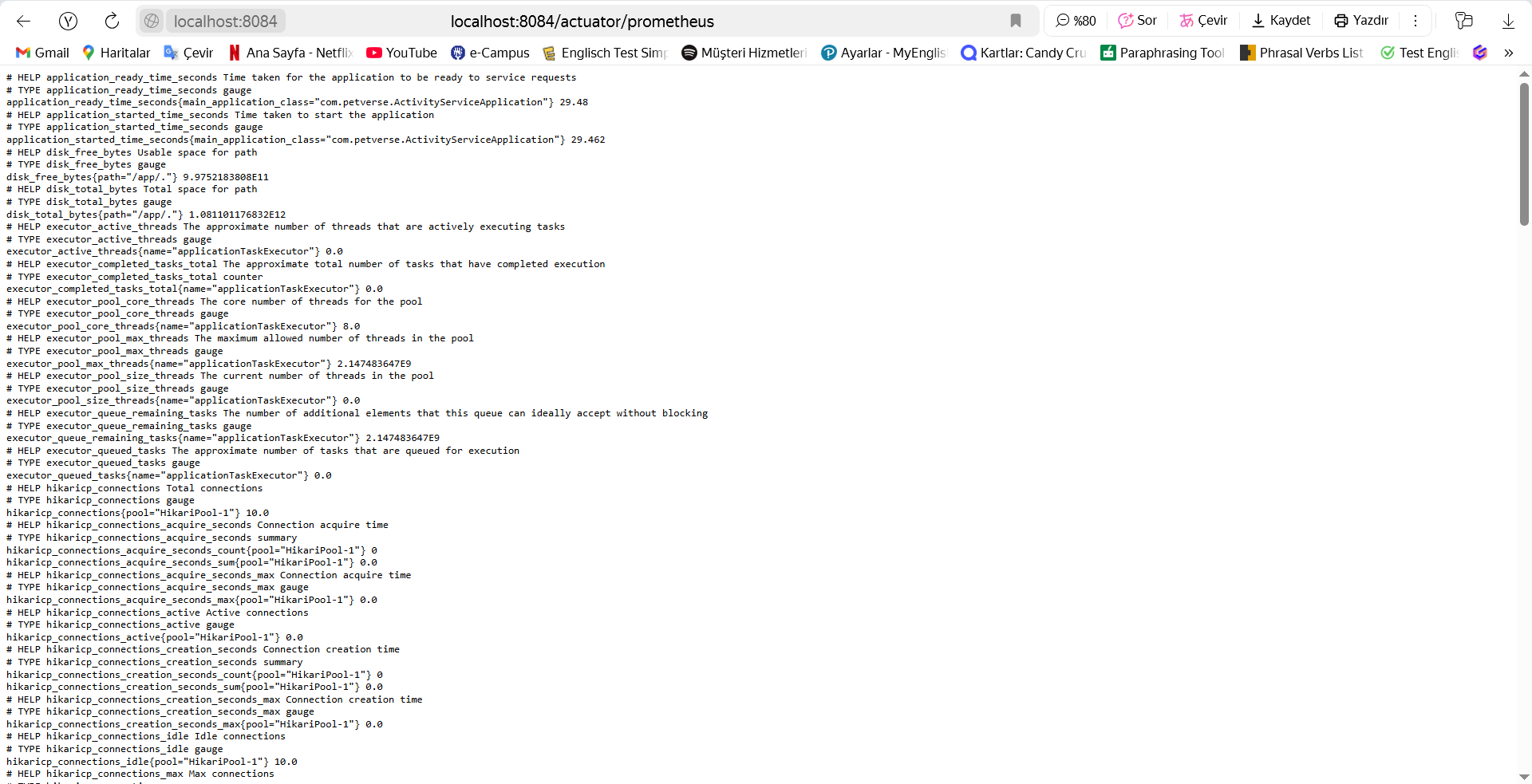
Prometheus, mikroservislerden metrik verilerini toplayan zaman serisi veri tabanı sistemidir. Aşağıdaki adımlar, Prometheus'un nasıl kullanıldığını ve servislerin izlenme durumunun nasıl kontrol edildiğini açıklamaktadır.

### Prometheus Target Kontrolü



Prometheus'un http://localhost:9090 arayüzünde 'Status > Targets' sekmesine gidilerek hangi servislerin takip edildiği ve veri alınıp alınamadığı kontrol edilir. Durumu 'UP' olan servislerden veri başarıyla alınmaktadır. (http://localhost:9090/targets)

### Prometheus Endpoint Çıktısı

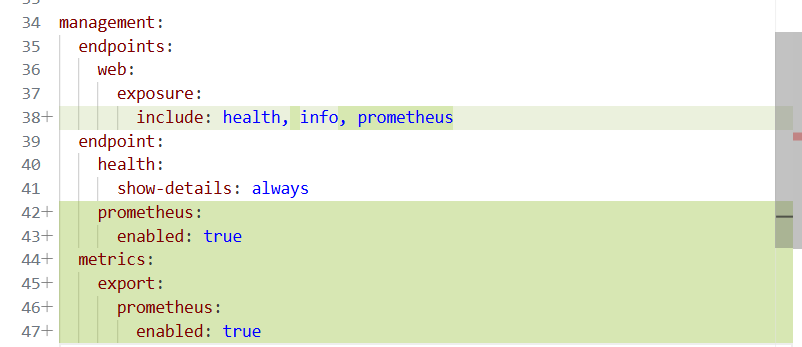


Servislerin '/actuator/prometheus' endpoint’ine gidildiğinde metrikler düz metin formatında listelenir. Bu metrikler Prometheus tarafından toplanır ve Grafana’ya aktarılabilir hale gelir.

## 5.4.1 Kurulumda Yapılan Kod Değişiklikleri

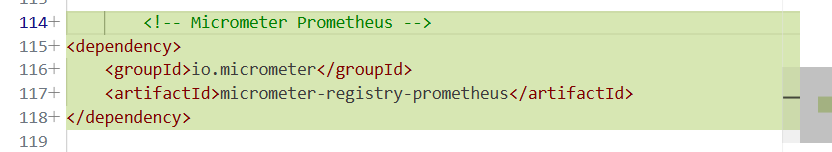
Gözlemlenebilirlik sistemi için her mikroservise özel bazı yapılandırmalar ve bağımlılıklar eklenmiştir. Bu değişiklikler aşağıdaki başlıklar altında gruplanmıştır:

