

ANKARA ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ



FİNAL PROJESİ RAPORU

(BLM3522-A) BULUT BİLİŞİM VE UYGULAMALARI

Rustem TUKHBETOV (21291001)
Perizat SAGYNBEKOVA (21290895)
Derda SİNA GÜNAY (20291274)

02.07.2025

LINKLER:

1. GitHub: <https://github.com/rustemio>
2. Anlatım videosu: <https://disk.yandex.ru/d/E2Bwe6Utk9Epbw>
3. Cloud linki: <https://gorev-backend-rustemio.azurewebsites.net/tasks>

PROJE 1: IOT VE AKILLI ŐEHİR UYGULAMASI

Backend Dili: Python (IoT cihazları ile veri gönderme/simölasyon için)

Bulut Platformu: AWS (Amazon Web Services)

Problem: Python ile AWS Tabanlı Akıllı Aydınlatma Uygulaması

AMAÇ: Akıllı Őehir uygulamaları, Őehir yaşamını daha sürdürölabilir, verimli ve kullanıcı dostu hale getirmeyi amaçlayan çölömlerdir. Bu projede, akıllı aydınlatma sistemi ele alınmıřtır. Python ile geliştirilen IoT tabanlı Akıllı Aydınlatma Sistemi, hareket sensörleri ve ışık Őiddeti sensörlerinden gelen verileri AWS IoT Core üzerinden işleyerek, sokak lambalarının otomatik olarak açılıp kapanmasını ve parlaklık ayarını optimize edecektir. Amaç; sokak lambalarının enerji verimliliğini artırmak, gereksiz çalışmayı engellemek ve uzaktan yönetilebilir hale getirmektir.

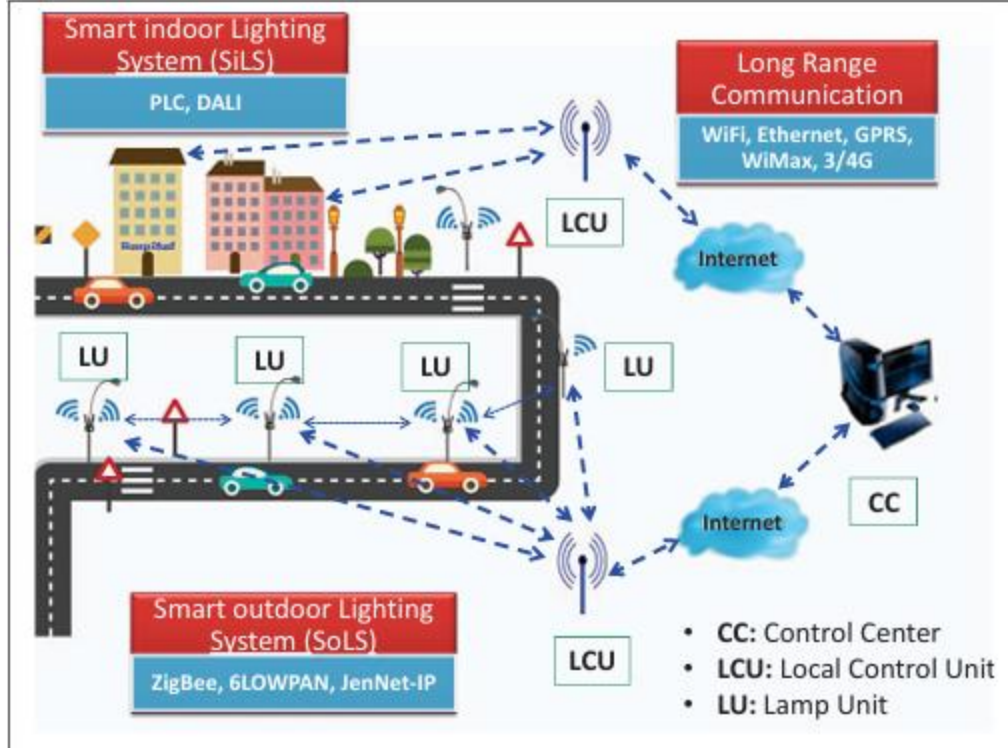
GİRİŐ

Günümüzde kentleşmenin hızla artması, Őehirlerdeki yaşam kalitesinin iyileřtirilmesi için daha akıllı ve verimli teknolojilere olan ihtiyacı da artırmaktadır. Bu bağlamda, akıllı Őehir kavramı; enerji, ulaşım, sağık gibi temel altyapıların bilgi ve iletişim teknolojileri (ICT) ile entegre edilmesini hedefleyen yenilikçi bir yaklaşımdır. Nesnelerin İnterneti (IoT) ise bu dönüşümün merkezinde yer almakta, farklı cihazların birbiriyle bağlantılı ve otonom Őekilde çalışmasını mümkün kılmaktadır.

Őehir altyapısı içerisinde önemli miktarda enerji tüketen sistemlerden biri olan aydınlatma, akıllı Őehir uygulamalarının öncelikli hedeflerinden biridir. Yapılan arařtırmalara göre, sadece kamusal alanlardaki aydınlatma sistemleri bile Őehirlerin toplam enerji tüketiminin yaklaşık %10'unu oluřturmaktadır. Bu enerji israfının önüne geçmek amacıyla geliştirilen Akıllı Aydınlatma Sistemleri (Smart Lighting Systems – SLS), hareket ve ışık sensörleri gibi bileřenlerle desteklenerek, aydınlatma yoğunluğunu çevresel kořullara göre otomatik olarak ayarlayabilmektedir.

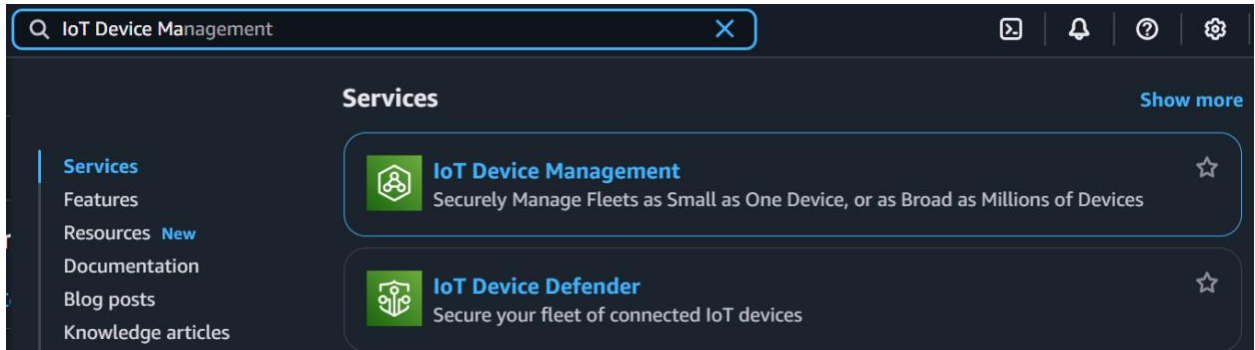
Bu projede, AWS platformu kullanılarak Python dili ile geliştirilen simölatif bir IoT tabanlı akıllı aydınlatma uygulaması sunulmaktadır. Uygulamada, bir akıllı lambadan (simölasyon cihazı) toplanan sensör verileri MQTT protokolü üzerinden AWS IoT Core'a iletilmekte, oradan da Lambda fonksiyonu aracılığıyla işlenip DynamoDB veritabanına kaydedilmektedir. Bu yapı sayesinde veriler hem analiz edilebilmekte hem de gerçek zamanlı olarak izlenebilmektedir.

Akıllı Őehirlerde akıllı aydınlatma sistemleri [1]:



1. Her Sokak Lambası Bir Iot Cihazı Gibi Düşünülür

İnsan varlığını algılayan hareket sensörü (PIR), ortam aydınlığını ölçen ışık şiddeti sensörü (LDR) ve aydınlatma ayarı yapan LED Strip (PWM Control) gibi gerçek fiziksel IoT cihazları (Raspberry Pi/ESP32) elimizde bulunmadığından bu projede AWS IoT Core ile yazılımsal sanal cihaz simülasyonu yapıyor olacağız.



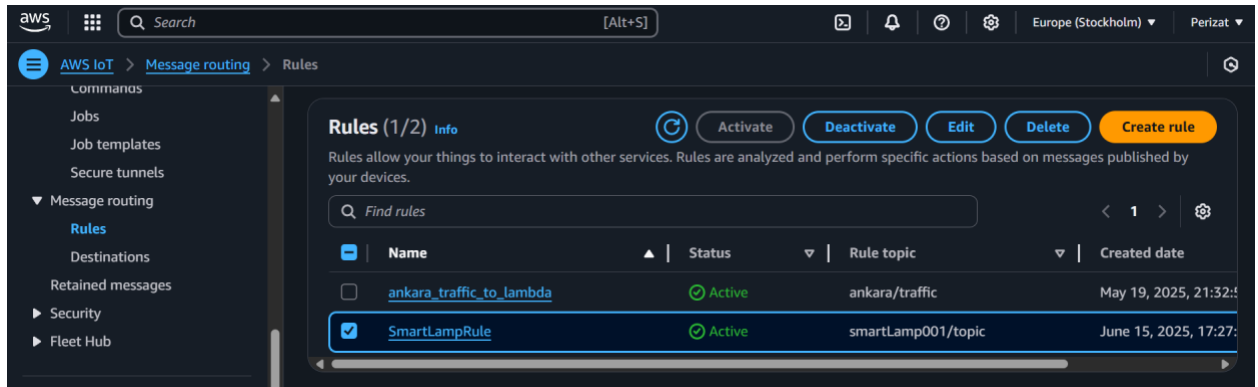
smartLamp001 adında bir nesne oluşturuyoruz:



Cihaz oluřtururken gerekli **.pem** dosyalarını (private key, public key, certificates) indiriyoruz.

2. Lambalar, Iřık Seviyesi (Lux), Hareket Algılama ve Enerji Tüketimi Gibi Verileri AWS Iot Core'a Gönderir

AWS IoT Core yapılandırması yaparken IoT cihazlarından gelen verileri (MQTT mesajları) alıp, başka bir yere (Lambda, DynamoDB, S3, vs.) aktarma yapmak için yeni bir Rule oluřturuyoruz:



Aktarma action'ı (eylem) olarak lambda fonksiyonunu seğıyoruz ve gerekli SQL komutlarını gerekli yere yazıyoruz:

SQL statement

SQL statement

SQL version

SELECT * FROM 'smartLamp001/topic'

2016-03-23

Actions

Error action

Tags

Actions (1)

View details

Actions occur when an event is triggered. Actions are executed until all actions are completed or an error occurs. To add or remove actions, you will need to edit the rule.

Service	Action
<input type="radio"/> Lambda	Send a message to a Lambda function

Python ile MQTT veri gönderimi (simülasyon) için VSCode'da bir Python dosyası oluşturuyoruz ve gerekli MQTT kodlarını yazıyoruz:

```
mqtt_sender.py 1 X
C: > Belgeler(Yurt, Üniversite, ...) > University(Ankara) > TERM6 > BLM3522 > mqtt > mqtt_sender.py > ...
1 from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
2 import json
3 import time
4
5 client = AWSIoTMQTTClient("SmartLampClient")
6 client.configureEndpoint("ayuovu91larx0-ats.iot.eu-north-1.amazonaws.com", 8883)
7 client.configureCredentials("AmazonRootCA1.pem", "2fc4c5a335ae3fc0d8f6c2aa8e50b27429123a920104e0e82c16fec8be", "2fc4c5a335ae3fc0d8f6c2aa8e50b27429123a920104e0e82c16fec8be")
8 client.connect()
9
10 topic = "smartLamp001/topic"
11
12 for i in range(3):
13     # payload = {
14     #     "device_id": "SmartLamp_001",
15     #     "light_level": 320,
16     #     "motion": True
17     # }
18
19     import random
20     payload = {
21         "device_id": "SmartLamp_001",
22         "light_level": random.randint(0, 100), # 0-100 arasında rasgele sayı
23         "motion": random.choice([True, False]) # True veya False rastgele
24     }
25     client.publish(topic, json.dumps(payload), 1)
26     print(f"Published: {payload}")
27     time.sleep(5)
```

Ortamın ne kadar aydınlık olduğunu gösteren light_level (ışık seviyesi) Lux (lüks) birimi ölçümünü kullanmaktadır.

