

# Serverless Microservices with AWS

---

Fatma EMÜL | Oğuz OK | Berat BAYRAM | Ahmet SAY | Mustafa Emre BAŞAR



- Bu çalışmamızda mikroservis mimarisinin avantajlarını keşfetmeyi amaçlarken, bu avantajları serverless anlayışı ile birleştirerek daha da bağımsız ve güvenilir bir yapı elde edebilmenin yöntemlerini aradık.
- Mikroservis mimarisi ile birlikte servislerin birbirinden bağımsızlığını sağlarken, serverless ile de iş mantığı ile altyapı arasındaki ilişkiyi de daha soyut hale getirebiliyoruz. Böylelikle her katmanda daha modüler bir anlayışı benimseyebilmiş oluyoruz.

# SERVERLESS

- Serverless altyapı hizmetleri ile ilgilenmeden uygulamaları geliştirme ve deploy etmek için kullanılan bir yaklaşımdır. Altyapısal hizmetler, bizim için bir Cloud Provider tarafından yürütülür. Bu sayede geliştiriciler daha çok iş mantıklarına odaklanabilir ve donanım, güvenlik, depolama, ölçeklendirme, bakım gibi çeşitli altyapı uğraşları ile ilgilenmek mecburiyetinde olmazlar.
- Güncel olarak serverless mimarileri genelde FaaS (Function as a Service) anlayışı ile çalışır. Bu anlayışta uygulama ayrık fonksiyonlar şeklinde geliştirilir ve her fonksiyon bir event tarafından tetiklenir.



Fonksiyon bulut sağlayıcıya  
gönderilir



Fonksiyon tetikleyicileri  
ayarlanır



Bulut sağlayıcısı, tetiklendiğinde  
fonksiyonu çalıştırır

# AVANTAJLARI



## Maliyet

Servislerin ücretlendirmesinde, kullandığın kadar öde mantığı vardır. Kullanılmayan durumlarda ücret ödenmez ve boşa kaynak kullanımı olmaz.



## Ölçeklendirme

Kullanılan servislerin ölçeklendirilmesi bulut sağlayıcısı tarafından yönetilir ve geliştiricilerin bir eforu gerekmez.



## Üretkenlik

Altyapı unsurlarından olabildiğince soyutlanmış geliştiriciler, daha çok iş mantıklarına odaklanır ve daha üretken hale gelirler. Uygulamanın geliştirilmesi ve deploy edilmesi arasında süre kısalmır.



## Lokalizasyon

Uluslararası bir hizmet sunulduğunda, servis sağlayıcısının dünyanın uzak noktalarında sahip olduğu sunucular sayesinde, isteklerin en yakın sunucu tarafından yanıtlanması sağlanır ve hız artmış olur.

# DEZAVANTAJLARI



## Kontrol Kaybı

Altyapı hizmetlerini bir dış servise bırakmak doğal olarak bunlar üzerindeki kontrolümüzü kısıtlar. Her ne kadar konfigüre edebiliyor olsak da en ince noktasına kadar ayarlama yapabilmemiz mümkün değildir.



## Güvenlik

Sahip olduğumuz veriler ve iş mantıklarımız servis sağlayıcısı tarafından ele alındığı için, iyi konfigüre edilmez ise güvenlik açıklarına sebep olabilir.



## Performans Etkileri

Uzun süre kullanılmayan fonksiyonlar ilk kez tetiklendiğinde **cold starts** durumu oluşur. Bu durum fonksiyonun çalışmaya başlamasını geciktirir ve performansı olumsuz etkiler.



## Test Etme

Geliştiriciler unit testleri sorunsuz bir şekilde geliştirebiliyor olsa da integration testler için aynı durum söz konusu değildir. Farklı bileşenler arası etkileşim gerektiren testlerin koşulması zor bir hale gelmektedir.