### application.yml Konfigürasyonu



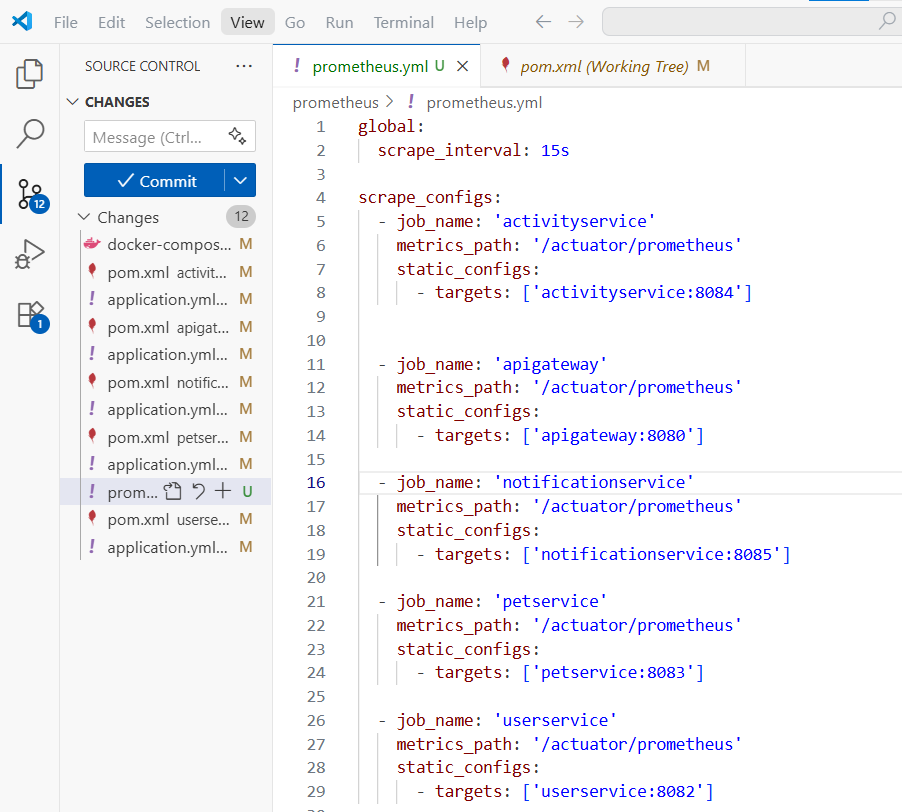
Her servisin `application.yml` dosyasına 'management.endpoints.web.exposure.include' ile `health`, `info`, ve `prometheus` endpoint'leri dahil edilmiştir. Ayrıca Prometheus’un etkinleştirilmesi için `management.metrics.export.prometheus.enabled` parametresi `true` olarak ayarlanmıştır.

### pom.xml Bağımlılığı



Micrometer Prometheus desteği için `micrometer-registry-prometheus` bağımlılığı eklenmiştir.

### prometheus.yml Yapılandırması



`prometheus.yml` dosyasında her servis için `job\_name` ve `targets` tanımlanmıştır. Bu yapılandırma, Prometheus’un bu endpoint’lerden düzenli olarak veri çekmesini sağlar.

### docker-compose.yml Dosyasına Eklemeler



`docker-compose.yml` dosyasına `prometheus` ve `grafana` container’ları eklenmiştir. Prometheus için yapılandırma dosyası mount edilmiş, ayrıca veri kalıcılığı için volume tanımlanmıştır.

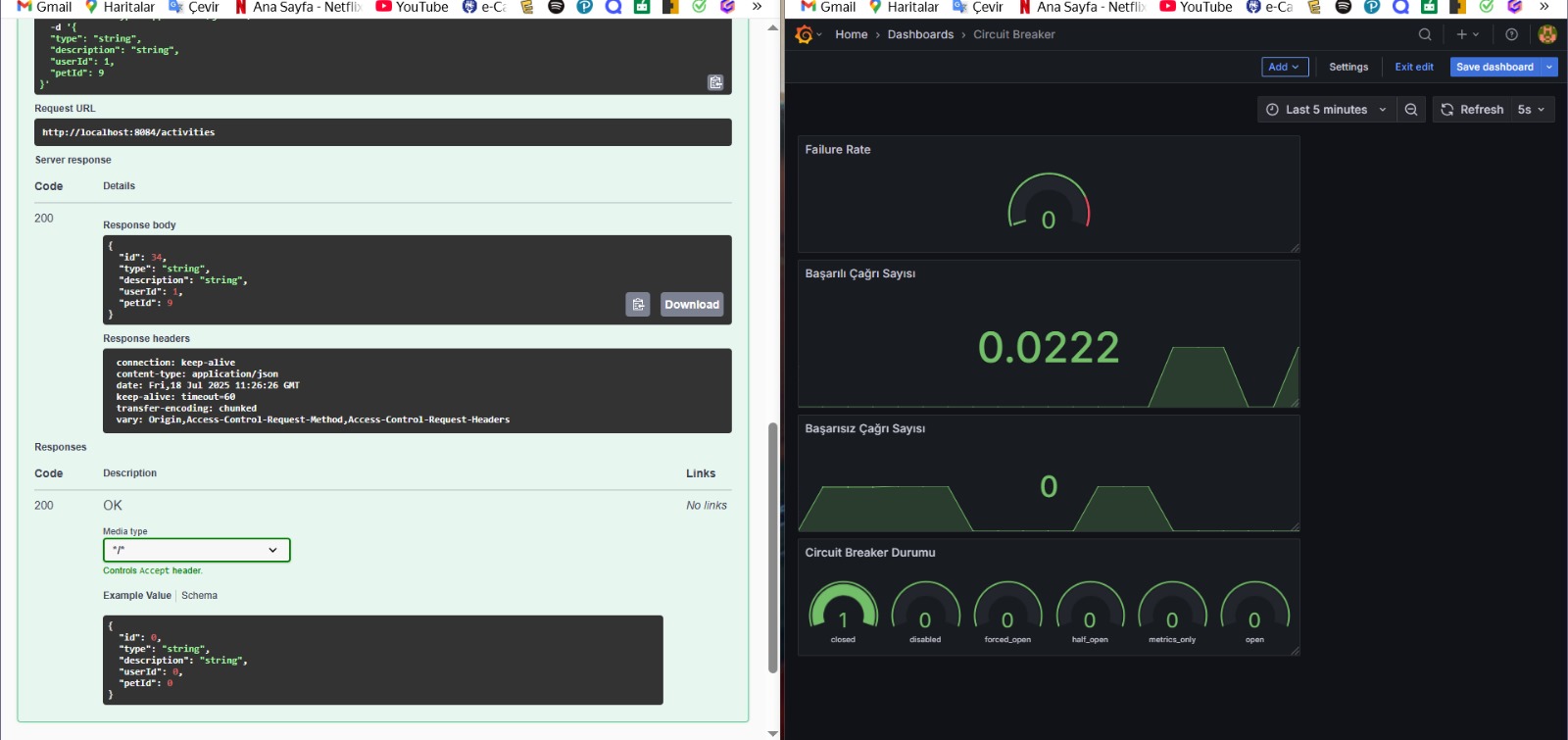
# 6. Circuit Breaker (Resilience4j) Yapılandırması

