ANKARA ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BLM3522

BULUT BİLİŞİM VE UYGULAMALARI

RAPORU

Proje 7

Ömer Asaf DEMİR

20290240

GitHub: AsafDemir/BulutBilisimVeUygulamalariBut

https://github.com/AsafDemir/BulutBilisimVeUygulamalariBut

1. Proje Genel Tanıtım

Kullanılan Yapı

- Gerçek zamanlı otopark doluluk takibi
- AWS tabanlı sunucusuz mimari
- MQTT üzerinden IoT cihaz simülasyonu
- Web tabanlı veri görselleştirme

Bu proje, farklı konumlardaki otopark alanlarından gelen doluluk verilerini IoT cihazlar üzerinden toplayarak AWS servisleri aracılığıyla işler ve kullanıcıya sunar. Sistem, MQTT protokolü ile çalışan sensör simülatörleri, AWS IoT Core, Lambda, DynamoDB ve API Gateway gibi bileşenlerle serverless ve ölçeklenebilir bir mimariye sahiptir.

2. Sistem Mimarisi ve Çalışma Mantığı

Temel Bileşenler

- mqtt_simulator.py (Veri üretimi ve iletimi)
- AWS IoT Core (MQTT mesaj karşılayıcı)
- Lambda – SaveParkingData (Veri işleme ve kaydetme)
- DynamoDB – ParkingData tablosu
- Lambda – GetParkingData (Veri alma)
- API Gateway (/v1 endpoint)
- Web frontend (Gerçek zamanlı dashboard)

Simülatör, her 30 saniyede bir 10 farklı otopark cihazı adına veri üretip AWS IoT Core'daki parking/data topic'ine gönderir. IoT Rule bu veriyi yakalayarak SaveParkingData fonksiyonunu tetikler. Lambda fonksiyonu gelen JSON verisini kontrol eder ve DynamoDB'ye kaydeder. GetParkingData fonksiyonu en güncel verileri filtreleyip API Gateway üzerinden frontend'e iletir.

3. Kodlama ve Dosya Yapısı

Öne Çıkan Dosyalar

- mqtt_simulator.py
- lambda_function.py (2 mod: write & read)

- config.py (AWS ayarları ve cihaz bilgileri)
- aws_config klasörü (iot_policy.json, dynamodb_schema.json)
- frontend/index.html, script.js, style.css

Simülatör scripti cihaz başına ayrı thread açarak paralel veri iletimi yapar. Lambda kodları hem yazma hem okuma işlevi içerir. Config dosyaları merkezi ayar yönetimini sağlar. Frontend kısmı responsive ve modern tasarımla gerçek zamanlı görselleştirme sunar.

4. Güvenlik ve Erişim Kontrolleri

Uygulanan Önlemler

- MQTT SSL/TLS veri şifreleme
- IoT Policy: Minimum yetkili bağlantı izinleri
- IAM: Lambda ve DynamoDB için sınırlı roller
- API Gateway CORS ve API Key kontrolü

Tüm MQTT bağlantıları SSL ile güvence altına alınmıştır. IoT policy dosyasında sadece belirli client ID'lerin publish etmesine izin verilmiştir. IAM rolleri sadece gereken kaynaklara erişime sahiptir. API Gateway CORS desteğiyle frontend erişimlerine izin verirken, opsiyonel olarak API anahtarı da kullanılabilir.

5. Performans ve Optimizasyon

Sağlanan İyileştirmeler

- MQTT bağlantı havuzu (connection pooling)
- Lambda batch işleme desteği
- DynamoDB'de composite key kullanımı
- Local caching ile frontend performansı
- CloudWatch log analizi

Sistem, ölçeklenebilirlik ve gecikme azaltımı hedefiyle yapılandırılmıştır. Lambda fonksiyonları kısa süreli işlem süresiyle optimize edilmiştir. DynamoDB'de timestamp + device_id yapısı ile zaman bazlı sorgular hızlandırılmıştır. Frontend tarafında local cache ve async API çağrıları ile akıcı bir kullanıcı deneyimi sağlanmıştır.

6. Test ve Kalite Güvencesi

Uygulanan Testler

- MQTT bağlantı ve veri bütünlüğü testleri
- Lambda → DynamoDB yazma senaryoları
- API Gateway → JSON veri alma kontrolü
- UI güncelleme ve anlık yenileme doğrulaması

Sistem uçtan uca test senaryolarıyla doğrulanmıştır. Cihazdan veritabanına, kullanıcı arayüzüne kadar olan tüm veri akışı test edilmiştir. Gerçek zamanlı veri tazeleme, bağlantı kopması ve yeniden bağlanma durumları göz önüne alınarak sistem hata toleranslı şekilde çalıştırılmıştır.

7. Proje Sonuçları

Elde Edilen Başarılar

- 10 cihazdan 30 saniyede bir veri aktarımı
- Gerçek zamanlı dashboard güncellemeleri
- Serverless yapı sayesinde bakım gereksinimi az
- Düşük gecikmeli API erişimi ve görselleştirme

Proje başarıyla tamamlanmış, gerçek dünya koşullarında kullanılabilecek bir altyapı kurulmuştur. Sunucusuz ve otomatik ölçeklenebilir mimari sayesinde sistemin çalışabilirliği yüksek, maliyeti ise düşük tutulmuştur.