



# HAVELSAN

## DAĞITIK SENSOR AĞI VE HEDEF LOKALİZASYON

Deniz Türk, Aday Mühendis Adayı  
23 May 2023

---

## PROJE ÖZETİ

### Proje Amacı

Bu projede 4 farklı java uygulamasının birbiri ile Apache Kafka kullanılarak iletişiminin sağlanması ve farklı kaynaklardan gelen verilerin merkezi bir birimde işlenerek anlamlı bir veri ortaya çıkarılması hedeflenmiştir.

### Kullanılan Teknolojiler

- Apache Kafka 3.4.0
- java 17
- Oracle OpenJDK version 17.0.7
- Spring Boot 3.1.0

### Proje Detayları

Bu sistemde 4 adet java uygulaması bulunmaktadır.

- 1) target-generator
  - 2) Sensor1
  - 3) Sensor2
  - 4) Central Processing Unit
-

## Apache Kafka

<https://kafka.apache.org/downloads> adresinden indirilebilir.

1) Zookeeper serisinin çalıştırılması

```
$ bin/zookeeper-server-start.sh config/zookeeper.properties
```

2) Kafka Broker Servisinin Çalıştırılması

```
$ bin/kafka-server-start.sh config/server.properties
```

## Target-Generator Uygulaması

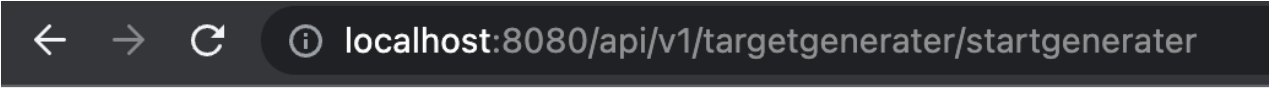
Bu projenin amacı sensörlerin üzerinde algılama ve hedef açısını belirleme işlemlerini gerçekleştireceği dünyanın ve bu dünyada bulunacak hedefin sensörlerden bağımsız bir şekilde üretilerek yayın yapması ve gerçek hayatı taklit etmesi hedeflenmiştir.

Ön tanımlı olarak üzerinde çalışılan alan 1000-1000 olarak tanımlı olup konfigürasyon dosyasından değiştirilebilir. Hedef tespiti için sağlıklı bir şekilde yapılabilmesi için üretilen her veriye bir id değeri atanır ,bu veri 2 farklı sensör tarafından gözlem yapılan zaman verisi gibi kullanılarak hedefin konumunu sağlıklı bir şekilde tespit edebilmek için merkezi birimde veri senkronizasyonunu sağlama amaçlı kullanılacaktır.

**Bu Uygulama'da veri üretimi işlemi Http isteği ile tetiklenecek şekilde yazılmıştır.**

-Herhangi bir doğrulama ve yetki kontrolüne gerek yoktur.Her istek kesikli olarak veri üretimini tetikler (1 istek = 1 veri) ve Apache kafka'da oluşturulan topic'e veriyi gönderir.Üretilen veri geri döndürülür.