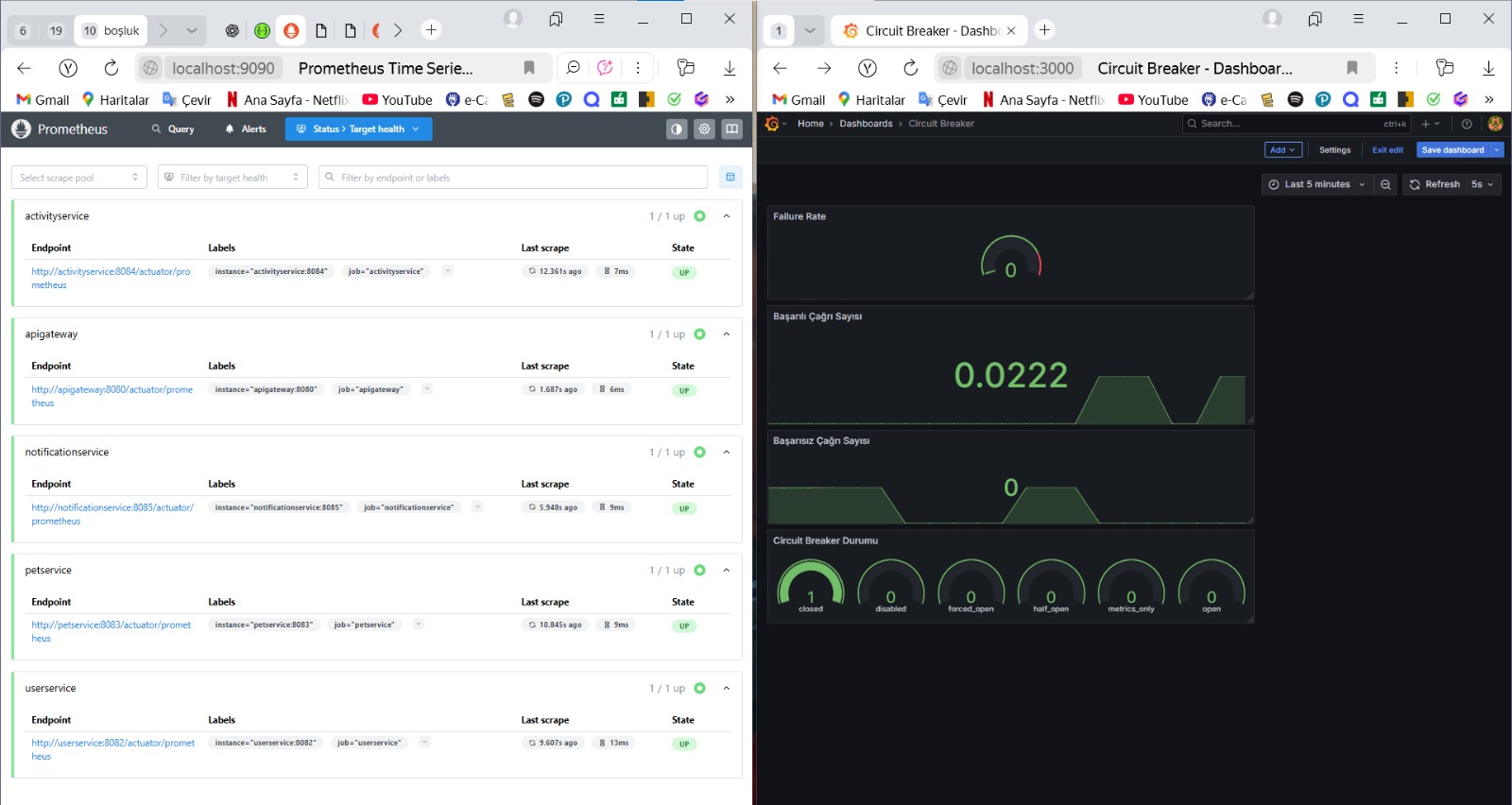
**Amaç:** PetService gibi dış bağımlılıklarda yaşanabilecek sorunlarda sistemin dayanıklılığını artırmak.  
**Yapılanlar:**

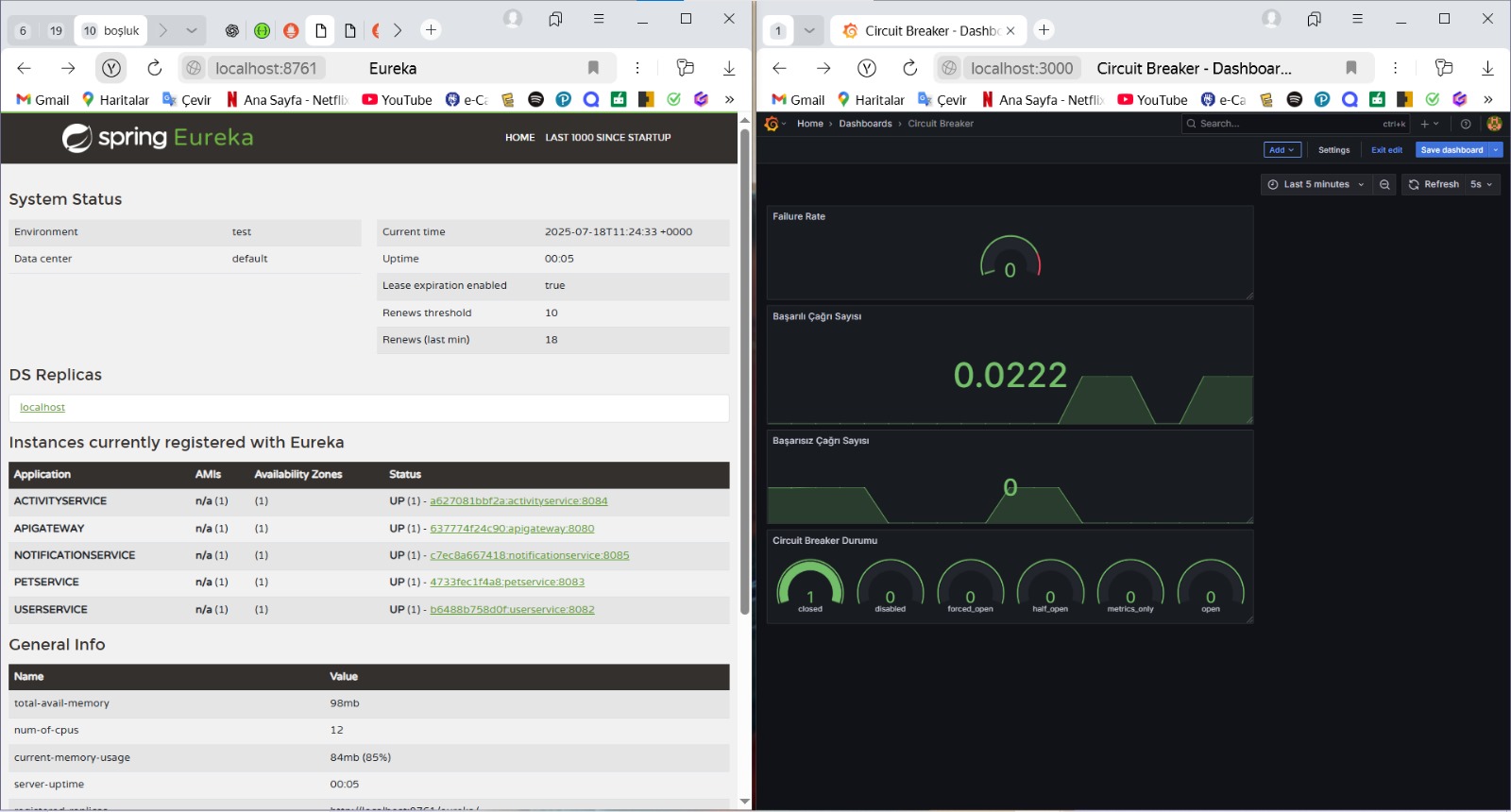
* Resilience4j kullanılarak ActivityService → PetService çağrısı koruma altına alındı.
* @CircuitBreaker, @Retry, @FallbackMethod anotasyonları kullanıldı.
* Başarısız çağrılar Prometheus metriklerine yansıdı.