## Sağlayıcı Kısıtlaması

Bir bulut sağlayıcısının sunduğu servisler kendi aralarında kolayca entegre olabilirken, farklı sağlayıcı servisleri arasında bu durum söz konusu olmayabilir. Bu durum bizi belirli bir bulut sağlayıcısının servislerini kullanmaya bağlı kılar.

# AWS



Amazon Web Services, Amazon tarafından sağlanan servislerin bulunduğu bir bulut platformudur. AWS, herhangi bir tipteki sunucu ile ilgilenmeden uygulama geliştirme imkanı sunar.



Sunucusuz teknolojiler, modülerliği arttırmak ve maliyetleri optimize etmek için otomatik ölçeklendirme, yerleşik yüksek erişilebilirlik ve kullanıma göre ödeme modeli içerir.



Temel olarak işleme, uygulama entegrasyonu, veri saklama başlıklarında hizmetler sunar.

# MICROSERVICES

- Uygulamalar başlangıcından itibaren gittikçe büyür ve gün geçtikçe kontrol edilmesi zor bir hal alır. Aslında birbirinden farklı işleri yapmakta olan hizmetlerin tek bir yapıda bulunuyor olması aralarında gereksiz bir bağımlılık oluşmasına sebep olur.
- Bu bağımlılıklar gittikçe daha kompleks bir hale gelir ve baş edilemez bir hale bürünebilir. Bu sorunu ortadan kaldırmak için iş tanımları ve kapsamaları belirli servisler oluşturarak, her bir servisin birbirlerinden bağımsız hatta habersiz çalışabilmesi sağlanabilir.



# AVANTAJLARI

---

## Ölçeklendirme

Servisler birbirlerinden bağımsız uygulamalar olduklarından, her biri tek başına ölçeklendirilebilir. Servislerin ihtiyaçları diğer servislerle bir etkisi olmadan giderilebilir.

---

## Veri İzolasyonu

Mikroservis mimarisine göre her bir servis kendi veri tabanına sahiptir. Bu servis ile alakalı verilerin gereksiz yere erişilmesini engeller. Aynı zamanda veri tabanlarının ayarlanması da servis özelinde olduğu için bir yan etki yaratması söz konusu olmaz.

---

## Hata İzolasyonu

Servisler birbirlerinden bağımsız olduklarından, birinde oluşan hatanın diğer servislerle veya bütün sisteme etkisi minimize edilir ve alakalı yerde kontrolü sağlanmış olur.

---

## Teknoloji Özgürlüğü

Birbirinden bağımsız sistemler, birbirleri ile haberleşme konusunda entegre olabildikleri sürece farklı teknolojileri kullanmakta özgürdürler. Bu da servis ile alakalı en uygun teknolojilerin seçilmesi ve daha iyi performansın alınabilmesini sağlar.

---

## Anlaşılabilirlik

Bütün sistem daha küçük parçalara bölüdüğü için her bir servis kendi içerisinde daha kolay kavranabilir hale gelir. Ayrıca her bir servis için atanan küçük takımlar bütün sistem yerine sadece bu servise özgü mantıkları daha doğru ve kolayca kavrayabilir.

---

## Bakım ve Geliştirme

Her servis birbirinden bağımsız geliştirildiği ve deploy edildiği için, yeni özellikler eklenmek istendiğinde veya hata çözümlerinde bütün sistemin deploy edilmesi gerekmez. Aynı şekilde hatalı bir sürümde rollback işlemini de sadece belirli serviste yapma imkanı sunar.

# DEZAVANTAJLARI

---

## Komplekslik

Her bir servis tek başına monolitik mimariden daha basit bir yapıya sahip olsa da servislerin birbiriyle iletişimi de düşünülduğünde bütün sistem daha kompleks bir yapıya sahiptir.

---

## Geliştirme ve Test

Servislerin geliştirilmesi ve test edilmesi, diğer servislere bağımlılığı bulunduğu durumlarda, monolitik yaklaşıma kıyasla daha zor bir hal almaktadır. Senaryoya dahil olan her bir servisin geliştirme esnasında uygun durumda bulunması gerekir.

