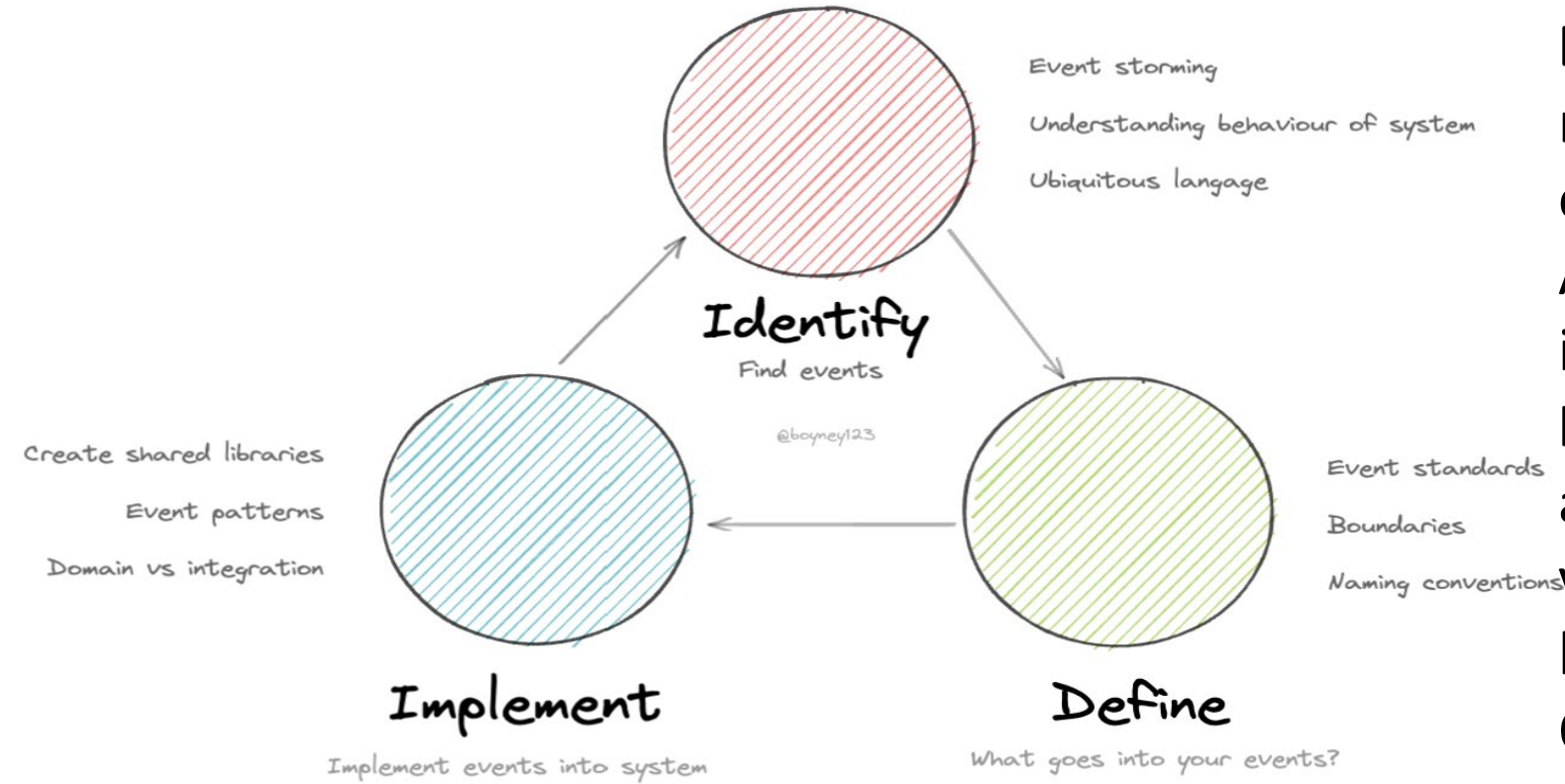


Bilgisayar Mühendisliđi
Yüksek Lisans Programı
2026 Bahar

Yazılım Mimarileri (devam)
27.04.2026

Event First Thinking*



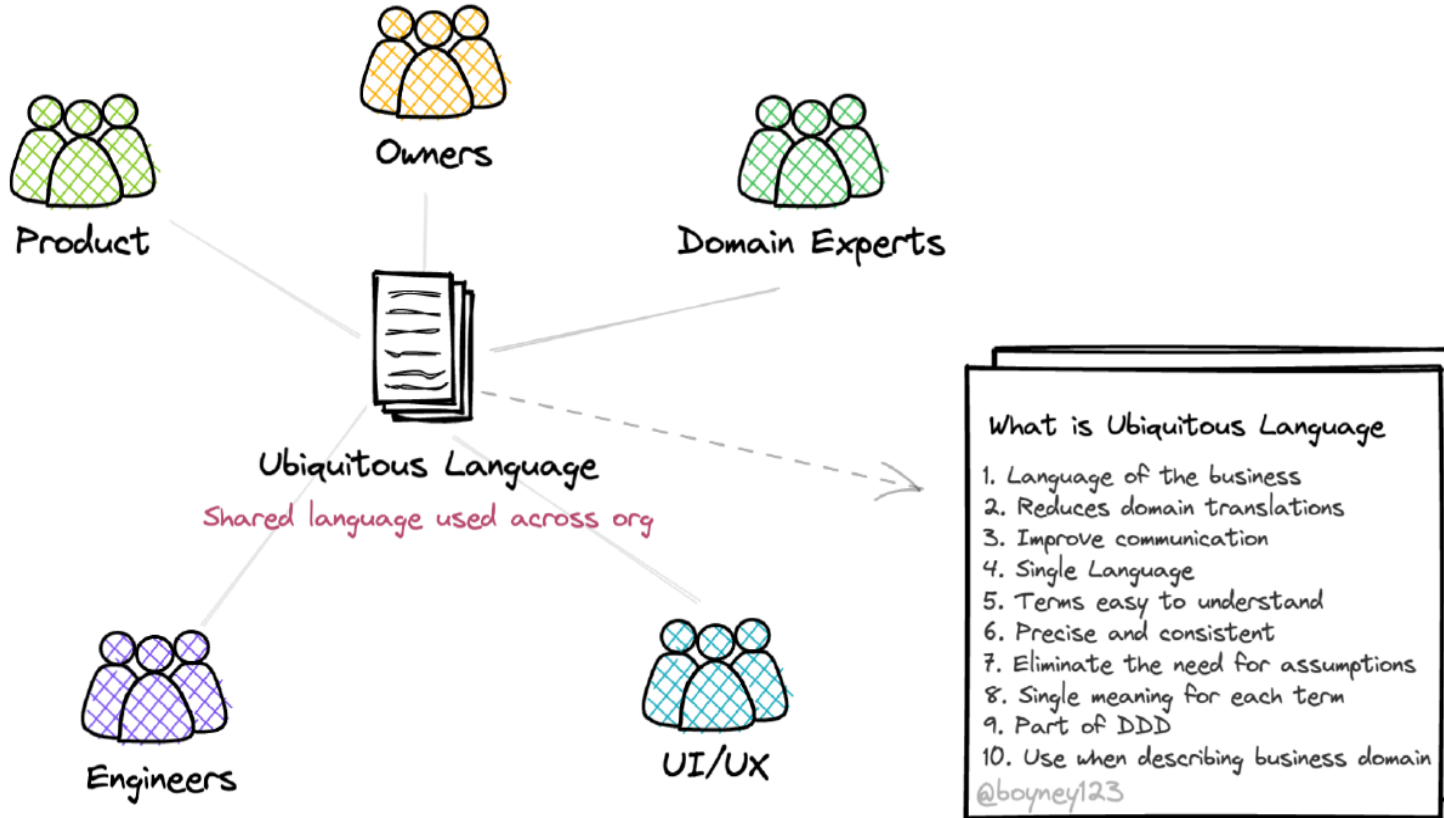
***EDA Visuals:** Small bite sized visuals about event-driven architectures David Boyne v1.2.02

Identify: Sistemin sınırlı bağlamı (context) ve etki alanları (domain) belirlenir; iletişim ortak bir dil (**ubiquitous lang.**) ile kolaylaştırılır.