## 6.1 Circut Breaker Dashboard Görüntülüleri

### 6.1.1 Başarılı Çağrı Senaryosu (Circuit Closed)





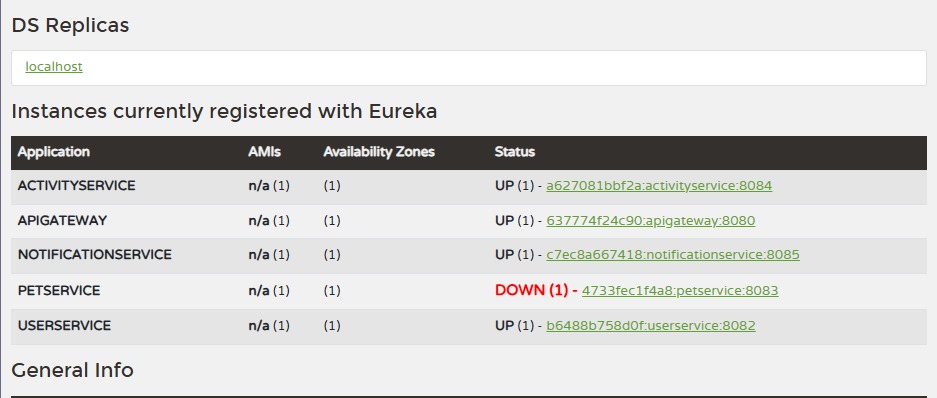
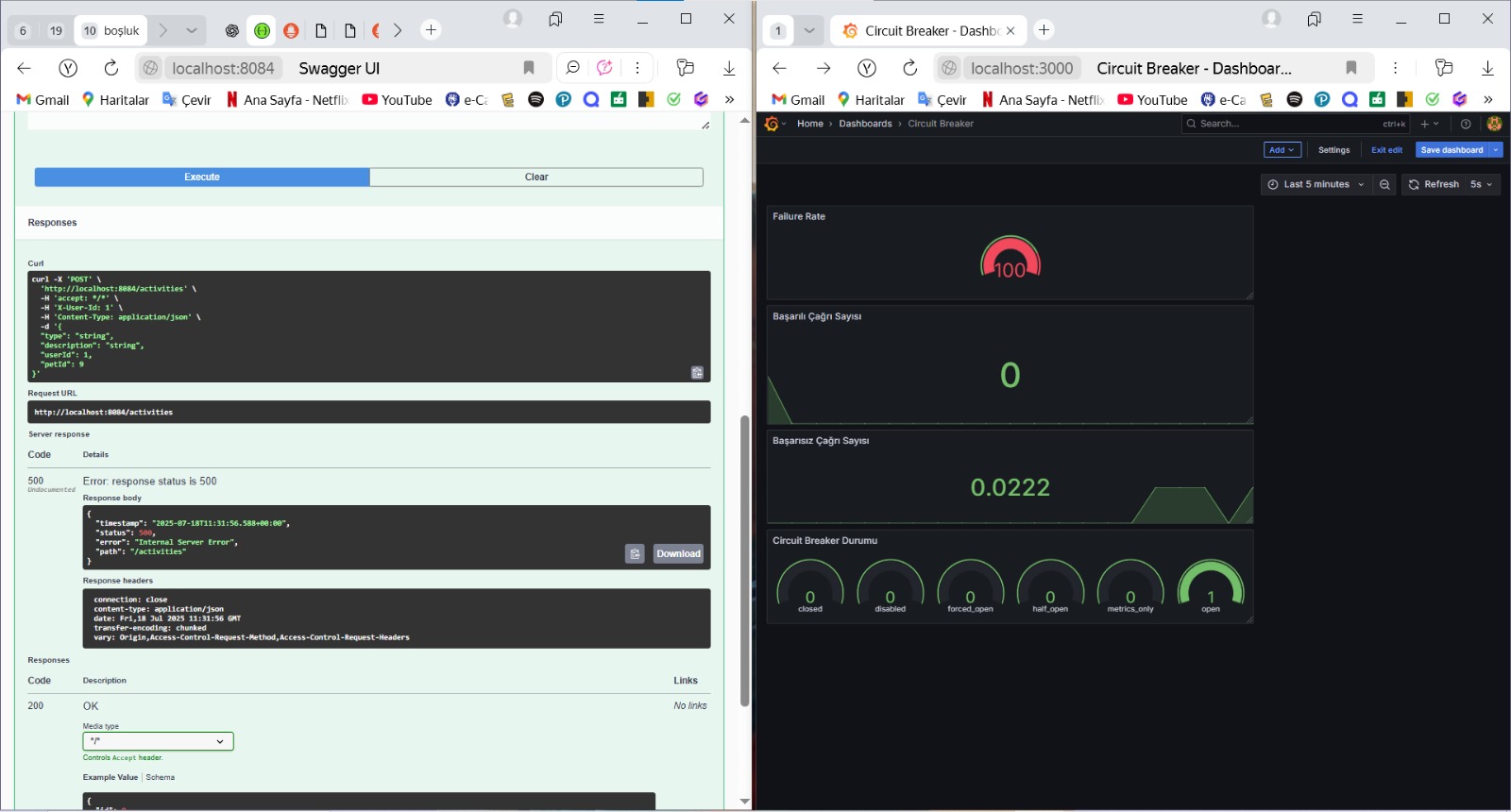


Sistem şu an açık ve çalışıyor prometheusta ve eurekada tüm servislerin ayakta olduğu göüzküyor. Swagger ve ya postmanden istek yolladığımız zaman servisler açıkken sistem başarılı çağrıyı alıyor ve hala close durumunda kalmaya devam ediyor yani circut breaker açılmıyor.