Lambda fonksiyonuna isim vererek ve Python dilini seçerek yeni bir function oluşturuyoruz:

Lambda > Functions > Create function

Basic information

Function name

Enter a name that describes the purpose of your function.

myFunctionName

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime [Info](#)

Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Python 3.13

Architecture [Info](#)

Choose the instruction set architecture you want for your function code.

☐ arm64

☒ x86_64

Permissions [Info](#)

By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

3. Veriler AWS Lambda Aracılığıyla Analiz Edilir

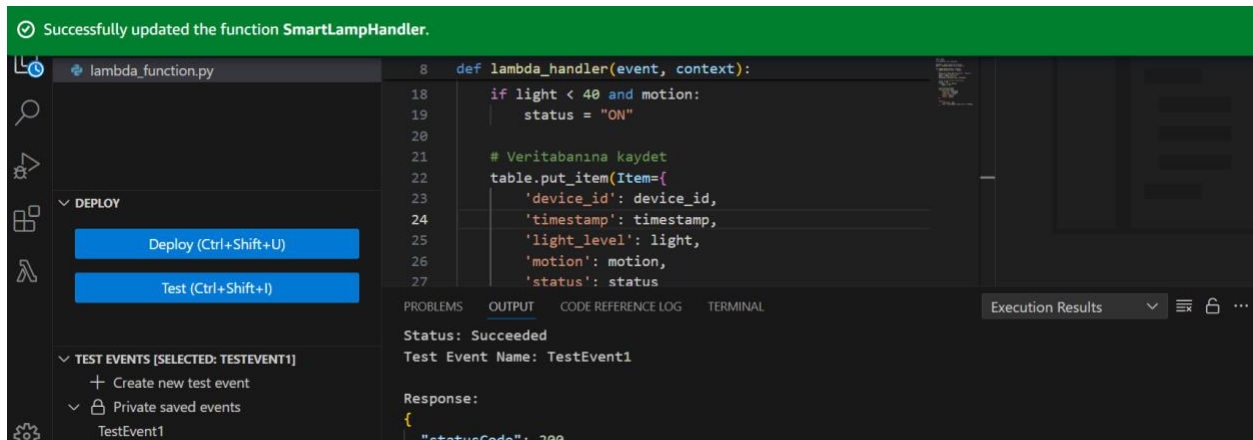
Eğer hava karanlık ve hareket algılandıysa → lambda açılır.

Eğer gündüzse veya hareket yoksa → lambda kapatılır.

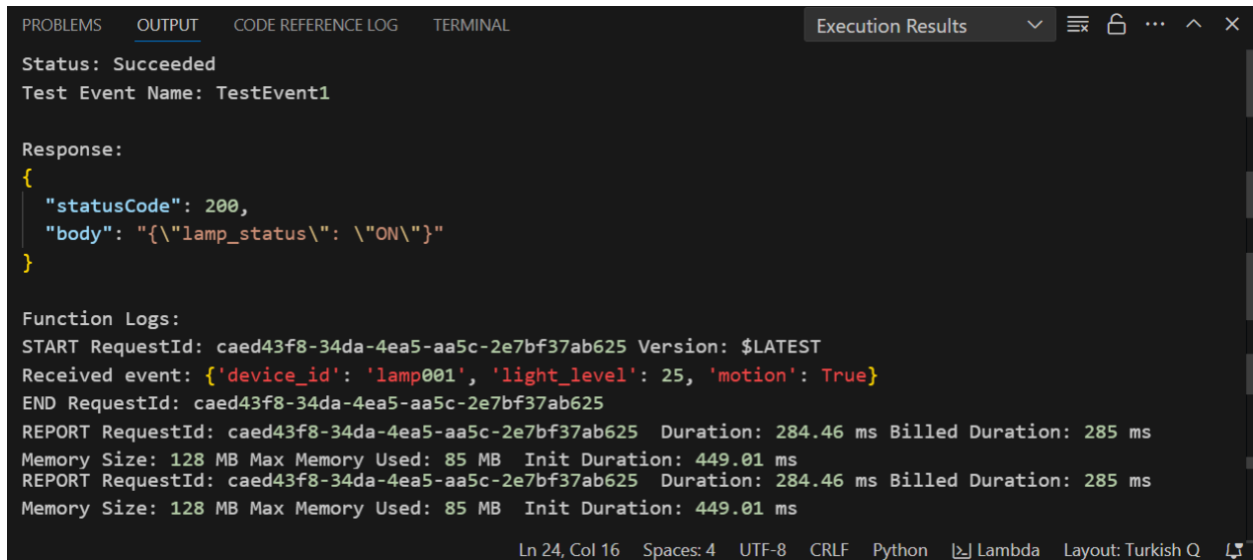
Akıllı aydınlatma sisteminizin beyni olarak çalışan Lambda fonksiyonu, sensör verilerini analiz ederek aydınlatma kurallarını uygular ve DynamoDB'ye alınan/işlenen verileri kaydeder:

```
lambda_function.py
1  import json
2  import boto3
3  from datetime import datetime
4
5  dynamodb = boto3.resource('dynamodb')
6  table = dynamodb.Table('SmartLampData')
7
8  def lambda_handler(event, context):
9      print("Received event:", event)
10
11      device_id = event.get('device_id', 'unknown')
12      light = event['light_level']
13      motion = event['motion']
14      timestamp = datetime.now().isoformat()
15
16      # Veri işleme ve karar mekanizması
17      status = "OFF"
18      if light < 40 and motion:
19          status = "ON"
20
21      # Veritabanına kaydet
22      table.put_item(Item={
23          'device_id': device_id,
24          'timestamp': timestamp,
25          'light_level': light,
26          'motion': motion,
27          'status': status
28      })
29
30      return {
31          'statusCode': 200,
32          'body': json.dumps({'lamp_status': status})
33      }
```

Deploy düğmesine basarak yaptığımız değişiklikleri kaydedebiliriz ve Test düğmesine basarak da kodun doğru çalışıp çalışmadığını görebiliriz:



Lambda fonksiyonunu test ederken elle verdiğimiz test verisinden (test event), light_level = 25 ve motion = True olduğu için şartı sağlandı → lambda "ON" olarak belirlendi:



4. Durum ve Log Verileri DynamoDb'ye Kaydedilir

AWS Management Console üzerinden DynamoDB servisini açarak yeni bir Table oluşturuyoruz:

[DynamoDB](#) > [Tables](#) > Create table

Create table

Table details [Info](#)

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name
This will be used to identify your table.

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

1 to 255 characters and case sensitive.

Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

1 to 255 characters and case sensitive.

mqtt_sender.py dosyasını CMD üzerinden çalıştırıyoruz ve üretilen değerleri görebiliyoruz:

```
M6\BLM3522\mqtt>python mqtt_sender.py
Published: {'device_id': 'SmartLamp_001', 'light_level':
48, 'motion': False}
Published: {'device_id': 'SmartLamp_001', 'light_level':
81, 'motion': False}
Published: {'device_id': 'SmartLamp_001', 'light_level':
71, 'motion': True}
```

Aynı zamanda Explore Table Items seçeneğine tıklayarak yeni gelen verileri canlı olarak gözlemleyebiliyoruz:

Table: SmartLampData - Items returned (3)

[Refresh](#) [Actions](#) [Create item](#)

Scan started on June 22, 2025, 18:02:04

device_id (String)	light_level	motion	status
SmartLamp_001	48	false	OFF
SmartLamp_001	81	false	OFF
SmartLamp_001	71	true	OFF

SONUÇ

Bu proje kapsamında geliştirilen akıllı aydınlatma sistemi, temel bir IoT mimarisi üzerinden AWS bulut servisleriyle başarıyla entegre edilmiştir. Simülasyon ortamında MQTT protokolü aracılığıyla oluşturulan ışık şiddeti ve hareket verileri AWS IoT Core'a iletilmiş, IoT kuralları sayesinde Lambda fonksiyonuna yönlendirilmiş ve burada işlenerek DynamoDB veritabanında kaydedilmiştir. Python ile simüle edilen cihaz verileri sayesinde sistemin farklı senaryolarda nasıl tepki verdiği gözlemlenebilmiş, belirli eşiklerin (örneğin ışık seviyesi 40'ın altına düştüğünde ve hareket algılandığında) lambanın "ON" durumuna geçmesi sağlanmıştır.

Sistem, çevresel verilere göre sokak lambalarının otomatik olarak açılıp kapanmasını sağlamakta ve bu sayede hem enerji tasarrufu hem de uzaktan izlenebilirlik avantajı sunmaktadır. Gerçek fiziksel cihazlar (ör. Raspberry Pi, LDR sensör, PIR sensör) ile genişletildiğinde, proje pratik uygulamalara dönüştürülebilecek niteliktedir. Bulut bilişim kaynaklarının sunduğu ölçeklenebilirlik ve güvenilirlik, bu tarz akıllı şehir çözümleri için büyük bir avantaj sunmaktadır.

KAYNAKLAR

1. A. K. Sikder, A. Acar, H. Aksu, A. S. Uluagac, K. Akkaya and M. Conti, "IoT-enabled smart lighting systems for smart cities," 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2018, pp. 639-645, doi: 10.1109/CCWC.2018.8301744. keywords: {Lighting;Protocols;Smart cities;Intelligent sensors;Wireless communication;Smart Lighting;Internet of Things;Smart City}
2. AWS Documentation: <https://docs.aws.amazon.com/>
3. GitHub: <https://github.com/Impasbaa/BulutBilisim>
4. YouTube: <https://youtu.be/JVRNKY-bnjl?si=hqfi95IjIc2smjRv>

PROJE 2: GERÇEK ZAMANLI SICAKLIK VERİLERİ İLE ANLIK KARŞILAŞTIRMA RAPORU

1. GİRİŞ

1.1. Problem Tanımı

Günümüzde gerçek zamanlı veri akışı ve analizi giderek daha fazla önem kazanmaktadır. Özellikle hava durumu verilerinin hızlı bir şekilde alınması ve değerlendirilmesi, çeşitli sektörler (tarım, ulaşım, enerji vb.) için kritik öneme sahiptir. Bu proje kapsamında İstanbul ve Ankara şehirlerinin anlık sıcaklık verilerinin gerçek zamanlı olarak elde edilmesi ve bu verilerin sürekli karşılaştırılarak hangi şehrin daha sıcak olduğunun belirlenmesi hedeflenmiştir.

1.2. Projenin Hedefleri

- Gerçek zamanlı sıcaklık verilerini güvenilir ve stabil bir şekilde elde etmek.
- Bu verileri güvenli ve sürekli erişilebilir şekilde saklamak ve analiz etmek.
- Otomatik çalışan ve müdahale gerektirmeyen bir sistem kurmak.
- Verileri kullanıcı dostu bir şekilde sunmak ve yorumlamak.
- Cloud Functions, Pub/Sub ve BigQuery gibi serverless bulut araçlarını kullanarak sistemin basit ve ölçeklenebilir olmasını sağlamak.

2. PROJE AKIŞI (AŞAMALAR)

2.1. Proje Altyapısının Kurulumu

Google Cloud Platform (GCP) üzerinde “temperature” adlı bir proje oluşturulmuştur. Projenin altyapı gereksinimleri için Cloud Shell aracı kullanılmış ve gerekli gizli değişkenler Secret Manager üzerinden yönetilmiştir.

2.2. Verilerin Alınması (Open-Meteo API)

Veri kaynağı olarak Open-Meteo API tercih edilmiştir. API üzerinden JSON formatında veriler çekilmiştir: - Şehir adı (“sensor_id”) - Sıcaklık değeri (“temp”) - Zaman damgası (“ts”)

Python dilinde yazılan main.py script’i sayesinde API’den veriler periyodik olarak alınarak işlenmiştir.