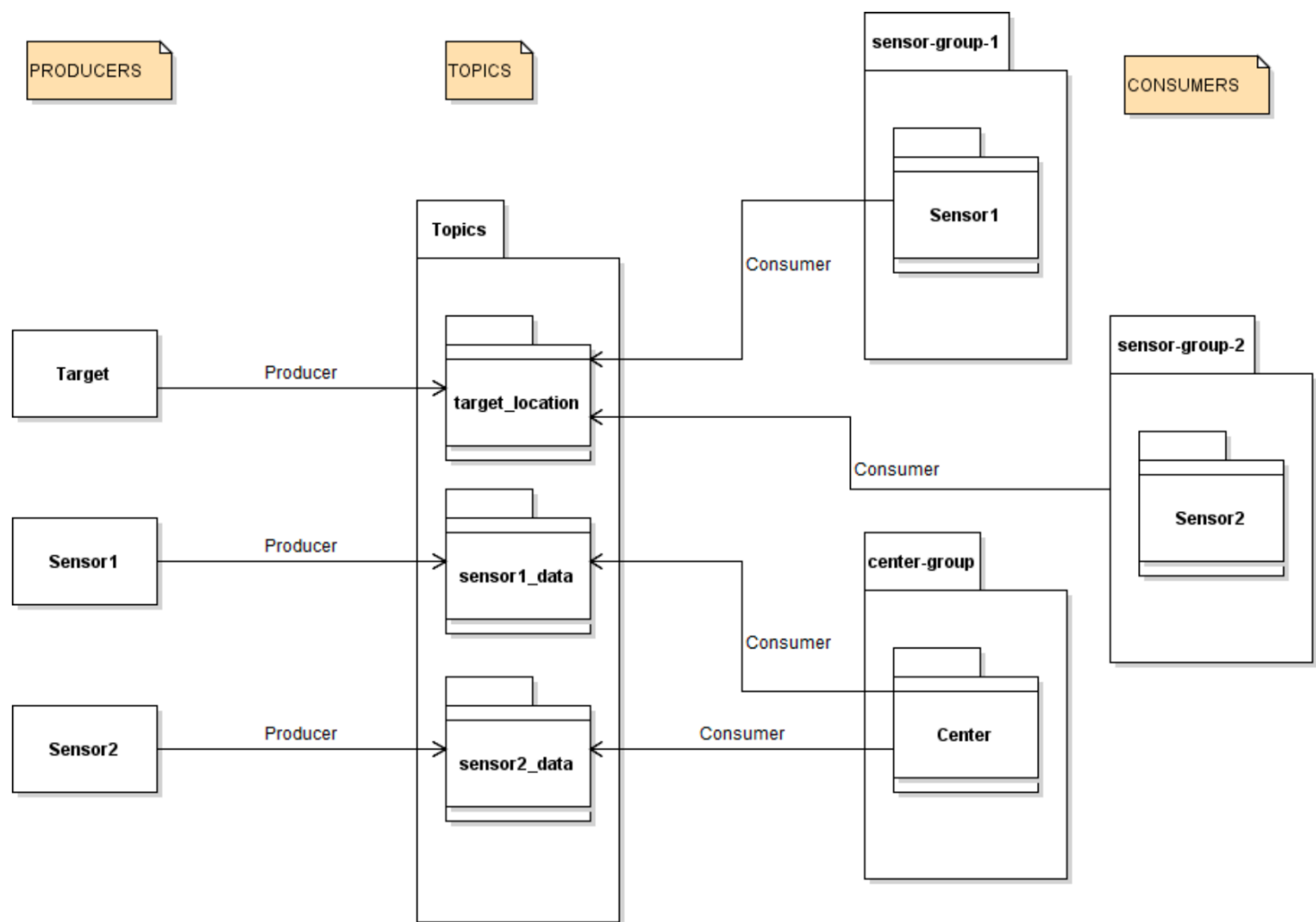
**URL: <http://localhost:8080/api/v1/targetgenerator/startgenerator>**



```
{"id":1,"x":1000,"y":1000,"targetX":889,"targetY":544}
```

---

# Kafka Diyagramı



## Sensör Uygulamaları

Sensör uygulamalarının herhangi biri Apache Kafka'dan veri okuduğunda dünya ve hedef verisinin diğer sensör uygulaması tarafından da tüketilebilmesi için her hedef verisi tüketicisi (sensör uygulamaları) 2 ayrı Consumer Group içerisinde tanımlanmıştır. Uygulamalar çalıştığı anda veriyi tüketen servis çalışmaya başlar. Bu veriler Singleton olarak oluşturulan bir kuyruk yönetimi yapan sınıfın yönettiği bir kuyruk veri yapısına veri geldikçe ittirilir(push).

dünya verisinin işlenerek açılma hesaplamasının yapıldığı ilgili sınıf gözlemci tasarım deseni(observer pattern) kullanılarak kuyruk yönetimi yapan sınıfa abone olur . Kuyruk yönetimi yapan sınıf kuyruğa yeni bir veri eklendiğinde bu sınıfın veriyi işlemek için gerekli olan işlemleri tetiklemiş olur. Tetiklenen bu sınıf ilgili matematiksel işlemleri yaptıktan sonra açılma bilgisini hesaplar ve veri gönderen servisi DI ile enjekte ederek merkezi birimin tüketiceği ilgili topic'e gönderir.

**Tasarımın Uygulanma Sebebi:** Sürekli olarak topic'i dinliyoruz ve tüketilebilecek veri oldukça veriyi kafkadan alıyoruz. Fakat veriyi işleme hızımız , verinin bize gelme hızından yüksek olursa ne olur sorusunu düşündüğümde gelen verilerin kaybedilmemesi için bu şekilde bir tasarım'a ihtiyaç olduğunu düşünerek verinin kaybedilmesini önleyebileceğimi düşündüm. Diğer bir sebep ise kuyruktaki veride değişiklik olduğunda ve değişiklik sonucu hala kuyrukta veri bulunuyorsa ilgili sınıfı tetikleyerek kaynakların sadece gerekli olduğunda kullanılmasını sağlayarak daha verimli çalışan bir sistem oluşturabileceğimi düşündüm. Alternatif çözümleri düşündüğümde program çalışırken bir döngü ile tüketim işlemini sürekli gerçekleştirecek mekanizmaları kurmaktansa, sadece veri olduğunda tetiklenen bu sistemin daha iyi bir çözüm olduğunu düşündüm.