### 6.1.2 Servis Hatası ve Circuit Open Durumu

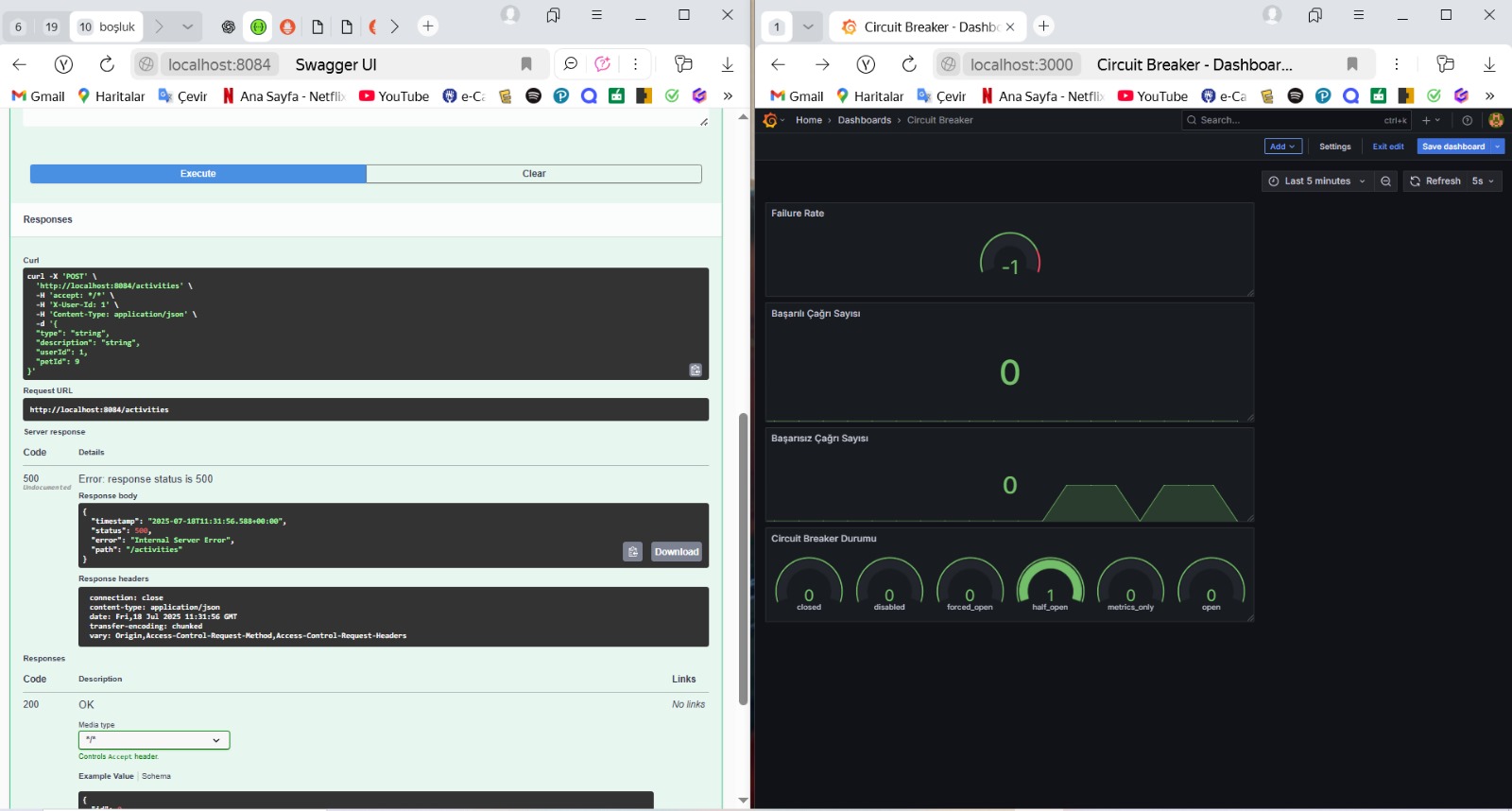
A screenshot of a computer

AI-generated content may be incorrect.

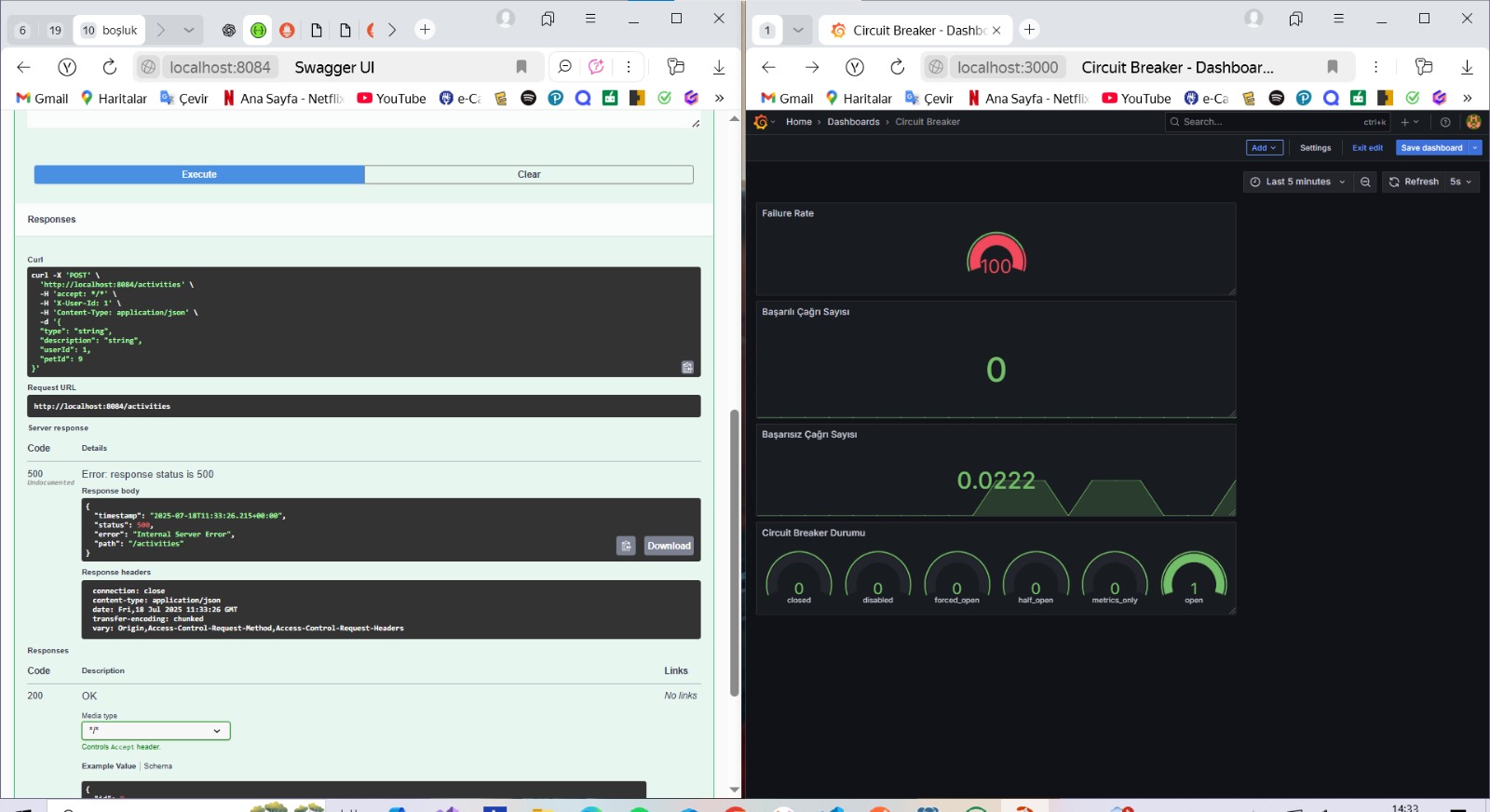


Petservice kapattığımız zaman prometheusta kapandığını gözlemliyoruz. Daha sonra bir istek yolladığımız zaman sistem başarısız çağırıyı alıyor ve circuit breaker aktif duruma geçiyor open oluyor.