2.3. Pub/Sub Topic Yapılandırması

Google Pub/Sub servisi kullanılarak “sensor-raw” isimli bir topic oluşturulmuştur. Bu topic, elde edilen verilerin anlık olarak iletilmesini ve dağıtılmasını sağlamıştır.

←

sensor-raw-bq-sub

EDIT

CREATE SNAPSHOT

REPLAY MESSAGES

PURGE

Subscription name

projects/tempeature/subscriptions/sensor-raw-bq-sub

Subscription state

active

Topic name

projects/tempeature/topics/sensor-raw

METRICS

DETAILS

MESSAGES

Delivery type

Write to BigQuery (Push)

BigQuery table

tempeature.iot.temp_readings

Schema used to write messages

BigQuery table schema

Write metadata

Disabled

Drop unknown fields

Enabled

Subscription expiration

Subscription expires in 31 days if there is no activity.

Acknowledgement deadline

10 seconds

Subscription filter

—

Subscription message retention duration

7 days

Topic message retention duration

—

Retain acknowledged messages

No

Exactly once delivery

Disabled

Message ordering

Disabled

Dead lettering

Disabled

Retry policy

Retry immediately

Labels

—

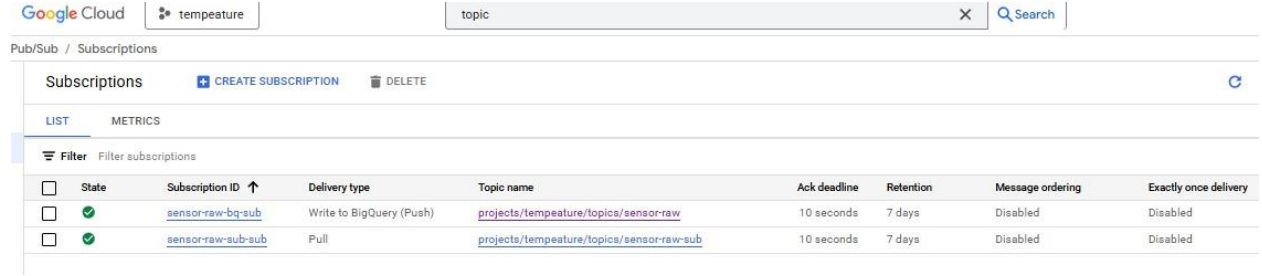
Şekil 1. sensor_raw-bq-sub detayları.

2.4. Cloud Functions (2. nesil) Yapılandırılması

Cloud Functions (Gen2) kullanılarak Python script'i bulut ortamına deploy edilmiştir. Deploy işlemi sırasında: - Avrupa bölgesi (Frankfurt) seçilmiştir. - Python 3.12 runtime kullanılmıştır. - Gizli değişkenler Secret Manager'dan güvenli şekilde alınmıştır. - HTTP tetikleyici kullanılmıştır.

2.5. Verilerin BigQuery'ye Aktarımı (Subscription Aracılığıyla)

Pub/Sub topic'ine gelen verileri doğrudan BigQuery'ye aktarmak için “BigQuery subscription” yöntemi kullanılmıştır. Bu yöntem sayesinde Dataflow gibi ek hizmetlere gerek kalmadan mesajlar otomatik olarak BigQuery'de saklanmıştır.



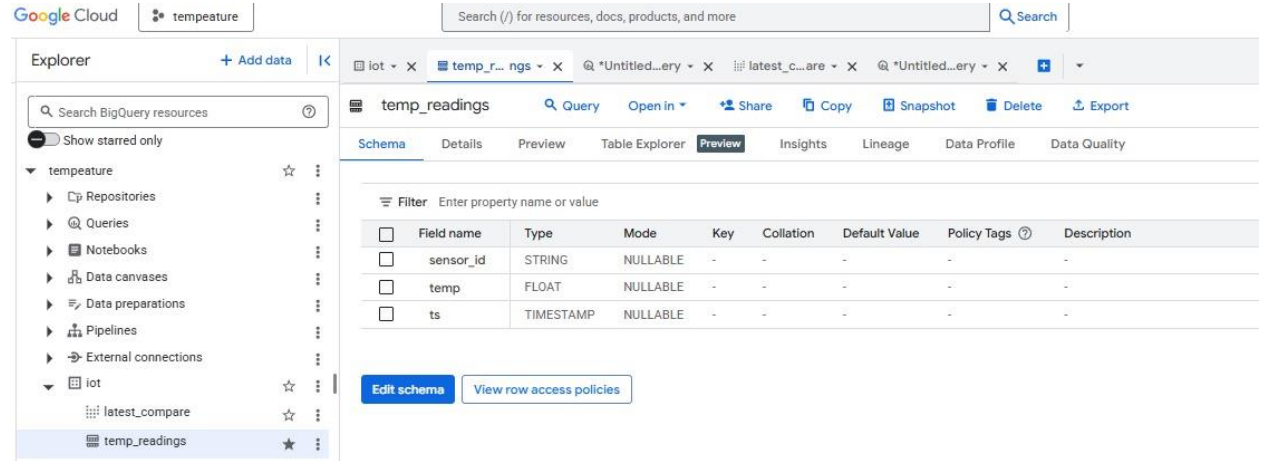
The screenshot shows the Google Cloud Pub/Sub Subscriptions page. At the top, there's a search bar with 'temperature' and a 'topic' filter. Below the search bar, there's a 'Subscriptions' section with 'CREATE SUBSCRIPTION' and 'DELETE' buttons. The 'LIST' tab is active, showing a table of subscriptions. The table has columns: State, Subscription ID, Delivery type, Topic name, Ack deadline, Retention, Message ordering, and Exactly once delivery. Two subscriptions are listed: 'sensorraw-bq-sub' and 'sensorraw-sub-sub'.

State	Subscription ID	Delivery type	Topic name	Ack deadline	Retention	Message ordering	Exactly once delivery
<input checked="" type="checkbox"/>	sensorraw-bq-sub	Write to BigQuery (Push)	projects/temperature/topics/sensorraw	10 seconds	7 days	Disabled	Disabled
<input checked="" type="checkbox"/>	sensorraw-sub-sub	Pull	projects/temperature/topics/sensorraw-sub	10 seconds	7 days	Disabled	Disabled

Şekil 2. Subscription.

2.6. BigQuery'de Veri Saklama ve Analizi

BigQuery üzerinde “iot.temp_readings” adlı tablo oluşturularak gelen veriler sürekli olarak bu tabloda saklanmıştır. Karşılaştırmalı analiz için “iot.latest_compare” VIEW'i oluşturulmuştur.



The screenshot shows the Google Cloud BigQuery interface. On the left, the 'Explorer' panel shows the 'temp_readings' table under the 'iot' dataset. The main panel shows the 'temp_readings' table schema. The schema table has columns: Field name, Type, Mode, Key, Collation, Default Value, Policy Tags, and Description. The fields are: sensor_id (STRING, NULLABLE), temp (FLOAT, NULLABLE), and ts (TIMESTAMP, NULLABLE).

Field name	Type	Mode	Key	Collation	Default Value	Policy Tags	Description
sensor_id	STRING	NULLABLE	-	-	-	-	-
temp	FLOAT	NULLABLE	-	-	-	-	-
ts	TIMESTAMP	NULLABLE	-	-	-	-	-

Şekil 3. temp_readings, tablo formatı.

3. ÇIKTILAR

3.1. Cloud Functions Çıktısı

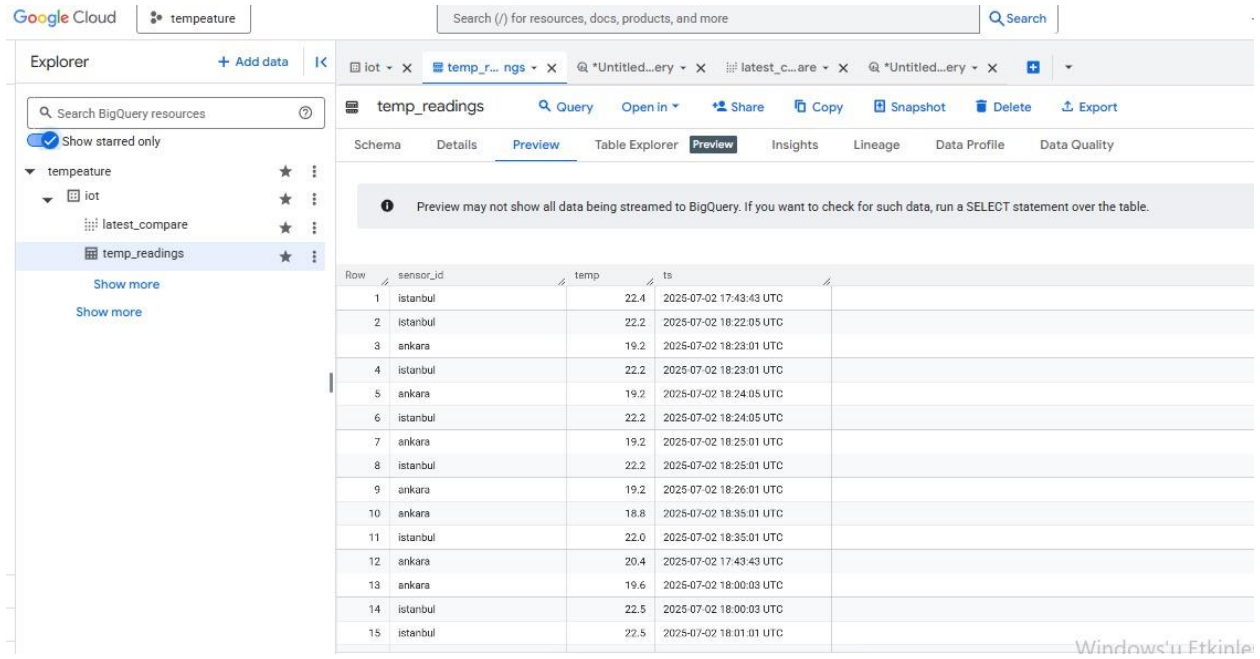
Deploy sonrası oluşturulan Cloud Functions URL'si sayesinde fonksiyon manuel ve otomatik olarak tetiklenebilmiştir.

3.2. Pub/Sub Topic Çıktısı

Pub/Sub servisi üzerinde JSON mesajları başarıyla gönderilmiş ve örnek mesajlar elde edilmiştir.

3.3. BigQuery Tablo Çıktısı

“iot.temp_readings” tablosunda verilerin doğru ve düzenli olarak aktığı görülmüş ve veri akışı doğrulanmıştır.