---

## Yönetim

Farklı teknolojilerin kullanılması proje genelinde bir standart oluşmasını zorlaştırabilir. Her bir servis kendi standartlarını oluşturur ise servisler arası tutarsızlıklar söz konusu olabilir.

---

## Monitörleme

Her bir servis çalışma süresince kendi loglarını üretir ve bu loglar servislere özgü bir şekilde monitör edilir. Bu durum servis sayısı çoğaldıkça log takibini zorlaştırabilir.

---

## Ağ Problemleri

Servisler arası iletişim ve zincirleme servis çağrılarının çoğaldıkça, gecikmeler performansa olumsuz etki edebilir.

---

## Veri Bütünlüğü

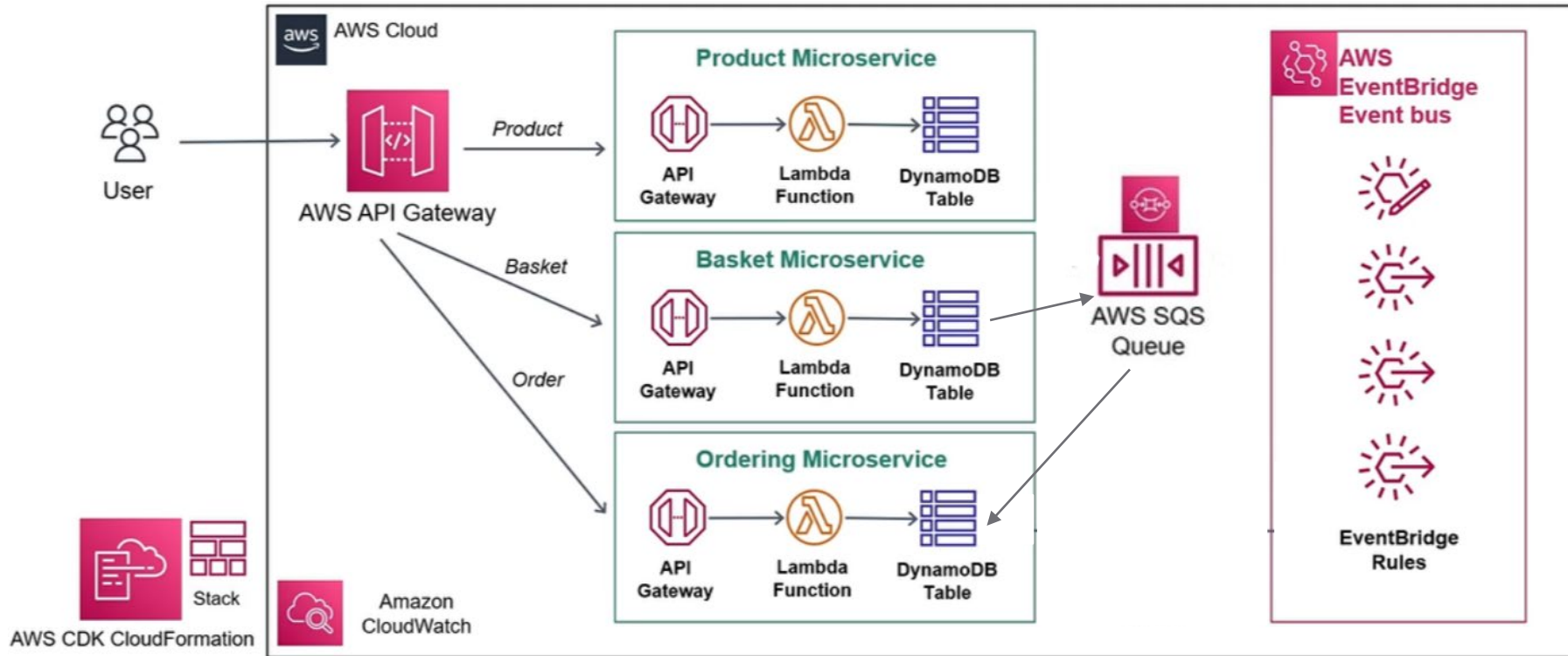
Her bir servis kendi veri tabanından sorumlu olduğu için servisler arası kullanılan verilerin tutarlılığını korumak bir zorluk yaratır. Bu durumda genellikle *eventual consistency* prensibi benimsenir.