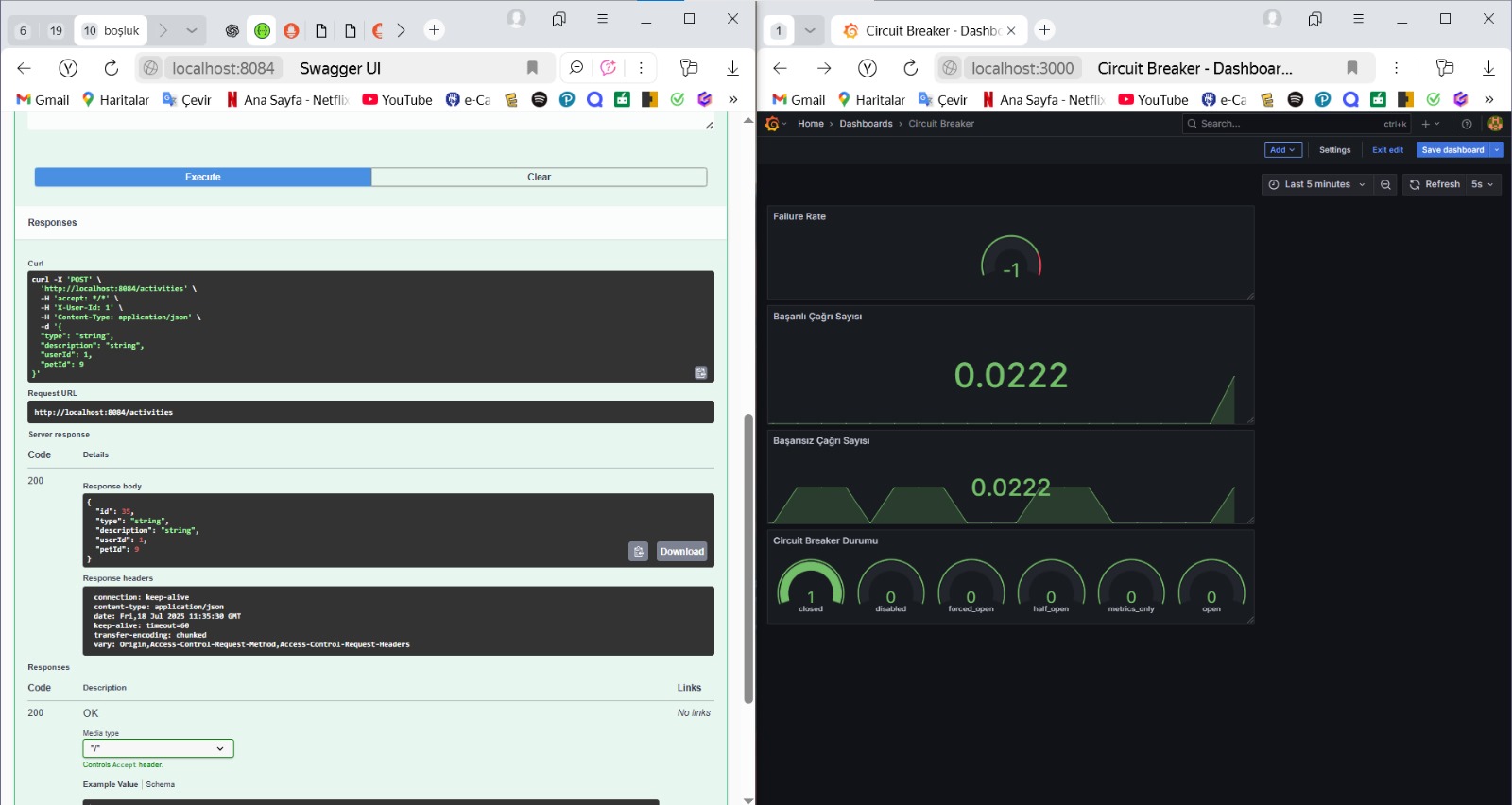
### 6.1.3 Recovery: Half-Open Durumu ve Yeniden Açılma



Başarısız işlemden 5 saniye sonra sistem half open durumuna geçer ve sistemi dinler.



Tekrar başarısız bir işlem gelirse opena düşer.

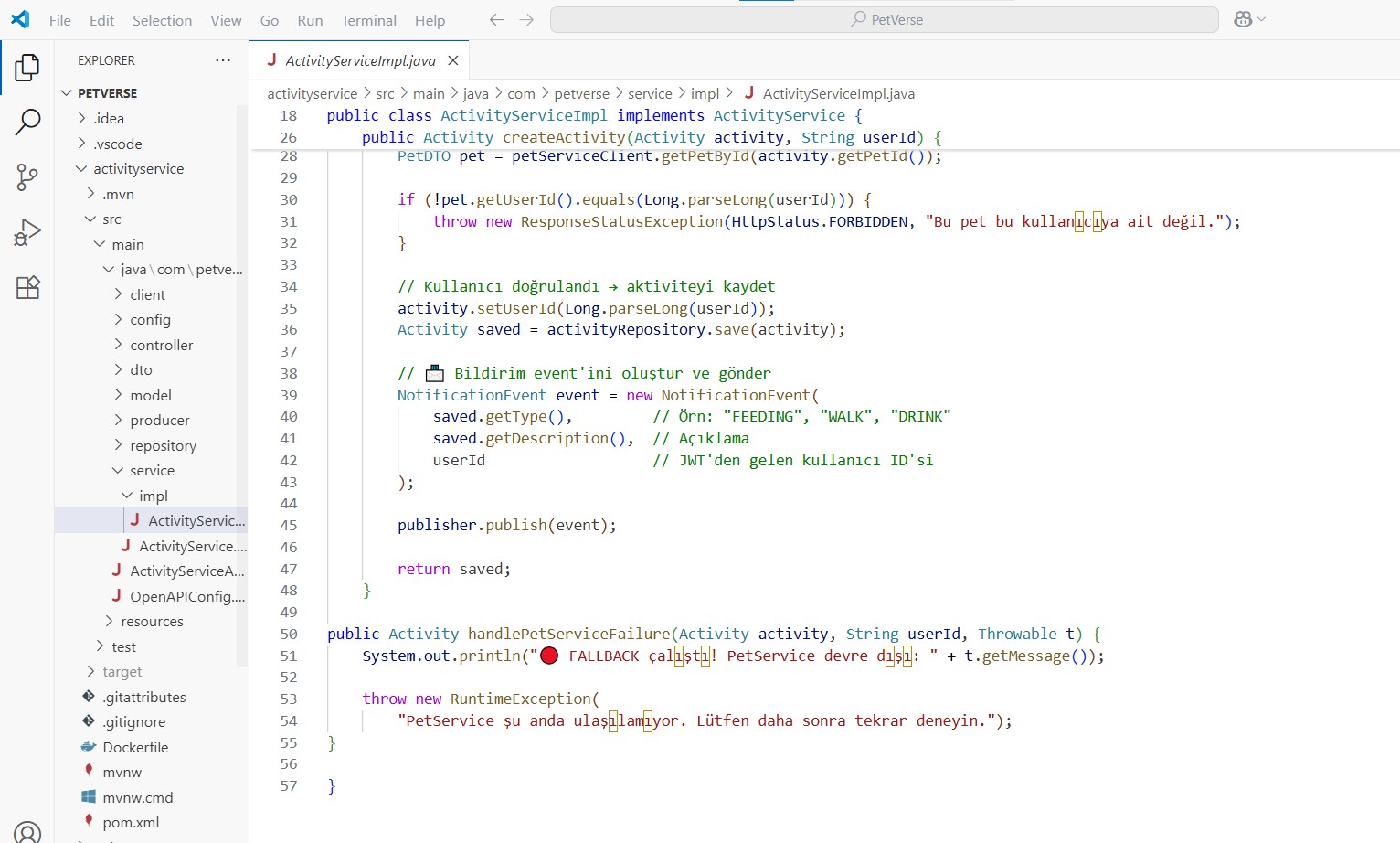
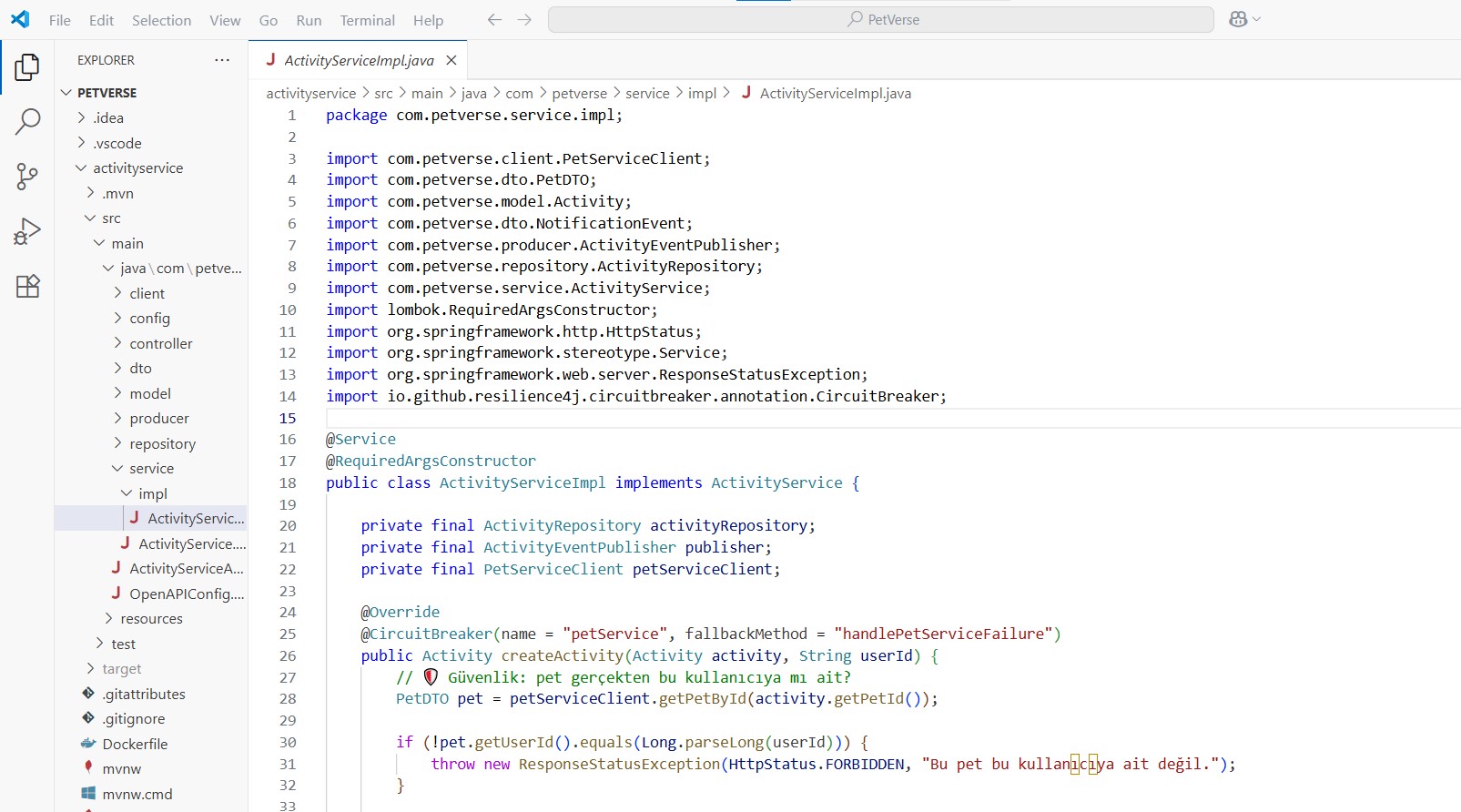