Google Cloud | temperature | Search (/) for resources, docs, products, and more | Search

Explorer | + Add data | K

Search BigQuery resources | Show starred only

temperature | iot | latest_compare | temp_readings | Show more | Show more

temp_readings | Query | Open in | Share | Copy | Snapshot | Delete | Export

Schema | Details | Preview | Table Explorer | Insights | Lineage | Data Profile | Data Quality

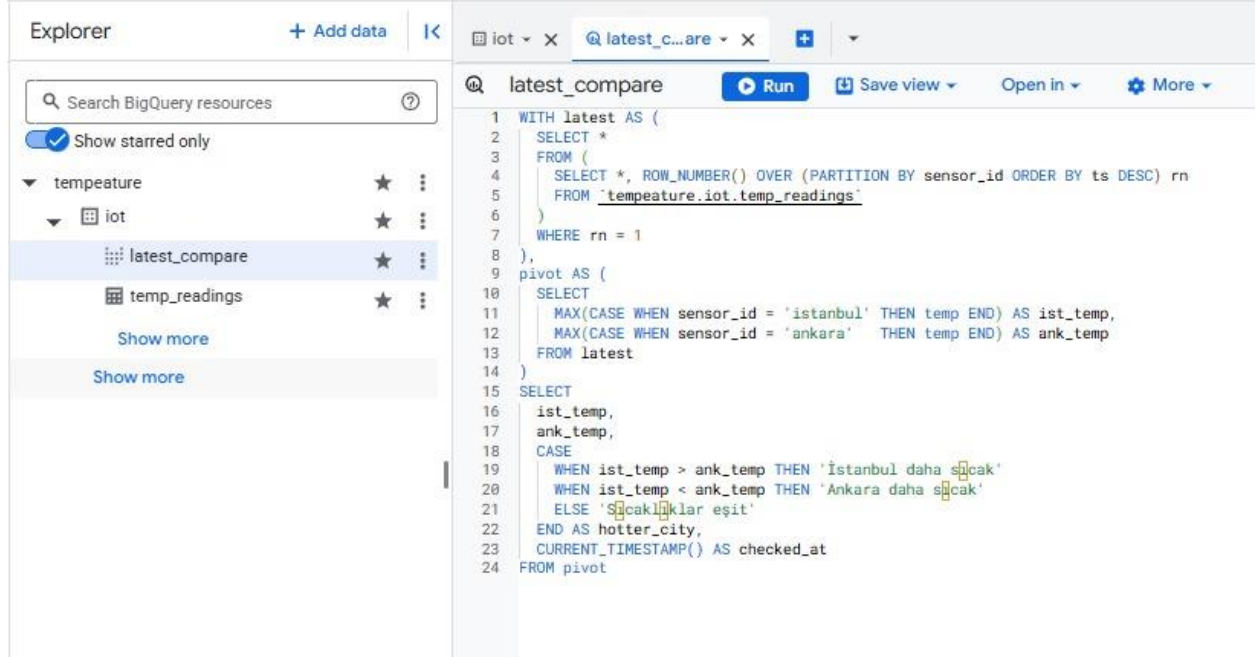
Preview may not show all data being streamed to BigQuery. If you want to check for such data, run a SELECT statement over the table.

Row	sensor_id	temp	ts
1	istanbul	22.4	2025-07-02 17:43:43 UTC
2	istanbul	22.2	2025-07-02 18:22:05 UTC
3	ankara	19.2	2025-07-02 18:23:01 UTC
4	istanbul	22.2	2025-07-02 18:23:01 UTC
5	ankara	19.2	2025-07-02 18:24:05 UTC
6	istanbul	22.2	2025-07-02 18:24:05 UTC
7	ankara	19.2	2025-07-02 18:25:01 UTC
8	istanbul	22.2	2025-07-02 18:25:01 UTC
9	ankara	19.2	2025-07-02 18:26:01 UTC
10	ankara	18.8	2025-07-02 18:35:01 UTC
11	istanbul	22.0	2025-07-02 18:35:01 UTC
12	ankara	20.4	2025-07-02 17:43:43 UTC
13	ankara	19.6	2025-07-02 18:00:03 UTC
14	istanbul	22.5	2025-07-02 18:00:03 UTC
15	istanbul	22.5	2025-07-02 18:01:01 UTC

Şekil 4. sub to bigquery ile table_readings'e yazılan veriler.

3.4. BigQuery VIEW Sonucu

“iot.latest_compare” VIEW’i sayesinde sorgu yapılarak anlık karşılaştırma sonucu elde edilmiştir (örneğin: “İstanbul daha sıcak”).



Şekil 5. latest_compare tablolarına ait sql sorgusu.

4. ÇIKTILARIN YORUMU

4.1. Teknik Yorumlama

Cloud Functions, Pub/Sub ve BigQuery arasında kurulan bağlantıların stabil ve sorunsuz çalıştığı gözlemlenmiştir. Otomatik sistemin güvenilirliği ve sürekli çalışabilirliği doğrulanmıştır.

4.2. Verilerin Anlamlılığı

Alınan veriler gerçek zamanlı olarak karşılaştırılarak tutarlı ve anlamlı sonuçlar üretilmiştir. Bu verilerin farklı sektörlerde kullanılabilirliği değerlendirilmiştir.

5. ELDE EDİLEN SONUÇLAR

5.1. Başlangıçta Belirtilen Hedeflere Ulaşıldı mı?

Proje kapsamında belirlenen tüm teknik ve işlevsel hedeflere başarılı bir şekilde ulaşılmıştır. Otomatik ve müdahale gerektirmeyen çalışma prensibi sağlanmıştır.

5.2. Karşılaşılan Sorunlar ve Çözümleri

Proje sırasında karşılaşılan servis hesabı yetkilendirme sorunları ve BigQuery subscription entegrasyon problemleri, IAM izinlerinin doğru tanımlanmasıyla hızlı bir şekilde çözülmüştür.

5.3. Projenin Güçlü ve Zayıf Yönleri

Projenin en güçlü yanı, serverless mimarisi sayesinde düşük maliyetli, yüksek performanslı ve ölçeklenebilir olmasıdır. İyileştirme yapılabilecek alan olarak ise daha kapsamlı veri kaynaklarının eklenmesi belirtilebilir.

6. ÖNERİLER

6.1. Mevcut Proje İçin Teknik Öneriler

Verilerin gerçek zamanlı olarak görselleştirilmesi için Looker Studio entegrasyonu yapılabilir. Ek şehir ve sensör verileriyle kapsam genişletilebilir.

6.2. Gelecekteki Geliştirmeler İçin Stratejik Öneriler

Yapay zekâ kullanılarak sıcaklık tahmini ve analizleri yapılabilir.

Kritik sıcaklık değişimlerinde otomatik uyarı ve bildirim sistemleri kurulabilir.

Cloud Run üzerine basit bir web arayüzü geliştirilerek sonuçlar görselleştirilebilir.

PROJE 3: GÖREV YÖNETİM UYGULAMASI

Kullanılan Platformlar:

Frontend: React (Netlify üzerinden yayınlandı)

Backend: Flask (Python, Microsoft Azure App Service üzerinde barındırıldı)

Versiyon Kontrol: Git & GitHub

Dağıtım: Netlify, Azure, GitHub

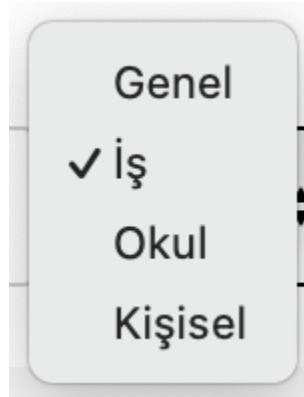
Proje Teknik Adımları:

1. React uygulaması frontend/ dizininde geliştirildi
 2. Backend Flask ile yazıldı ve Azure'a deploy edildi
 3. Axios ile frontend & backend bağlantısı kuruldu
 4. CORS yapılandırıldı
 5. DELETE, POST, GET endpoint'leri oluşturuldu
 6. Netlify'da redirects ayarı yapıldı
- Uygulama üst kısmında tarih, zaman, hava durumu bulunmaktadır. Sağda ise Karanlık/Aydınlık modları mevcuttur.

11 Mayıs 2025 Pazar 11:30:07 — Hava: 🌤️ 22°C (Ankara)

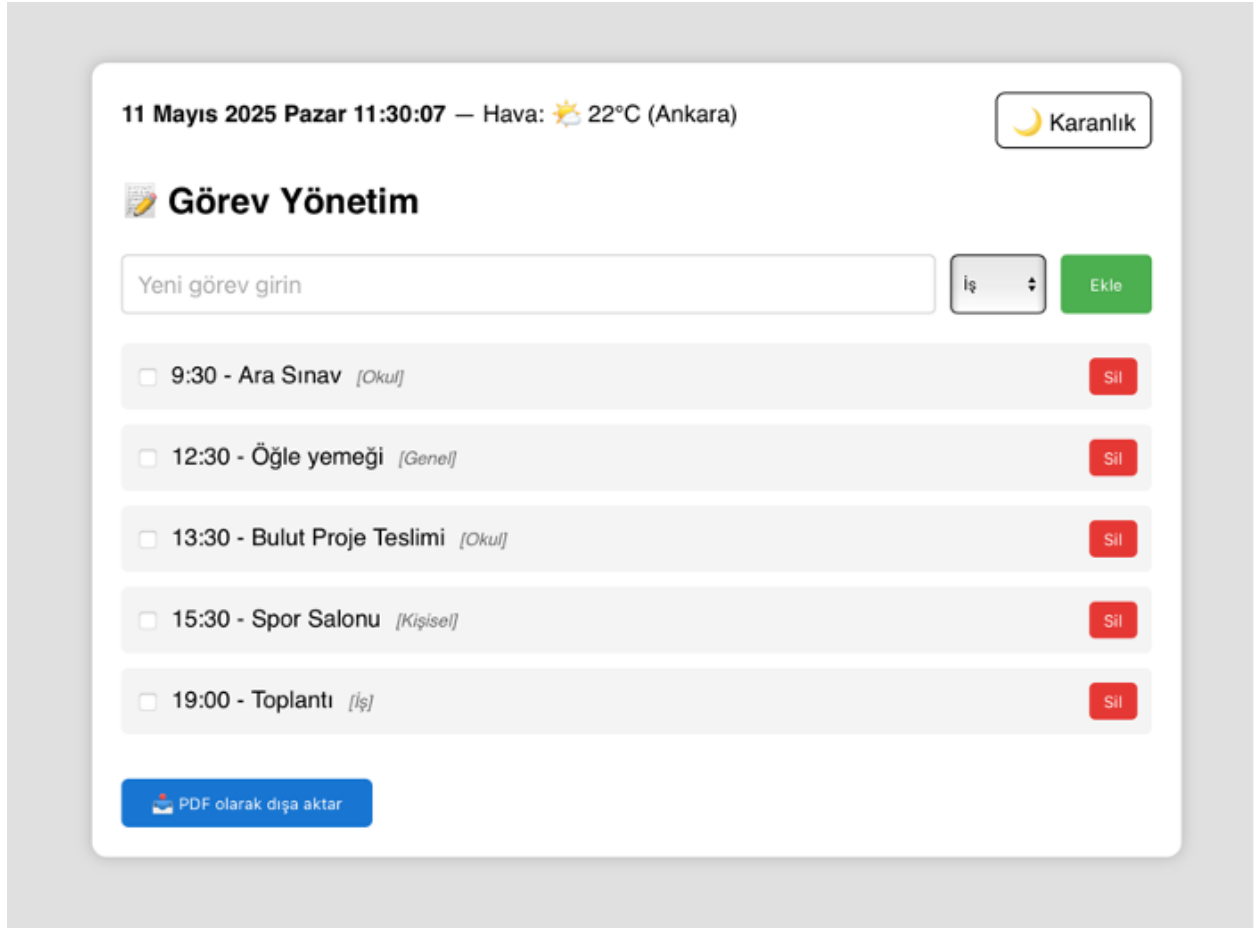
🌙 Karanlık

- Kullanıcı kendi görevlerini 4 kategoriye ayırabilir – “Genel”, “İş”, “Okul” ve Kişisel:

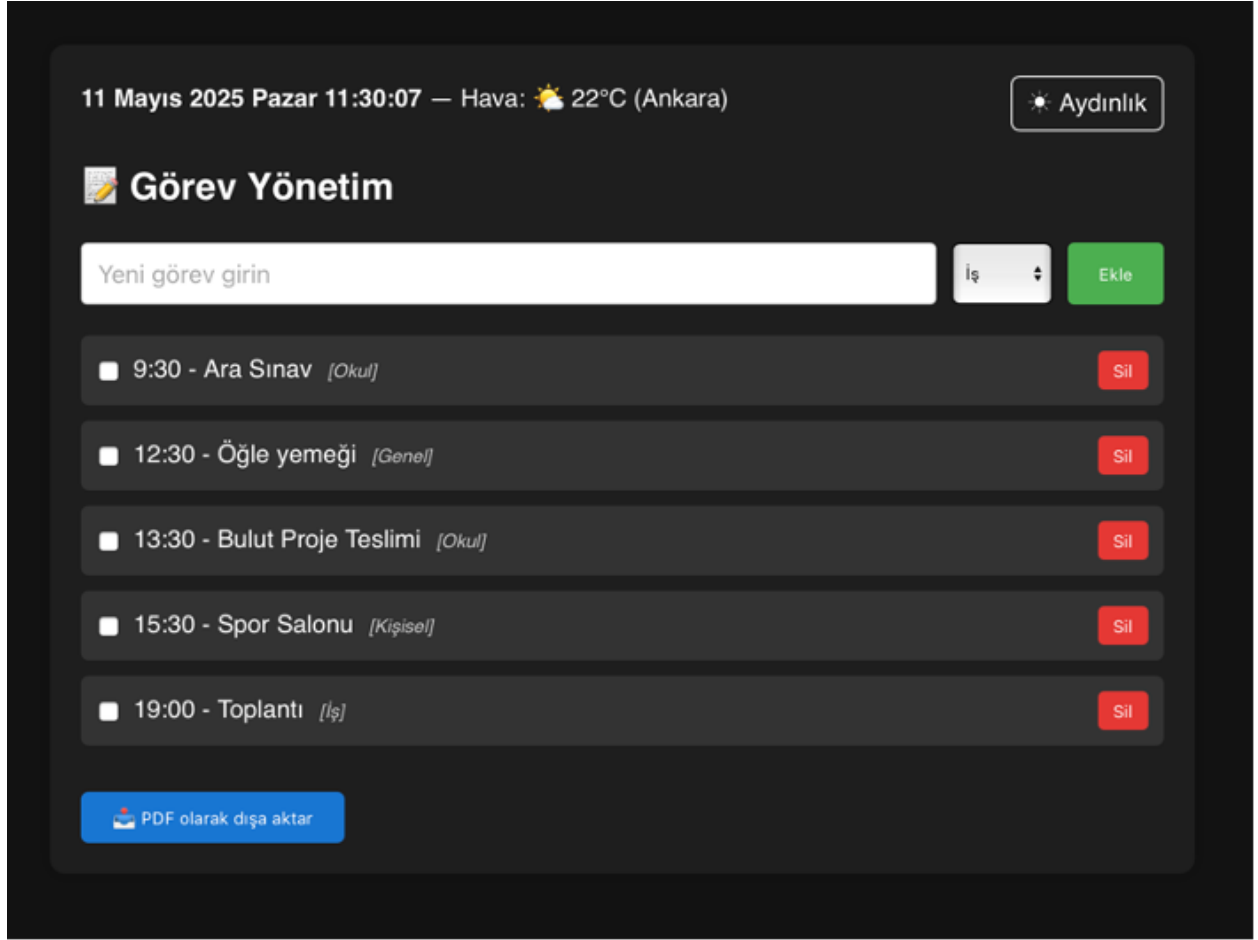


<input type="checkbox"/> 9:30 - Ara Sınav [Okul]	Sil
<input type="checkbox"/> 12:30 - Öğle yemeği [Genel]	Sil
<input type="checkbox"/> 13:30 - Bulut Proje Teslimi [Okul]	Sil
<input type="checkbox"/> 15:30 - Spor Salonu [Kişisel]	Sil
<input type="checkbox"/> 19:00 - Toplantı [İş]	Sil

Resim 1. Görevler ise eklendikten sonra aşağıdaki gibi listelenecektir



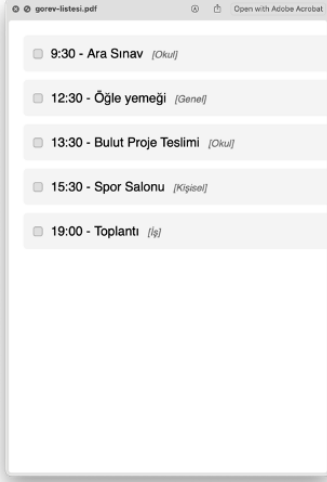
Resim 2. Uygulama “Aydınlık” modunda



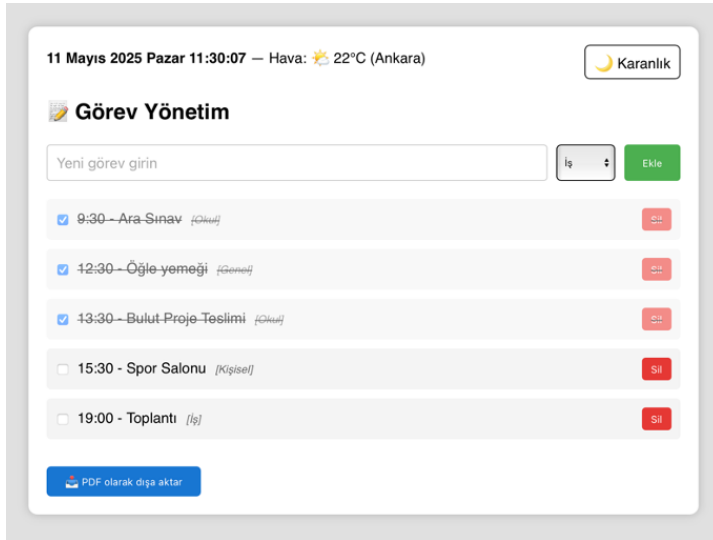
Resim 3. Uygulama “Karanlık” modunda



Resim 4. Uygulamamızın avantajlarından birisi – listeyi PDF olarak indirmektir



Resim 5. Yükleğinde böyle pdf dosyası oluşur.



Resim 6. Görevler tamamlandığında görülmesi.

- **Görevler, bulut teknolojisi sayesinde sayfanın yenilendiğinde kaybolmuyor ve “Sil” butonuna tıkladığımızda siliniyor, sayfanın yenilendiğinde tekrar gözüküyor.**

PROJE 4: AKILLI VERİ ANALİTİĞİ VE MAKİNE ÖĞRENMESİ UYGULAMASI

Problem: Makine Öğrenimi API'si (Application Programming Interface)

Backend: Python

Makine Öğrenmesi Kütüphaneleri: Scikit-learn

Veritabanı: Veri, doğrudan bir CSV dosyasından alınıyor

Bulut Platformu: AWS (Amazon Web Services) - Sagemaker

Veri Seti: Iris

AMAÇ

Bu projenin amacı, makine öğrenmesi ile oluşturulmuş bir modeli kullanarak tahmin işlemlerini gerçekleştirebilen bir sistem kurmaktır. Bu sistemde model, AWS SageMaker üzerinde barındırılmakta ve bu modele veri gönderip sonuç almak için AWS Lambda fonksiyonu kullanılmaktadır. Lambda fonksiyonu, bir API Gateway ile bağlantılıdır. Böylece kullanıcılar, oluşturulan API üzerinden veri göndererek tahmin sonucunu alabilmektedir.

Proje sayesinde, eğitilmiş bir modeli doğrudan internet üzerinden erişilebilecek şekilde yayınlama, bu modeli Lambda ve API Gateway ile bağlayarak otomatik çalışan bir sistem oluşturma hedeflenmiştir. Bu yapı sayesinde hem sunucusuz (serverless) bir ortamda çalışılmış hem de ölçeklenebilir ve düşük maliyetli bir çözüm elde edilmiştir.

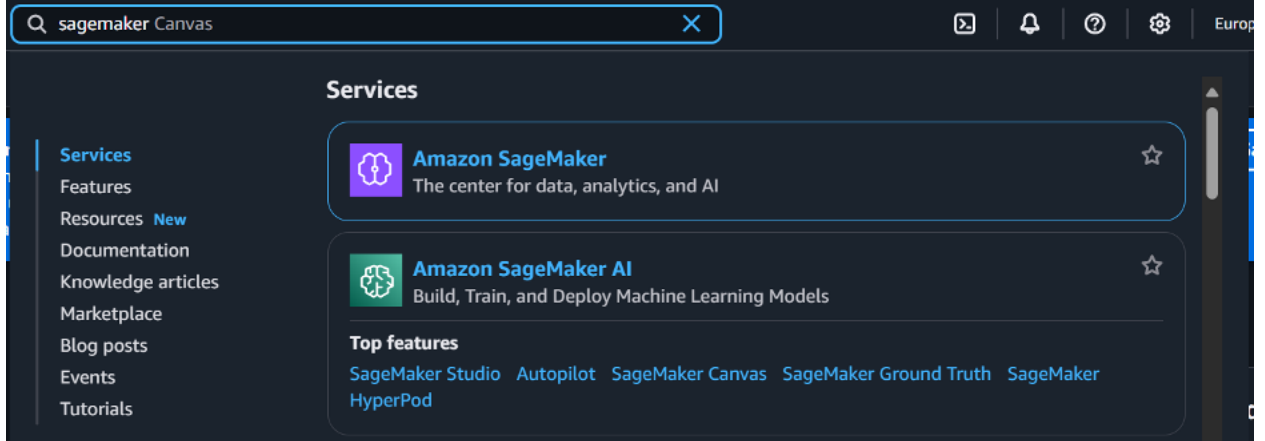
GİRİŞ

Bu projede, gerçek bir veri kümesi üzerinde çalışarak bir makine öğrenmesi modeli geliştirme süreci gerçekleştirilmiştir. Modelin eğitimi, Amazon SageMaker gibi bulut tabanlı bir platform üzerinde gerçekleştirilmiş ve ardından yine bulut ortamında modelin dağıtımı (deployment) sağlanmıştır. Böylece model yalnızca eğitim amacıyla değil, aynı zamanda canlı ortamlarda da kullanılabilir hale getirilmiştir.

Veri ön işleme, analiz ve model eğitimi adımlarının ardından, veriden anlamlı bilgiler çıkarma ve belirli girişlere karşılık tahminlerde bulunma gibi temel makine öğrenmesi hedeflerine ulaşılmıştır. Ayrıca, Lambda ve API Gateway servisleri kullanılarak modelin web üzerinden erişilebilir olması sağlanmış ve uçtan uca bir makine öğrenmesi çözümü tamamlanmıştır.

Bu çalışma sayesinde hem makine öğrenmesi algoritmalarının pratikte nasıl kullanıldığı hem de bulut bilişim teknolojilerinin bu süreçlerdeki önemi konusunda değerli deneyimler kazanılmıştır.

Amazon SageMaker ve Notebook Instance:



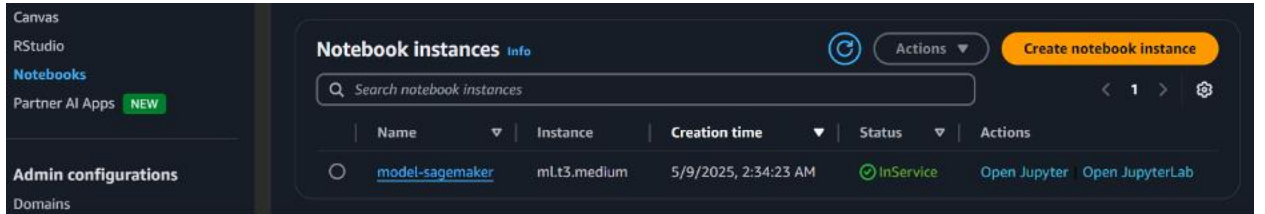
Şekil 1. Amazon SageMaker AI Hizmeti.

AWS SageMaker, makine öğrenmesi projelerini baştan sona yönetmemizi sağlayan bir bulut hizmetidir.

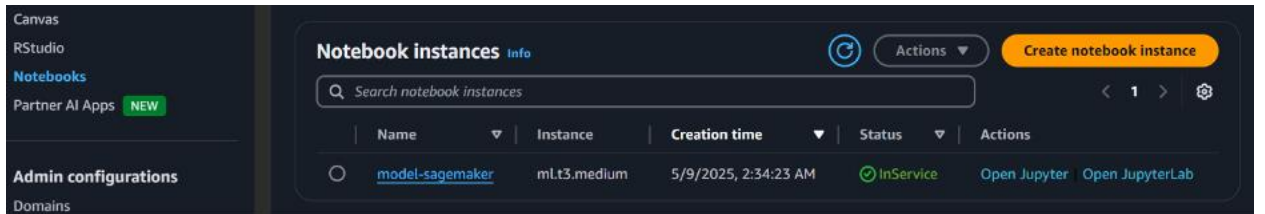
- Veri hazırlama
- Model Eğitme
- Hiperparametre Optimizasyonu
- Model Dağıtım

İşlemlerinin hepsini AWS SageMaker hizmetiyle yapmamız mümkündür.