Define: Olaylar üreticiler ve müşteriler için dokümanlaştırılarak olay kataloğu hazırlanır (sözleşme). Adlandırma kuralları ve kullanımlar ilgili standartlarla hazırlanır.

Implement: Standartlar, adlandırma kuralları veya meta veriler için paylaşılan kütüphanelerin kullanılır. Olayların internal- external ayrımında anlaşılması. Olaylar aynı bağlam içinde mi kullanılıyor, sınırlı bağlamlar arasında iletişim kurmak için mi kullanılıyor?

Ubiquitous Language- Güçlü Dilin Katkısı



Olay Tabanlı Mimari (EDA)

Sistemler arasında **asen kron iletişimi** sağlayarak bileşenlerin bağımsız ve esnek bir şekilde çalıştıran yazılım tasarım modelidir.

Olaylar: Sistem içinde veya çevresinde meydana gelen önemli olaylar veya durum değişiklikleridir.

Event Producer (Olay Üreticileri): Olayların iletimini başlatan bileşenlerdir.

Event Consumers (Olay Tüketicileri): Olayları alan ve işleyen bileşenlerdir.

- ❖ Eylemleri *olay tüketicileri* gerçekleştirir;
- ❖ Fazla sayıda olayı tetikleyerek olaylara tepki veren uygulama, servis, iş akışları veya diğer herhangi bir varlık (entity) olabilir.

Event Mediators – (Olay Aracıları): Olayların üreticilerden tüketicilere iletilmesini sağlayan altyapıdır.

Event Brokers /Channels (Olay Aracıları/Kanalları): Üreticiler ve tüketiciler arasında olayların iletimini kolaylaştıran altyapı.

- ❖ Olay aracılığı, mesaj kuyrukları (örneğin, **RabbitMQ**, ActiveMQ), yayınla-abone ol sistemleri (örneğin, Kafka, Redis) veya akış işleme platformları (örneğin, Apache Kafka, Apache Flink) olabilir.

RabbitMQ

- ❑ Dağıtık sistemlerde ve mikroservislerde mesajlaşma sistemlerini bileşenler arasında kesintisiz iletişimi (seemless communication) sağlar.
- ❑ Açık kaynaklı mesaj aracı (broker) RabbitMQ, uygulamalar arasında mesaj gönderip almak için AMQP'yi (**Advanced Message Queuing Protocol**) kullanır.
- ❑ Erlang* dilinde geliştirilen bu sistem güçlü bir mimariye sahiptir.
- ❑ Yüksek eşzamanlılığı (high concurrency) etkin şekilde yönetir.
- ❑ Esnekliği (flexibility), güvenilirliği (reliability) ve kullanım kolaylığı (easy use) temel üstünlükleridir.

*Fonksiyonel ve genel amaçlı dildir. Lisp, PLEX,[2] Prolog, Smalltalk dillerinden etkilenerek 1986 yılında Ericsonn tarafından yazılmıştır. Dağıtık, fault tolerant ve gerçek zamanlıdır. Dart, Scala, Go, Elixir bu dilden etkilenmiştir.

RabbitMQ'nun Uygulamalardaki Kullanım Alanları

eBay: Arkayüzdeki görevleri yönetmek ve daha iyi ölçeklenebilirlik için servisleri birbirinden ayırır. **Mikroservis iletişimi** rolü üstlenir.

Instagram: Yüklemeden sonra görüntüleri ve videoları asenkron olarak işler. **Görev kuyruklarını** oluşturma rolü üstlenir.

Mozilla: Uygulama günlüklerini gerçek zamanlı olarak toplar ve yönlendirir. **Gerçek zamanlı veri toplama** ve raporlama rolü üstlenir.

BBC: Platformlar arasında gerçek zamanlı içerik güncellemeleri yapar.

SoundCloud: Mesaj kuyruklarıyla mikroservisler arasındaki iletişimi kolaylaştırır.

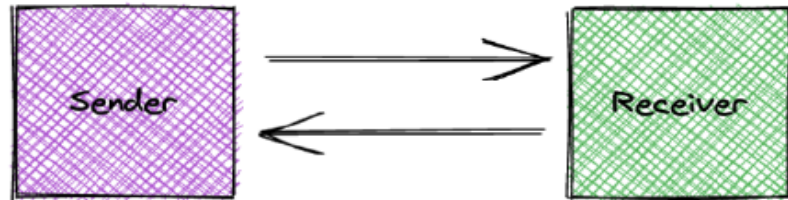
Slack: Gerçek zamanlı bildirimleri ve mesajları kuyruğa alır ve iletir.