Tekrar bağlantı kurup istek yollayınca sistem close durumuna geçer.

## 6.2 Kurulumda Yapılan Kod Değişiklikleri

Gözlemlenebilirlik sistemi için her mikroservise özel bazı yapılandırmalar ve bağımlılıklar eklenmiştir. Bu değişiklikler aşağıdaki başlıklar altında gruplanmıştır:

### ActivityServiceImpl.java – Fallback Mekanizmasının Eklenmesi



* Eğer PetService çağrısı başarısız olursa, handlePetServiceFailure() metodu çalışarak fallback davranışı tanımlar.
* Bu metot kullanıcıya sistemin devre dışı olduğunu bildiren bir hata mesajı üretir ve hata konsola loglanır.

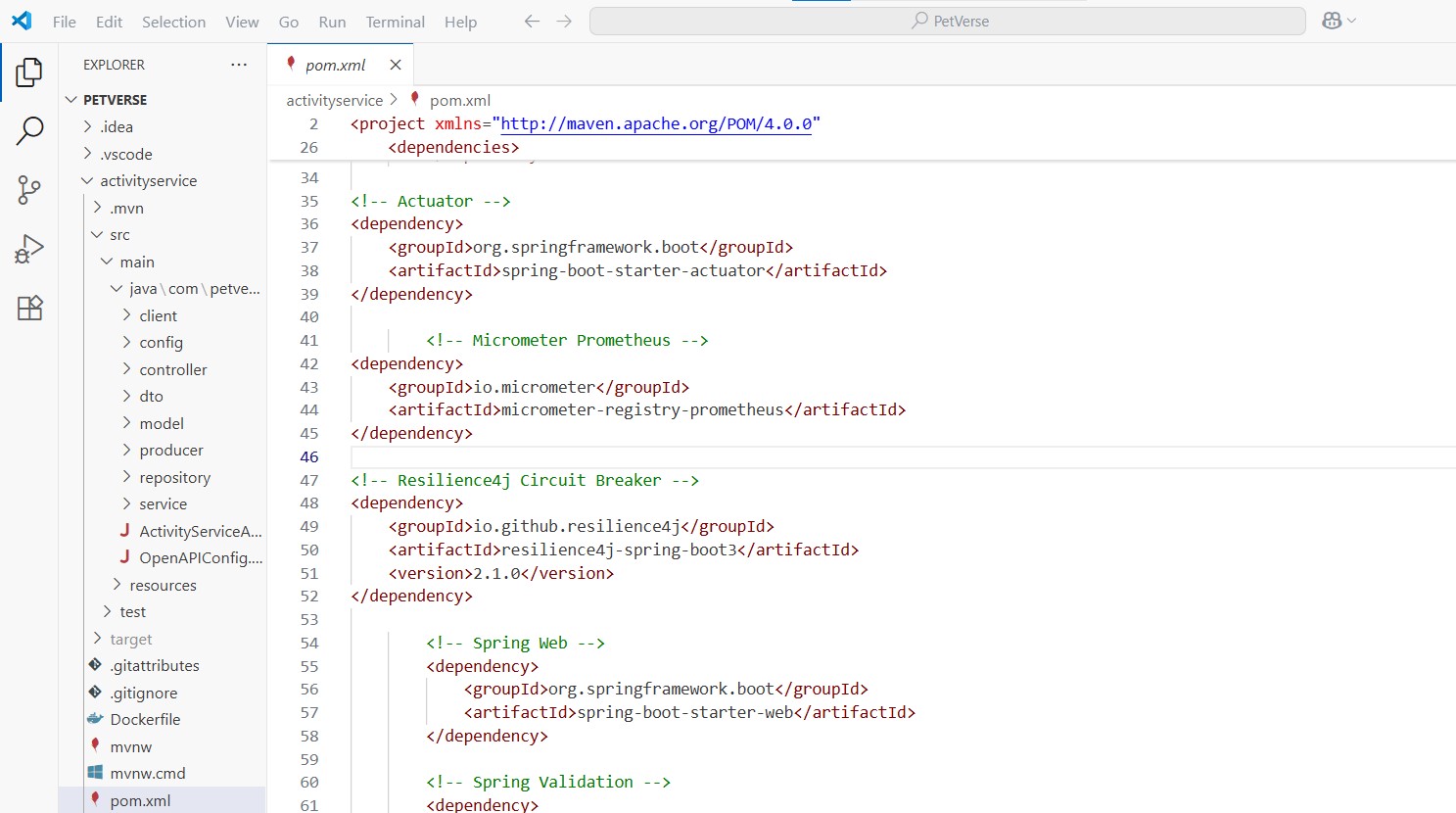
### application.yml eklemeler

A screenshot of a computer

AI-generated content may be incorrect.