---

## Versiyonlama

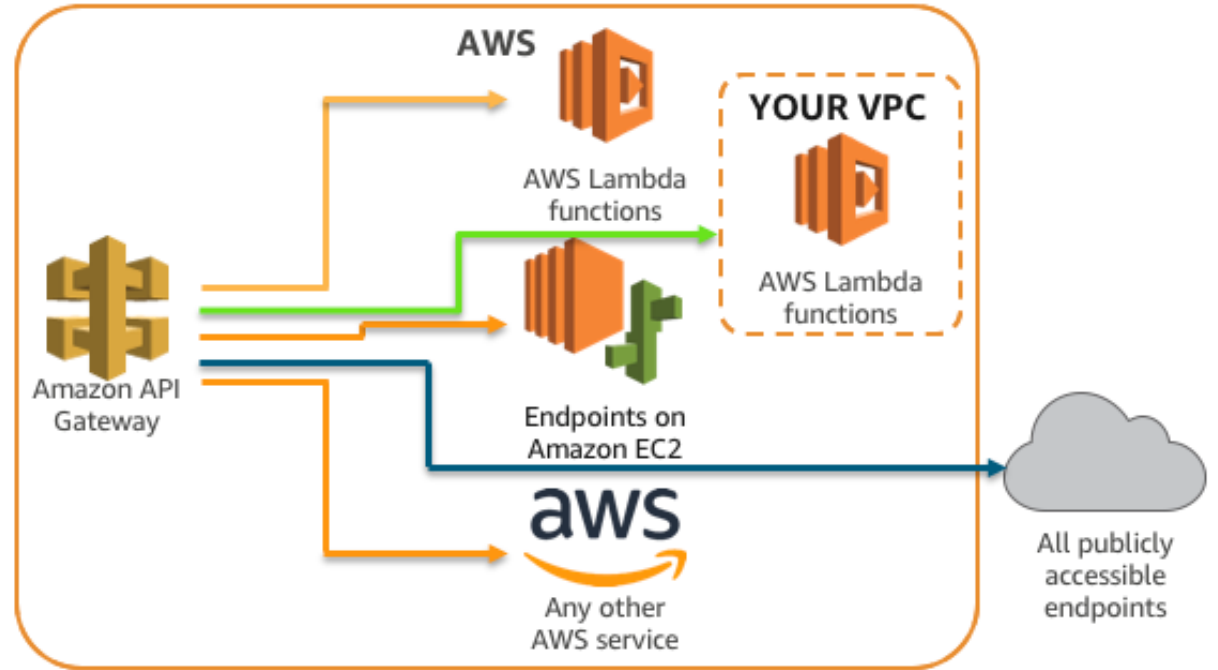
Her bir servis birbirinden bağımsız versiyonlanabilir olduğundan, diğer servisler ile entegrasyonuna yan etkileri olmamasına dikkat edilmelidir.