Proje kapsamında amacımız bir model eğitip tahmin işlemleri gerçekleştirebildiğimiz bir sistem kurmak ve model dağıtımını yapmak olduğu için AWS SageMaker hizmetini kullanmaya karar verdik.



Şekil 2. Notebook Instance Oluşturmak.

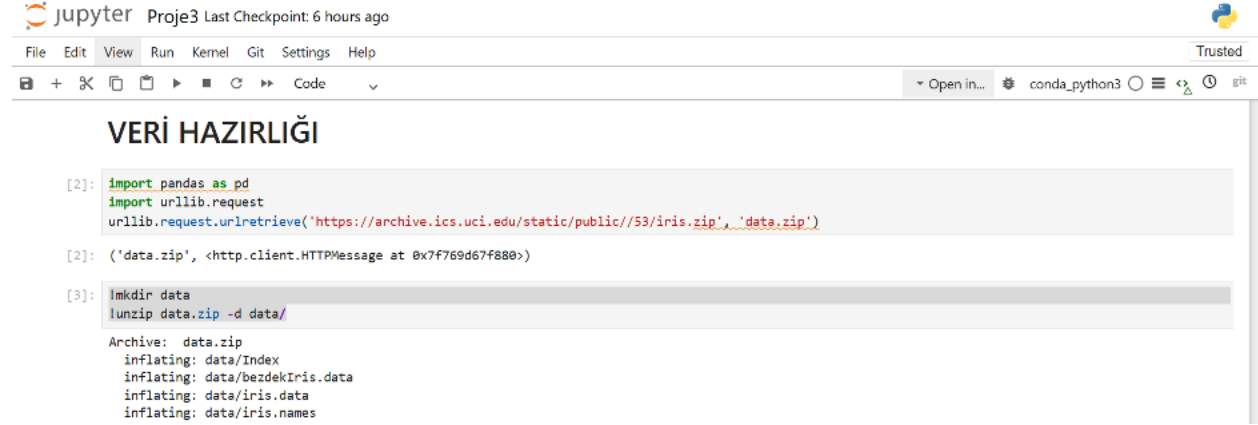


Şekil 3. Jupyter Notebook.

Amazon SageMaker'ın en önemli özelliklerinden birisi içerisinde notebook instance oluşturma imkânımız olmasıdır.

Notebook instance oluşturmayı AWS'nin avantajlarından yararlanarak kullanabildiğimiz bir jupyter notebook oluşturmak olarak ifade edebiliriz, bu sayede bulut üzerinde veri seti yükleme, veri işleme ve model eğitme işlemlerini yapabileceğimiz bir ortama sahip olabiliyoruz.

Veri Hazırlığı:



```
[2]: import pandas as pd
import urllib.request
urllib.request.urlretrieve('https://archive.ics.uci.edu/static/public/53/iris.zip', 'data.zip')

[2]: ('data.zip', <http.client.HTTPMessage at 0x7f769d67f880>)

[3]: mkdir data
unzip data.zip -d data/

Archive: data.zip
  inflating: data/Index
  inflating: data/BezdekIris.data
  inflating: data/iris.data
  inflating: data/iris.names
```

Şekil 4. Veri Setini Çekme İşlemi.

Bu aşamada:

- Urllib kütüphanesini kullanarak yukarıdaki görselde gözüken web sitesine istekte gönderdik ve model eğitmek için kullanacak olduğumuz iris veri setini indirdik.
- İndirdiğimiz veri setini unzip komutunu kullanarak ayıkadık.

```
[57]: # veriyi okumak
df = pd.read_csv('data/iris.data', header=None)
# data = pd.read_csv('data/iris.data', header = None)

# sayısal değerlere dönüştürmek
data[4] = data[4].replace('Iris-setosa', 0)
data[4] = data[4].replace('Iris-virginica', 1)
data[4] = data[4].replace('Iris-versicolor', 2)

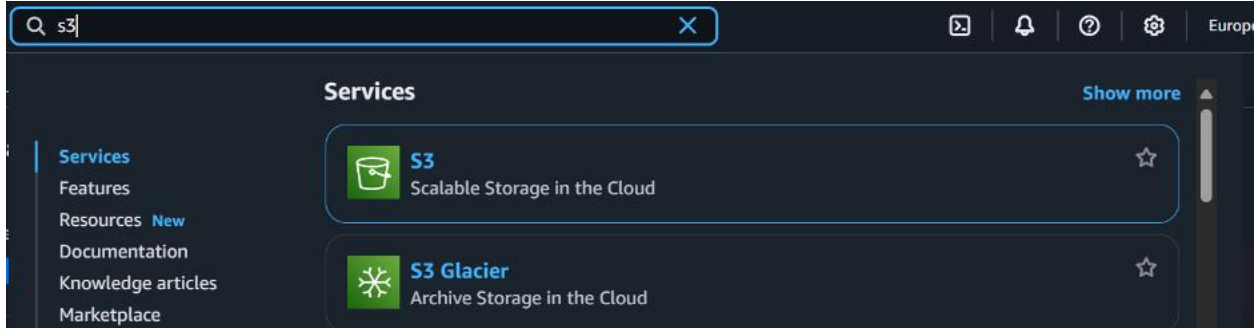
# karıştırmak
data = data.sample(frac = 1).reset_index(drop = True)

# bölmek (eğitim ve doğrulama veri kümeleri)
# %80 eğitim veri seti / %20 doğrulama veri seti
train_data = data[:120]
val_data = data[120:]
```

Şekil 5. Veri Ön işleme Süreci.

- Model eğitmeden önce veri setini inceledik ve gerekli veri ön işleme işlemlerini gerçekleştirerek eğitim ve doğrulama veri setlerini oluşturduk.

S3 ve Bucket Düzenlemeleri(Depolama Düzenlemeleri):



Şekil 6. S3'e Giriş.

Amazon S3(Simple Storage Service), AWS'nin nesne tabanlı depolama hizmetidir. Eğitim ve doğrulama dosyalarımızı SageMaker'ın eğitim işlerinin doğrudan okuyabileceği şekilde buluta yüklemek model eğitim sürecini sorunsuz yürütmemizi sağlayacak. S3'e girerek "sagemaker-kurma-ve-dagitma-modeli" isiminde bir bucket oluşturduk. (Bucket AWS S3'te kullandığımız klasör benzeri bir yapıdır.)

VERİNİN S3'E TAŞINMASI

```
[58]: import boto3
      bucket_name = 'sagemaker-kurma-ve-dagitma-modeli'

      data = pd.concat([data[4], data.drop(columns=[4], axis=1)])
      train_data.to_csv('data.csv', header = False, index = False)
      key = 'data/train/data'
      url = 's3://{}/{}/{}'.format(bucket_name, key)
      boto3.Session().resource('s3').Bucket(bucket_name).Object(key).upload_file('data.csv')

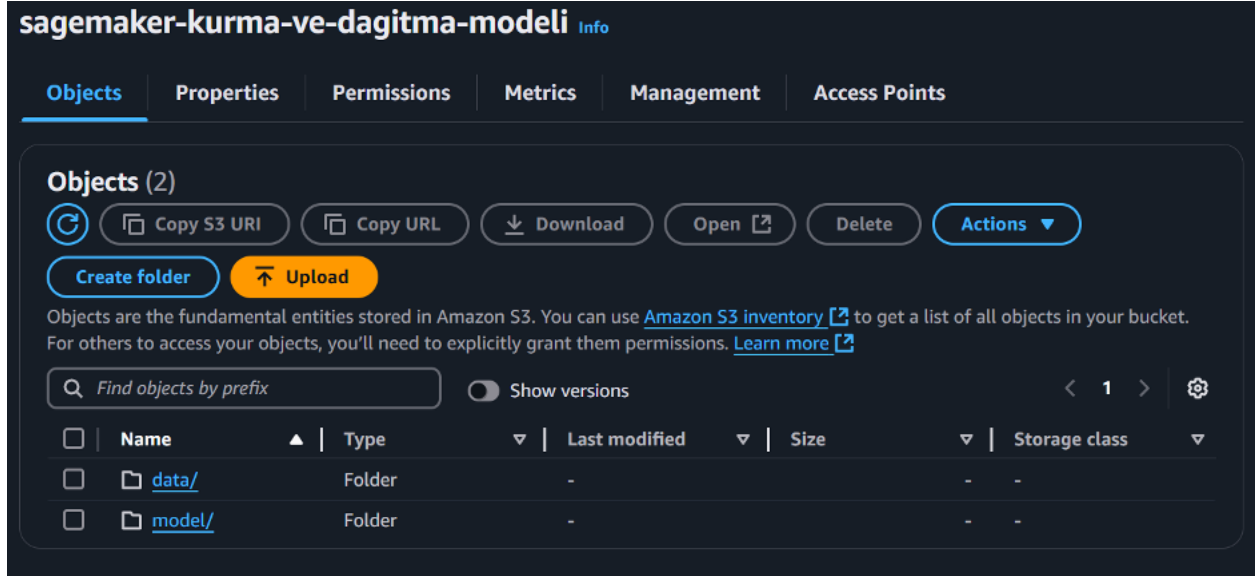
      val_data.to_csv('data.csv', header = False, index = False)
      key = 'data/val/data'
      url = 's3://{}/{}/{}'.format(bucket_name, key)
      boto3.Session().resource('s3').Bucket(bucket_name).Object(key).upload_file('data.csv')
```

[05/11/25 16:38:10] INFO Found credentials from IAM Role: BaseNotebookInstanceEc2InstanceRole credentials.py:1132

INFO Found credentials from IAM Role: BaseNotebookInstanceEc2InstanceRole credentials.py:1132

Şekil 7. AWS S3'e Veri Taşıma.

- Boto3 kütüphanesiyle AWS S3'e bağlandık.
- bucket_name değişkeniyle hedef bucket'ı belirledik, data üzerinde sütun düzenlemesi yaptıktan sonra train_data ve val_data'yı data.csv dosyasına kaydettik.
- "data/train/data" ve "data/val/data" anahtarlarıyla S3'teki bucket'ımıza yükleme işlemini gerçekleştirdik.
- Boto3, IAM rolü üzerinden otomatik kimlik doğrulama sağladı.



Şekil 8. Data Klasörü.

Ekran görüntülerimizi süreci tamamladıktan sonra aldığımız için model klasörü de gözüküyor ancak normalde bu aşamadayken sadece data klasörüne sahiptik.

Docker ve Eğtilecek Modelin Şablonu:

MODEL OLUŞUMU

```
[48]: import sagemaker
from sagemaker.amazon.estimator import get_image_uri
from sagemaker import get_execution_role
from sagemaker import image_uris

key = 'model/xgb_model'
s3_output_location = 's3://{}/{}/{}'.format(bucket_name, key)

container = image_uris.retrieve(
    framework='xgboost',
    region=boto3.Session().region_name,
    version='1.7-1'
)
xgb_model = sagemaker.estimator.Estimator(
    # image_uri=container,
    get_image_uri(boto3.Session().region_name, 'xgboost'),
    role = get_execution_role(),
    train_instance_count = 1,
    train_instance_type = 'ml.m5.xlarge',
    train_volume_size = 5,
    output_path = s3_output_location,
    sagemaker_session = sagemaker.Session()

xgb_model.set_hyperparameters(
    max_depth = 5,
    eta = 0.2,
    gamma = 4,
    min_child_weight = 6,
    # silent = 0,
    verbosity = 1,
    objective = 'multi:softmax',
    num_class = 3,
    num_round = 10)

[05/11/25 16:18:07] INFO Ignoring unnecessary instance type: None. image_uris.py:530

WARNING The method get_image_uri has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details. deprecations.py:34

INFO Ignoring unnecessary instance type: None. image_uris.py:530

WARNING train_instance_count has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details. deprecations.py:34

WARNING train_instance_type has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details. deprecations.py:34

WARNING train_volume_size has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details. deprecations.py:34
```

Şekil 9. Docker ve Estimator.