* /actuator/health ve /actuator/prometheus endpoint'lerinin erişilebilir olması sağlanır.
* Prometheus için metrik export’u etkinleştirilir.
* resilience4j.circuitbreaker altında petService için devre kesici parametreleri yapılandırılır:
* management.endpoints.web.exposure.include ile gerekli endpoint'ler (health, prometheus) dışarıya açılır.
* resilience4j.circuitbreaker.instances.petService altındaki parametrelerle devre kesici davranışı yapılandırılır.
* failureRateThreshold: 100 ile başarısızlık oranı %100 olduğunda devre kesici açık konuma geçer.
* waitDurationInOpenState: 5s ile açık durumdan half-open'a geçiş süresi belirlenir.

### pom.xml dosyasına eklemeler



* spring-boot-starter-actuator: Servis sağlık durumu ve metrik bilgileri için gerekli.
* micrometer-registry-prometheus: Prometheus ile uyumlu metrik toplama için.
* resilience4j-spring-boot3: Circuit breaker (devre kesici) kullanımı için gerekli.

# 7. Logback ile Loglama Yapılandırması

### 7.1 Bağımlılık Tanımlaması

Her servisin pom.xml dosyasına aşağıdaki bağımlılık eklenmiştir:

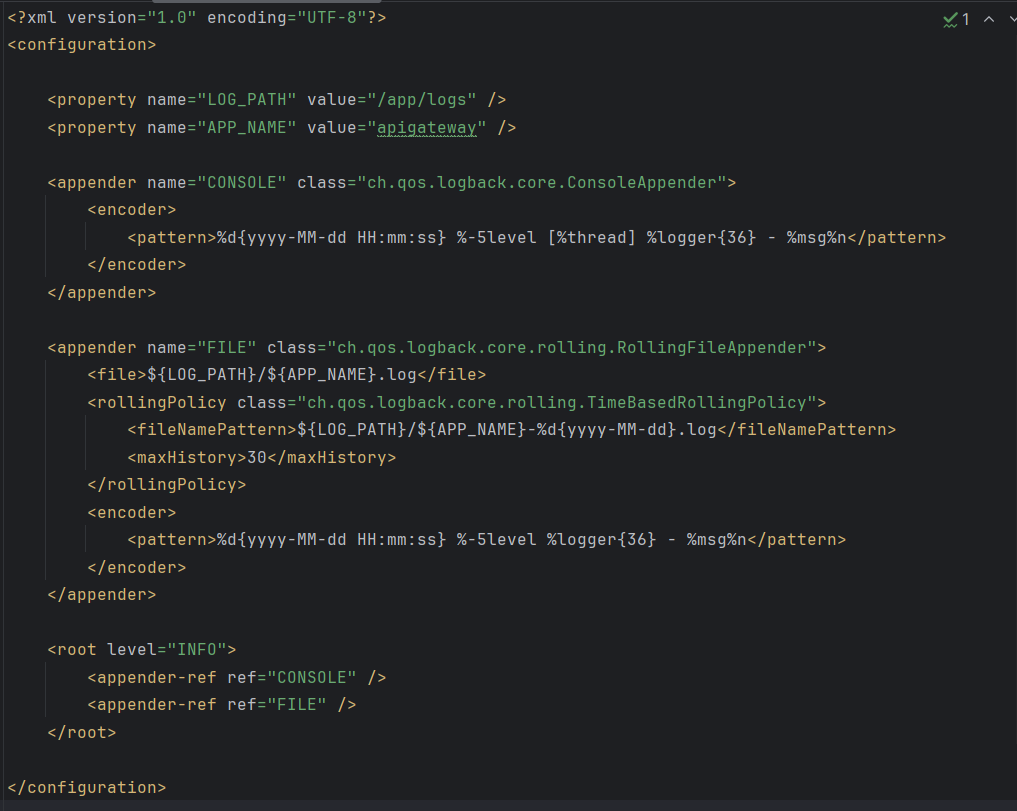
A screen shot of a computer screen

AI-generated content may be incorrect.

Bu bağımlılık sayesinde Logback üzerinden JSON log çıktıları üretilebilir hale gelmiştir.

### 7.2 logback-spring.xml Dosyası Yapılandırması

Her servisin src/main/resources dizinine logback-spring.xml dosyası eklenmiştir. Bu dosyada log yolu ve servis adı parametreler aracılığıyla dinamik olarak tanımlanmıştır:



APP\_NAME her serviste servis adıyla (activityservice, userservice vb.) değiştirilmiştir.

### 7.3 Docker Volume Tanımlaması

Logların dışarı yazılabilmesi için docker-compose.yml dosyasına her servis için aşağıdaki gibi volume tanımı eklenmiştir:

A screen shot of a computer

AI-generated content may be incorrect.

Bu sayede her servis kendi klasörüne log yazmaktadır:

A screenshot of a computer

AI-generated content may be incorrect.

### 7.4 Log Formatı ve Örneği

Loglar, Logstash uyumlu JSON formatında yazılmaktadır. Örnek bir log çıktısı:

{"@timestamp":"2025-07-28T16:37:48.501560519+03:00","

@version":"1",

"message":"Request execution failed with message: null",

"logger\_name":"com.netflix.discovery.shared.transport.decorator.RetryableEurekaHttpClient",

"thread\_name":"DiscoveryClient-CacheRefreshExecutor-1",

"level":"WARN","level\_value":30000}

# 8.Kaynaklar ve Dökümantasyon

* <https://grafana.com/docs/>
* <https://prometheus.io/docs/introduction/overview/>

# 9. Sonuç

Bu kullanım kılavuzu ile PetVerse mikroservislerinin gözlemlenebilirlik altyapısını kurabilir, izleyebilir ve ileri düzey önerilerle sisteminizi geliştirebilirsiniz.

Bu kılavuzda, PetVerse mikroservisleri için gözlemlenebilirlik bileşenlerinin nasıl kurulduğu ve yapılandırıldığı detaylı bir şekilde açıklanmıştır. Prometheus ile metrik toplama ve Grafana ile izleme altyapısı başarıyla kurulmuştur.  
Bu gözlemlenebilirlik yapısı sayesinde:  
• Sistem sağlığı merkezi olarak izlenebilir hale gelmiştir.  
• Servislerin performansı anlık takip edilebilir.  
• Hatalar ya da gecikmeler grafiklerle kolayca tespit edilebilir.  
• Prometheus ve Grafana ile kurulan altyapı Docker Compose üzerinden kolayca ayağa kaldırılabilir.