# UYGULAMA



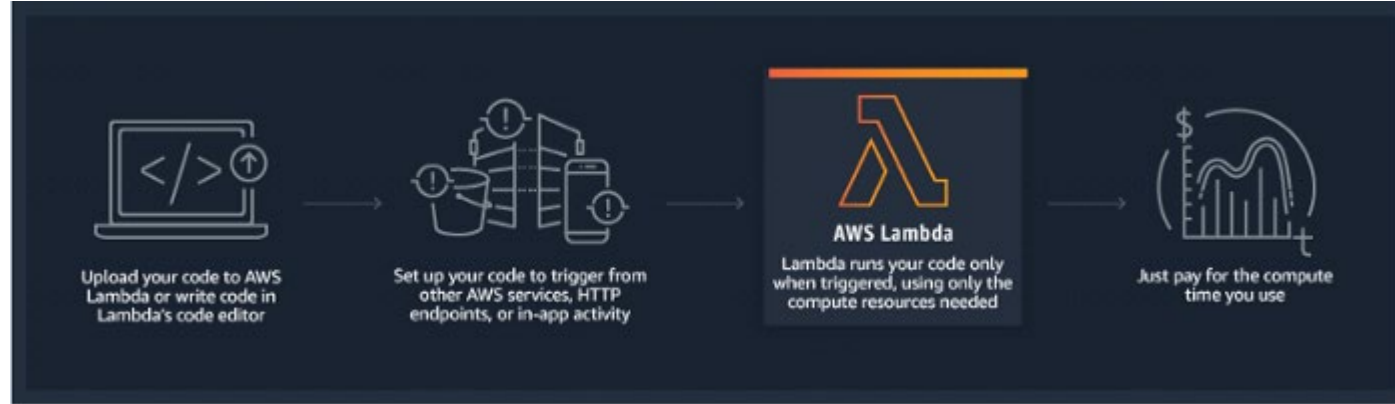
# API Gateway

- Amazon API Gateway kullanıcılardan gelen istekleri almak ve bu istekleri çeşitli konfigürasyonlar, kurallara göre ilgili servislere yönlendirmekle görevlidir
- Uygulamada RESTful API geliştirilmesi ve senkron isteklerin, cevapların paylaşılmasında AWS API Gateway kullanılmakta. Gateway mikroservislerde bulunan Lambda fonksiyonlarına sunucusuz bir API Proxy sağlar, bu sayede CRUD isteklerini ilgili mikroservislere iletir.