Sagemaker'ın içinde bulunan Estimator, eğitim sürecini tanımladığımız bir python nesnesidir.

- Container oluşturarak stabil bir çalışma ortamı kurguladık ve model eğitiminde hangi algoritmanın kullanılacağını tanımladık.
- Estimator ile oluşturduğumuz Container ortamında hangi ayarlarla model eğitimi yapacağımızı belirttik.

Bu aşamada Hangi algoritma, hangi veri, kaç tane ve ne tip sunucu kullanılacağı, hiperparametreler gibi tanımlamalarımızı yaptık.

Model Eğitim ve Dağıtım Süreci:

MODEL EĞİTİMİ

```
[49]: import numpy as np
labels = data[4]
unique_labels = sorted(np.unique(labels))
label_mapping = {label: idx for idx, label in enumerate(unique_labels)}
data[4] = np.vectorize(label_mapping.get)(labels)
np.sort(np.unique(data[4]))

[49]: array([0, 1, 2])

[53]: import pandas as pd

df_train = pd.read_csv('data.csv', header=None)
print("Unique labels in training set:", df_train[0].unique())

df_val = pd.read_csv('data.csv', header=None)
print("Unique labels in validation set:", df_val[0].unique())

Unique labels in training set: [6.4 6.3 5.4 6.7 5.5 7.7 5.2 5.8 4.8 4.9 4.6 6.2 5.1 5.7 6.1 6.9 6. 5.6
5. 6.6 4.4]
Unique labels in validation set: [6.4 6.3 5.4 6.7 5.5 7.7 5.2 5.8 4.8 4.9 4.6 6.2 5.1 5.7 6.1 6.9 6. 5.6
5. 6.6 4.4]

[60]: from sagemaker.inputs import TrainingInput

train_data = 's3://{}/{}/{}'.format(bucket_name, 'data/train')
val_data = 's3://{}/{}/{}'.format(bucket_name, 'data/val')

# train_channel = sagemaker.session.s3_input(train_data, content_type='text/csv')
train_channel = TrainingInput(train_data, content_type='text/csv')
# val_channel = sagemaker.session.s3_input(val_data, content_type='text/csv')
val_channel = TrainingInput(val_data, content_type='text/csv')

data_channels = {'train': train_channel, 'validation': val_channel}
xgb_model.fit(inputs=data_channels)

[5]#011train-merror:0.008333#011validation-merror:0.066667
[16:40:41] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 2 extra nodes, 0 pruned nodes, max_depth=1
[16:40:41] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 2 extra nodes, 4 pruned nodes, max_depth=1
[16:40:41] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 4 extra nodes, 2 pruned nodes, max_depth=2
[6]#011train-merror:0.008333#011validation-merror:0.066667
[16:40:41] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 2 extra nodes, 0 pruned nodes, max_depth=1
[16:40:41] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 2 extra nodes, 4 pruned nodes, max_depth=1
[16:40:41] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 4 extra nodes, 2 pruned nodes, max_depth=2
[7]#011train-merror:0.008333#011validation-merror:0.066667
[16:40:41] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 2 extra nodes, 0 pruned nodes, max_depth=1
[16:40:41] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 2 extra nodes, 4 pruned nodes, max_depth=1
[16:40:41] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 4 extra nodes, 2 pruned nodes, max_depth=2
[8]#011train-merror:0.016667#011validation-merror:0.066667
[16:40:41] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 2 extra nodes, 4 pruned nodes, max_depth=1
[16:40:41] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 4 extra nodes, 2 pruned nodes, max_depth=2
[8]#011train-merror:0.016667#011validation-merror:0.066667
[16:40:41] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 2 extra nodes, 0 pruned nodes, max_depth=1
[16:40:41] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 2 extra nodes, 4 pruned nodes, max_depth=1
[16:40:41] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 4 extra nodes, 4 pruned nodes, max_depth=2
[9]#011train-merror:0.008333#011validation-merror:0.066667

2025-05-11 16:41:00 Completed - Training job completed
Training seconds: 89
Billable seconds: 89

[51]: df = pd.read_csv('data.csv')
print(df.head())
print(df.iloc[:, 0].unique())

6.4 3.2 5.3 2.3 1
0 6.3 3.3 4.7 1.6 2
1 5.4 3.0 4.5 1.5 2
2 6.7 3.3 5.7 2.1 1
3 5.5 2.6 4.4 1.2 2
4 7.7 2.8 6.7 2.0 1
[6.3 5.4 6.7 5.5 7.7 6.4 5.2 5.8 4.8 4.9 4.6 6.2 5.1 5.7 6.1 6.9 6. 5.6
5. 6.6 4.4]
```

Şekil 10. Model Eğitimi.

Birkaç veri düzenleme işlemi yaptık ve modelimizi eğittik

```
MODELİN DAĞITIMI 11

[61]: xgb_predictor = xgb_model.deploy(initial_instance_count = 1,
    instance_type = 'ml.m5.xlarge')

[05/11/25 16:45:47] INFO Creating model with name: xgboost-2025-05-11-16-45-47-848 session.py:4094

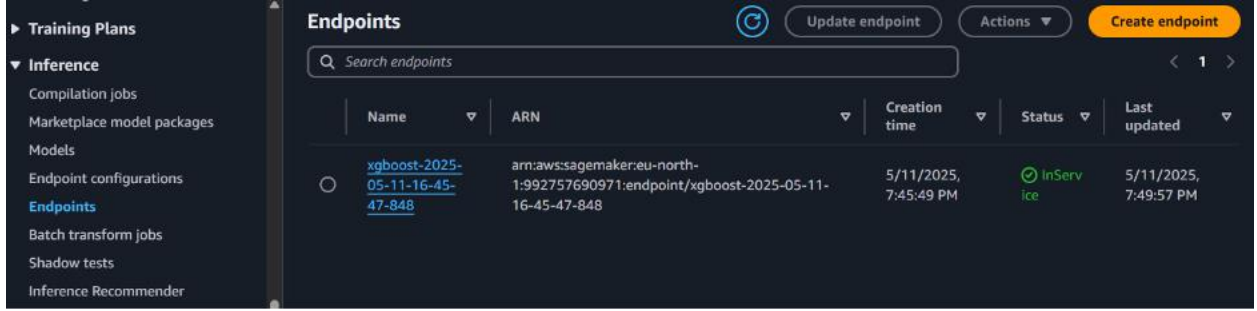
[05/11/25 16:45:48] INFO Creating endpoint-config with name xgboost-2025-05-11-16-45-47-848 session.py:6019

INFO Creating endpoint with name xgboost-2025-05-11-16-45-47-848 session.py:4841

-----]
```

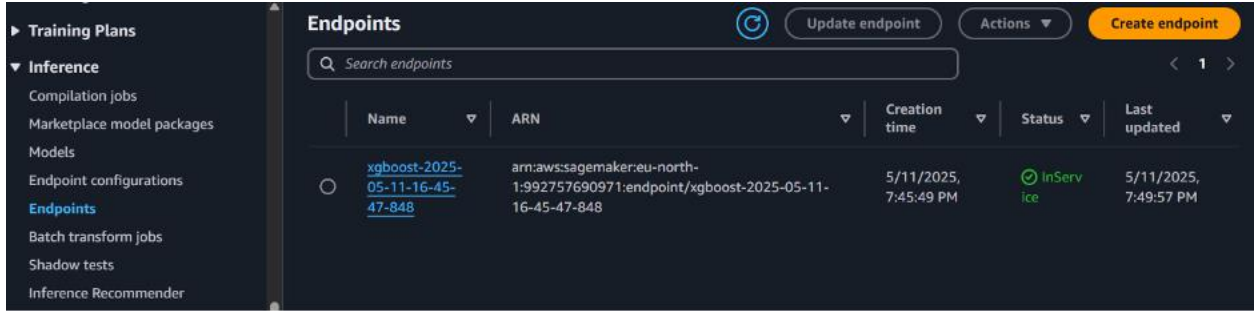
Şekil 11. Model Dağıtımı.

Gerçek zamanlı bir tahmin sunucusu kurduk (Endpoint oluşturduk).
Bu sayede dışarıdan istek atılıp, tahmin sonucunun elde edilebileceği sistemi oluşturmuş olduk.



Name	ARN	Creation time	Status	Last updated
xgboost-2025-05-11-16-45-47-848	arn:aws:sagemaker:eu-north-1:992757690971:endpoint/xgboost-2025-05-11-16-45-47-848	5/11/2025, 7:45:49 PM	InService	5/11/2025, 7:49:57 PM

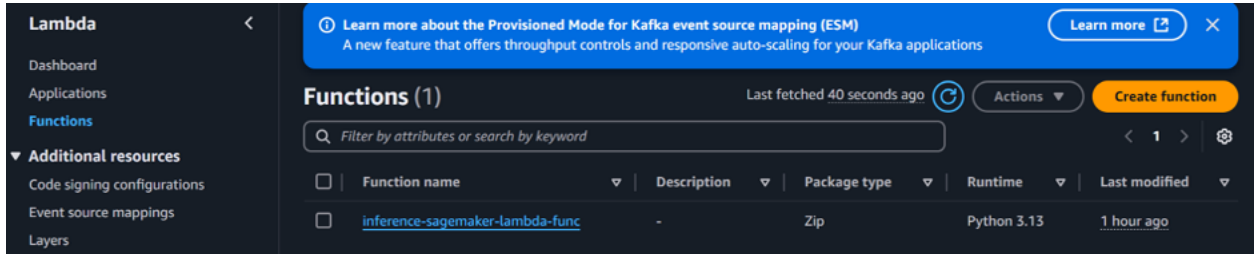
Şekil 12. Oluşan Dosya ve Klasörler.



Name	ARN	Creation time	Status	Last updated
xgboost-2025-05-11-16-45-47-848	arn:aws:sagemaker:eu-north-1:992757690971:endpoint/xgboost-2025-05-11-16-45-47-848	5/11/2025, 7:45:49 PM	InService	5/11/2025, 7:49:57 PM

Şekil 13. İki Aşama Önce Oluşturduğumuz Endpoint.

API Gateway Düzenlemeleri ve Lambda Fonksiyonu:



Function name	Description	Package type	Runtime	Last modified
inference-sagemaker-lambda-func	-	Zip	Python 3.13	1 hour ago

Şekil 14. Oluşturulan Lambda Fonksiyonu.