Sensör uygulamaları hem consumer hem de Producer olarak çalışmaktadır.

```
c.h.m.concretes.Sensor2MessageConsumer2 : data received -> {"id":5,"x":1000,"y":1000,"targetX":846,"targetY":292}
c.h.m.concretes.Sensor2MessageProducer2 : Message sent {"sensorId":1,"id":5,"x":1000,"y":1000,"targetX":846,"targetY":292,"isProcessed":1}
c.h.m.concretes.Sensor2MessageConsumer2 : data received -> {"id":6,"x":1000,"y":1000,"targetX":56,"targetY":99}
c.h.m.concretes.Sensor2MessageProducer2 : Message sent {"sensorId":1,"id":6,"x":1000,"y":1000,"targetX":56,"targetY":99,"isProcessed":1}
c.h.m.concretes.Sensor2MessageConsumer2 : data received -> {"id":7,"x":1000,"y":1000,"targetX":160,"targetY":916}
c.h.m.concretes.Sensor2MessageProducer2 : Message sent {"sensorId":1,"id":7,"x":1000,"y":1000,"targetX":160,"targetY":916,"isProcessed":1}
c.h.m.concretes.Sensor2MessageConsumer2 : data received -> {"id":8,"x":1000,"y":1000,"targetX":129,"targetY":257}
c.h.m.concretes.Sensor2MessageProducer2 : Message sent {"sensorId":1,"id":8,"x":1000,"y":1000,"targetX":129,"targetY":257,"isProcessed":1}
c.h.m.concretes.Sensor2MessageConsumer2 : data received -> {"id":9,"x":1000,"y":1000,"targetX":752,"targetY":858}
c.h.m.concretes.Sensor2MessageProducer2 : Message sent {"sensorId":1,"id":9,"x":1000,"y":1000,"targetX":752,"targetY":858,"isProcessed":1}
```

## Central Processing Unit ve Veri Senkronizasyonu

-Bu uygulama 2 farklı sensör uygulamalarından gelen verileri tüketir. Sensör uygulamalarında belirtilen tasarımın benzeri bu uygulamada da uygulanmıştır. Bu uygulamada 2 farklı veri kaynağı olduğu için 2 farklı kuyruk yönetici sınıfı bulunur. Bu sınıflar farklı kaynaklardan gelen verinin senkronizasyonunun sağlanabilmesi için veri işleyen(konum tespit eden) katmandaki ilgili sınıfın fonksiyonlarını doğrudan tetiklemek yerine, önce veri işleyen katmandaki kontrol biriminin ilgili kuyruk sınıfı için oluşturulmuş bayrağını tetikler daha sonra kontrol birimini tetikler. Kontrol birimi her tetiklendiğinde , farklı 2 veri kaynağı için ayrılmış 2 kontrol bayrağı da tetiklenmiş durumda ise her 2 kuyruktan en son veriyi çekerek veri işleme fonksiyonunu çağırır. Herhangi bir hedefin T anındaki konumunu saptamak için her 2 sensörden de T anında yapılan ölçümünü veya kabul ettiğimiz T+,- tolerans aralığında yapılan ölçümlerini beraber işleme alarak konumunu saptamamız gerekir. Sensörler tarafından gelen veriler ağ problemleri ,verilerin gönderilme sıklığı, sensörlerin kabiliyetleri, bağımlı olarak kullandığımız apache kafka platformu veya herhangi bir sebeple merkezi işlem birimine kayıpla ulaşabilir(sensörler tarafından kesikli olarak belli bir zaman aralığında veri aldığımızı kabul ettim) bu durumda bize kontrol birimi tarafından sağlanan kuyruktaki en son verilerin birlikte işleme alınması; herhangi bir veride merkeze gelene kadar kayıp olmuştur , T ve T-x ölçüm zamanlarına sahip verilerin birlikte işleme alınmasına engel olacak kontrolü sağlayamayabilir ve yanlış konum tespiti yapılır. Projemizde ölçüm yapılan zaman id verileri ile simüle edildiği için aynı Id numarasına sahip veriler işleme alınır eğer Id verileri uyuşmuyorsa küçük olan Id deki veriler kalıcı olarak depolanması için farklı bir kuyruğa ittilir ta ki id numaraları birbirine eşlenen kadar. Bu senaryoda zamansal olarak kayıp ettiğimiz bir sensör verisine karşılık diğer sensörün o zaman aralığında hangi ölçümleri yaptığını kaydetmiş oluyoruz. Bu projede uygulanmasa da kaydettiğimiz tek sensör verisi ile veri kaybı yaşadığımız sensörün en son veriyi aldığımız zamandan itibaren geriye dönük belirli zaman aralıklarındaki açı değişimlerinin büyüklüğünden yola çıkarak, gözlemsel olarak bir hedef tahmini yapma kabiliyetimizi kaybettiğimiz durumda tahminsel olarak da hedef tespiti yapabiliriz.Fakat bu yöntem bu projede uygulanmamıştır.

Hesaplama: Her sensörün konumu ve tespit ettiği açı bir ışın meydana getirir.Bu 2 ışının kesişim noktası bize hedefin konumu verir .Dikkate alınması gereken nokta ise 2 ışının birbirine paralel olması durumunda ya 2 ayrı hedef var ya da sensörler hatalı veri üretiyor olabilir bu durumda hedef tespiti yapılamaz.

---