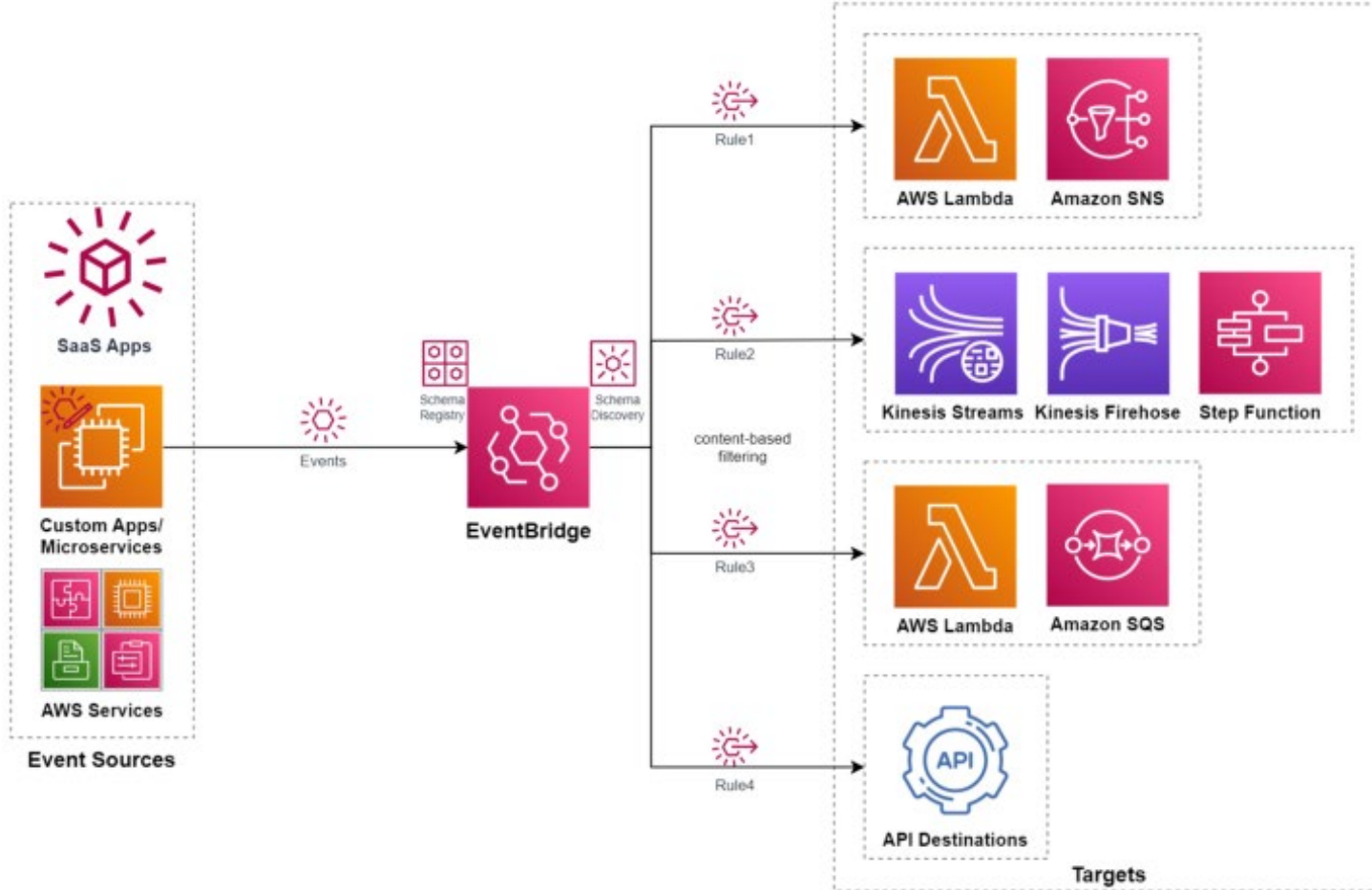


# Lambda

- AWS Lambda, sunucu sağlama ve yönetme ihtiyacı olmadan yazılan kodu çalıştırmaya yarayan sunucusuz event-driven programming ve serverless computing platformudur
- **Kod çalışmadığı durumlarda herhangi bir ücretlendirme olmaz.**
- Otomatik olarak ölçeklendirme
- Sistem bakımı
- Monitoring
- Loglama