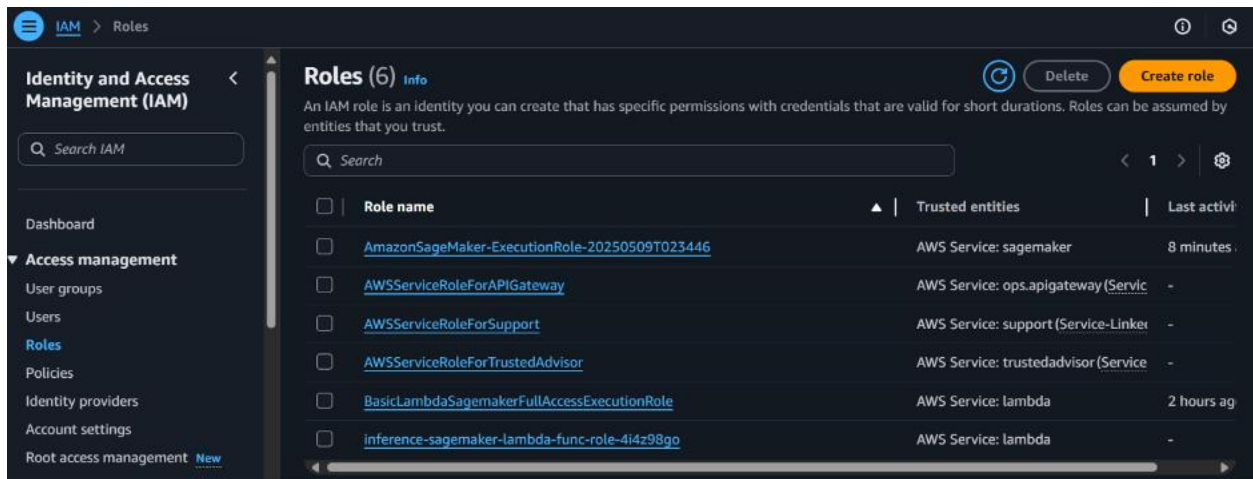
```
lambda_function.py X
lambda_function.py
1 import boto3
2 import json
3 import ast
4
5 def lambda_handler(event, context):
6     # TODO implement
7
8     runtime_client = boto3.client('runtime.sagemaker')
9     endpoint_name = 'xgboost-2025-05-11-16-45-47-848'
10    sample = '{},{},{},{}'.format(ast.literal_eval(event['body'])['x1'],
11                                  ast.literal_eval(event['body'])['x2'],
12                                  ast.literal_eval(event['body'])['x3'],
13                                  ast.literal_eval(event['body'])['x4'])
14
15    response = runtime_client.invoke_endpoint(EndpointName = endpoint_name,
16                                              ContentType = 'text/csv',
17                                              Body = sample)
18    result = int(float(response['Body'].read().decode('ascii')))
19
20    return {
21        'statusCode': 200,
22        'body': json.dumps({'prediction':result})
23    }
24
```

Şekil 15. Lambda Fonksiyonunun Kodu.

Lambda Fonksiyonu yazdığımız kod sayesinde gelen HTTP isteklerinin nasıl yönetileceğini belirler.

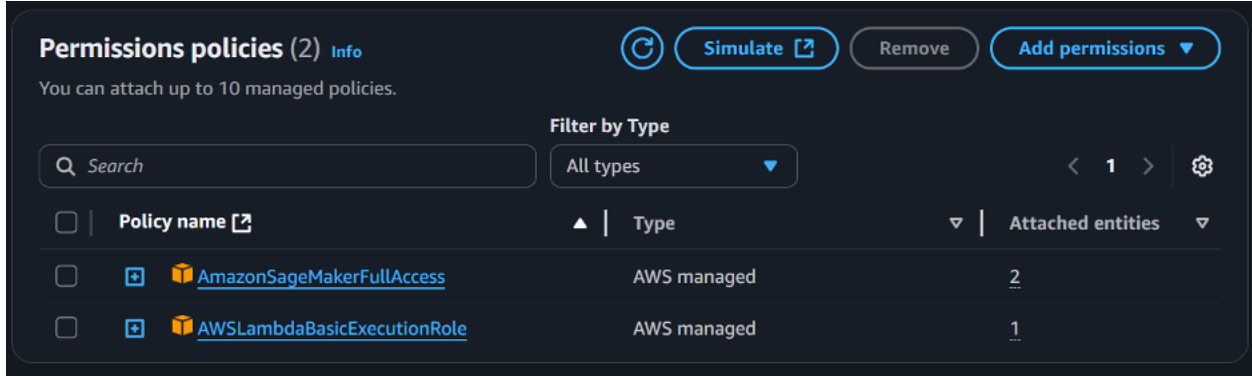
Helen isteği alır eğittiğimiz modelin olduğu endpointe yönlendirir, tahmin sonucunu alır uygun formata çevirir ve istemciye tahmin sonucunu geri gönderir.

Ayrıca sunucuyu sadece istek atıldığı zamanlarda aktifleştirerek maliyetleri ve enerji tüketimini azaltır.



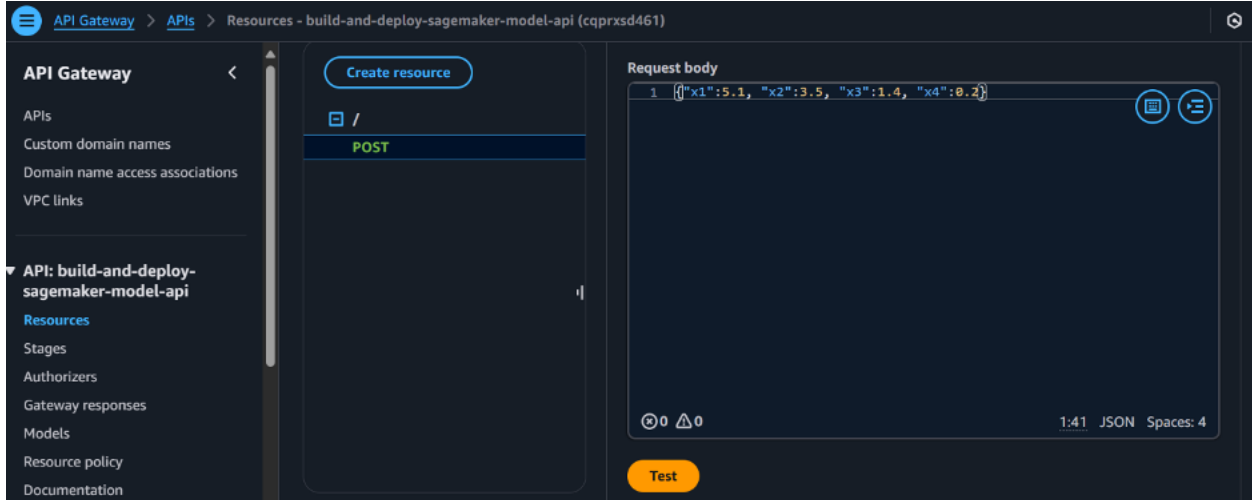
Şekil 16. IAM Rol Oluşturma.

Lambda kodu çalıştığında SageMaker endpoint'ine istek atabilsin, S3'ten veri okuyup CloudWatch'a log yazabilsin diye bu tam erişim yetkilerine sahip rolü oluşturduk.



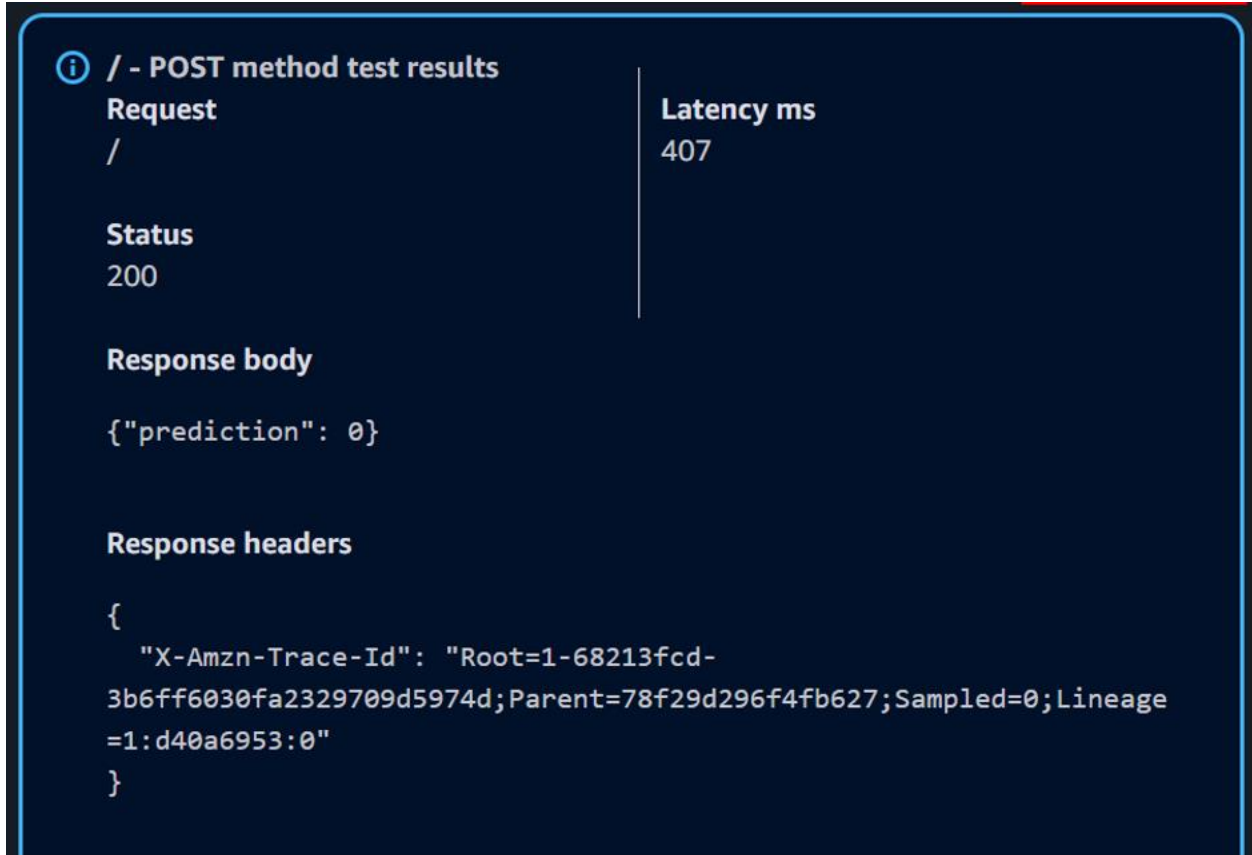
Şekil 17. Rol için Seçilen Politikalar.

Uygulama:

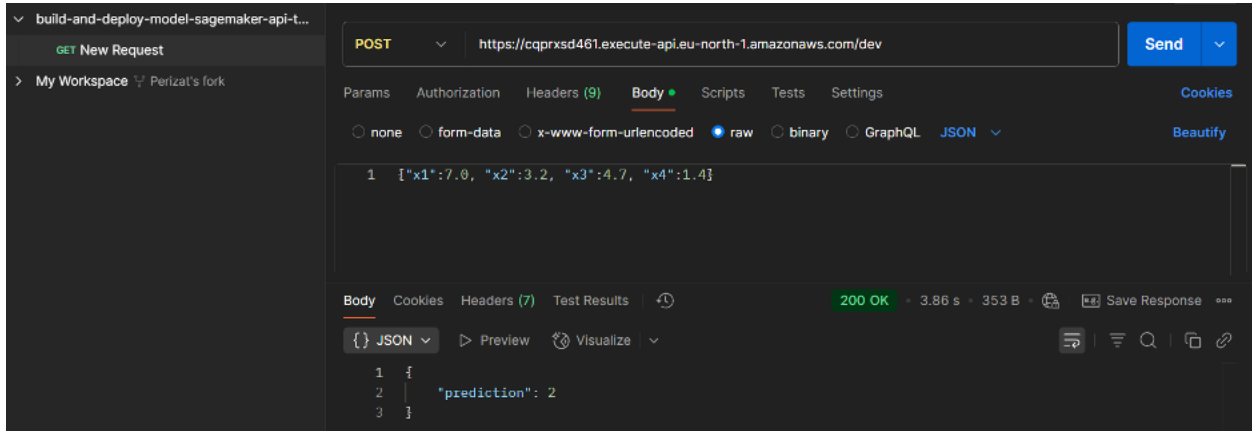


Şekil 18. API Gateway POST Metodu.

- Oluşturduğumuz endpoint'e istek attık.



Şekil 19. Tahmin Çıktısı.



Şekil 20. All-In-One API Platformu (Postman) Üzerinde Deneme.

Sonuç olarak, bulut ortamında dağıtık çalışabilen ve kendisine istek gönderildiğinde tahmin sonuçları döndürebilen bir model geliştirdik.

KAYNAKLAR

1. Veri Seti: <https://archive.ics.uci.edu/dataset/53/iris>

2. XGBoost Hiperparametreleri:

https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost_hyperparameters.html