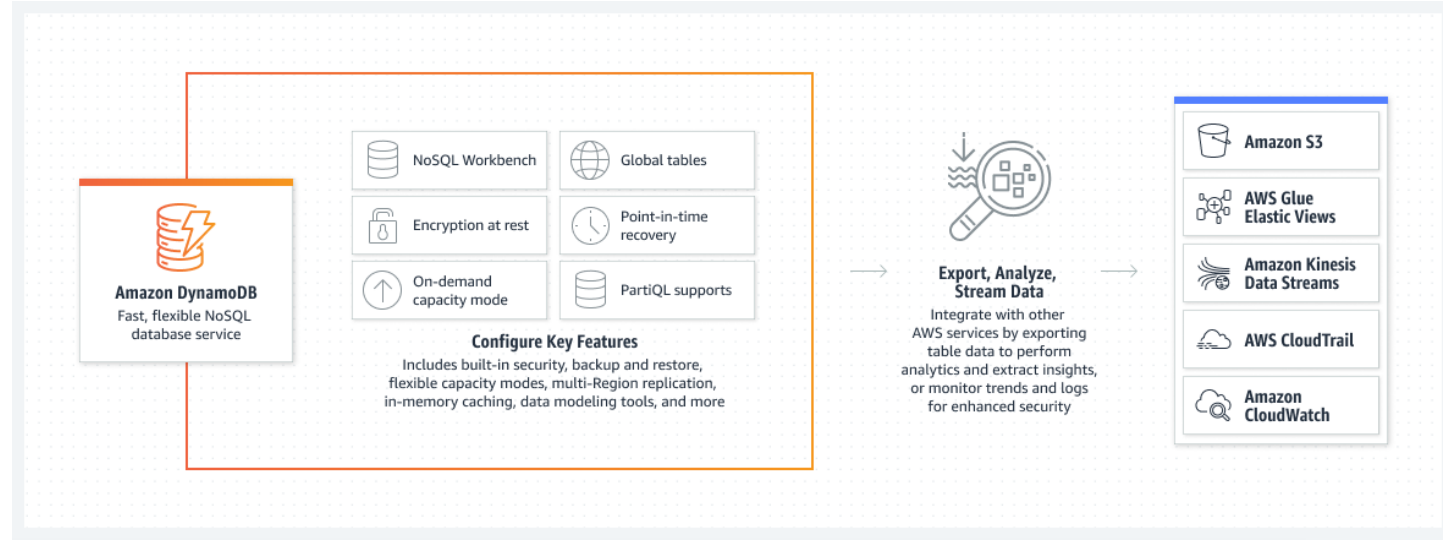
# EventBridge



- Mikroservis mimarisinde her servisin birbirinden olabildiğince bağımsız olabilmesi önemli noktalardan biridir.
- Bu sebeple de; EventBridge servisler arası **event tabanlı iletişim** ile asenkron bir haberleşme sağlar
- AWS EventBridge kullanarak ve ilgili servisimizin **birbirinden haberi dahi olmadan** haberleşebiliyor olmasını amaçladık.

# DynamoDB

- DynamoDB bir NoSQL veritabanıdır. DynamoDB, tam olarak yönetilebilen bir veritabanıdır, sorunsuz ölçeklenebilmesiyle hızlı ve öngörülebilir bir performans sağlar.



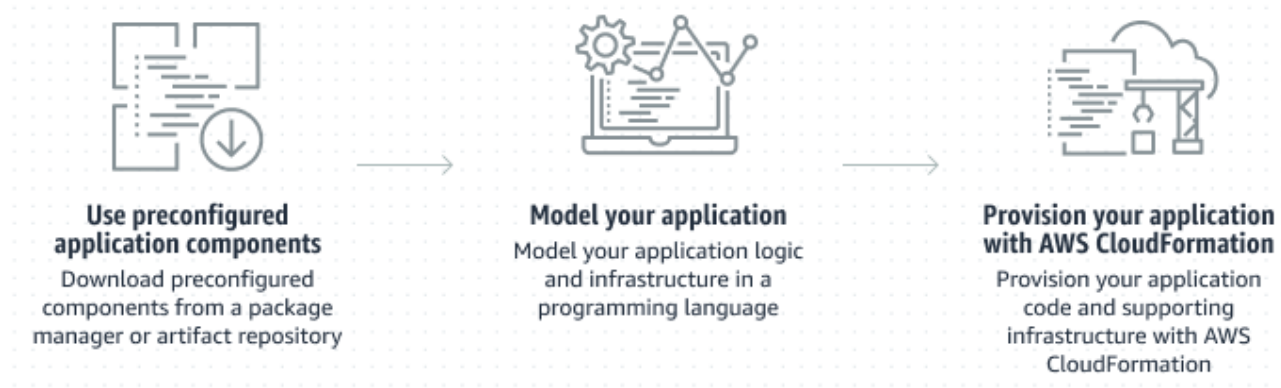
- Uygulamamızda mikroservis mimarisinin önerdiği üzere her servisin kendi verilerini saklamasını sağlamak için AWS DynamoDB kullanıldı. DynamoDB her boyuttaki servis için iyi bir performans verebiliyor olması dolayısıyla mikroservis mimarisi için iyi bir tercih.

# SQS

- Başka herhangi bir servise ihtiyaç olmadan mikroservisler arasındaki iletişimin asenkron şekilde, mesaj kuyrukları yardımıyla kurulmasını sağlar. Örnek uygulamada mesaj kuyrukları yardımıyla servisler arası iletişim (Basket – Ordering Mikroservisleri) kurmak için AWS SQS kullanılmakta.
- Amazon Event Bridge, Basket mikroservisinden gelen checkout isteğini AWS SQS'deki mesaj kuyruğu ile paylaşır ve paylaşılan bu mesaj Order mikroservisi tarafından asenkron olarak işlenmek üzere kuyruktan alınır. Bu sayede daha esnek ve dayanıklı bir yapı oluşmasını sağlar.



```
Parameters:
  KeyName:
    Description: The EC2 Key Pair to allow SSH access to the instance
    Type: 'AWS::EC2::KeyPair::KeyName'
Resources:
  Ec2Instance:
    Type: 'AWS::EC2::Instance'
    Properties:
      SecurityGroups:
        - !Ref InstanceSecurityGroup
        - MyExistingSecurityGroup
      KeyName: !Ref KeyName
      ImageId: ami-7a11e213
  InstanceSecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: Enable SSH access via port 22
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: 0.0.0.0/0
```

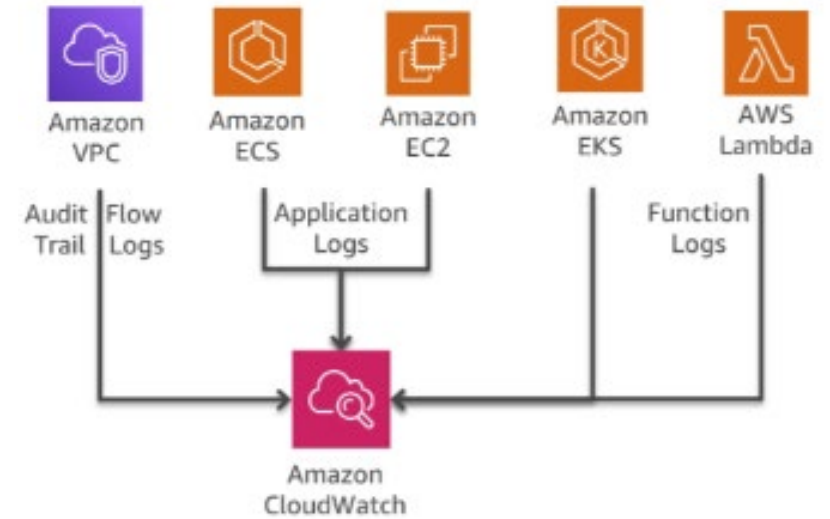


# Cloud Formation

- CloudFormation, Infrastructure as Code tekniğini kullanarak AWS kaynaklarını konfigürasyon dosyalarında tutarak yönetmeyi amaçlamaktadır. Örnekte belirtilen uygulama mimarisinin tamamı CloudFormation ile oluşturulabilir.

# CloudWatch

- Mikroserviste izlenmesi gereken birçok servisi vardır. Servislerdeki kaynak kullanımı, uygulama performansı ve operasyonel işlem hakkında sistem çapında görünürlük elde etmek için CloudWatch kullanılabilir. Geliştiriciler her servis için ayrı kriterler yazarak istenilen verilerin izlenmesi sağlayabilir.
- Loglar sorunları gidermek ve sorunları belirlemek için kritik öneme sahiptir. Amazon EC2 bulut sunucularında çalışan uygulamalar için, günlük dosyalarını CloudWatch Logs'a göndermek için bir arka plan programı mevcuttur. Lambda ve Amazon ECS, logları CloudWatch Logs'a gönderir.



# SONUÇ

- Mikroservis ile serverless mimarisi kullanıldığında maliyet daha düşüktür ve verimliliği yüksektir.
- Uygulamanın ölçeğini genişletirken yazılması gereken ilk kod miktarının azaltılmasına yardımcı olur.
- Serverless mikroservisler karmaşık ve hızlı-gelişen uygulamalar için uygundur. Servislerin kolayca yönetilip ölçeklenebilmelerini sağlar.
- Mikroservisler ve serverless mimariler aynı yapısal ilkeleri izler. Her ikisi de ölçeklenebilirliğe ve esnekliğe öncelik veriyor. Amazon API Gateway, entegrasyon ek yükünü azaltmak için bir dizi popüler programlama dilinde programatik olarak oluşturulmuş istemci SDK'ları sağlıyor.



Dinlediğiniz  
İçin  
Teşekkürler