Sync vs Async Communication

Request and wait for a response
Fire and forget with messages/events

@boyney123

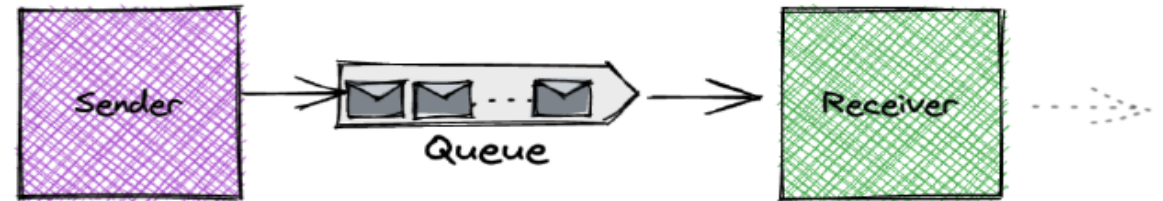
Synchronous



Request-response model

Send a request and wait for some response
Example: API requests

Asynchronous



Fire and forget model

Send message/event and forget
Example: Event Driven Architecture

Message Queue : Mesajlar bir kuyruğa yerleştirilir; Consumer mesajı işler.

Mesajların tüketildiği onaylanır ve silinir. Mesajlar Consumers arasında bölünür, bu da sistemle events arasında iletişimi zorlaştırır.

Örneği Amazon SQS'dir. Mesajlar kuyrukta yer alır (publish) ve ardından listening, processing ve removing gerçekleştirilir.

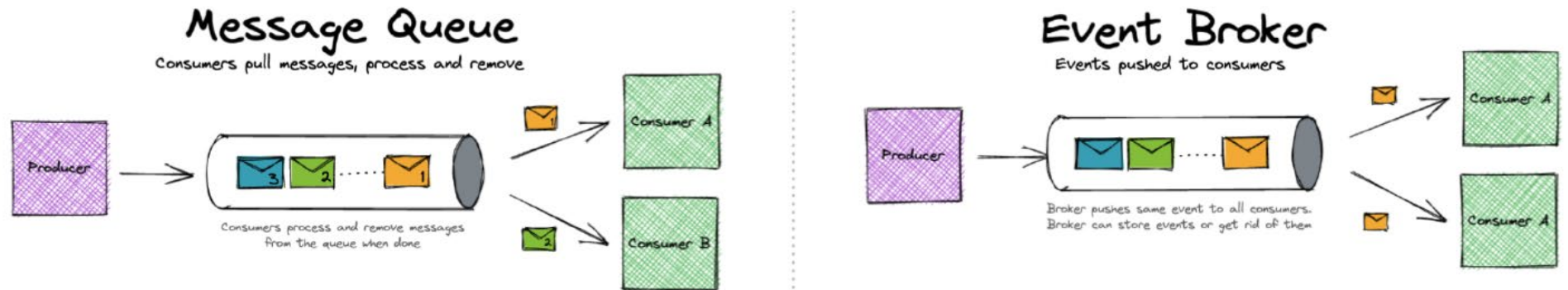
Event Broker: Olay araçları bir push sistemidir.

Bu sistemle, olaylar Consumers a doğru *downstream* olarak **push** gerçekleştirir. Amazon EventBridge örneğidir.

Message queues vs Event Brokers

Queue your message up to be processed...
Broadcast events throughout your architecture...

@boynof123



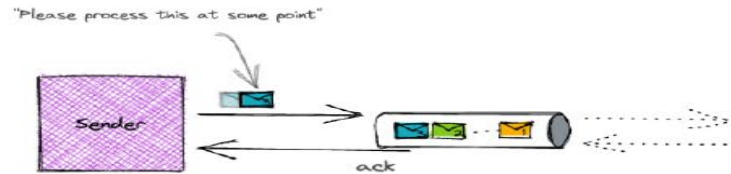
Push-Pull Model

Olayların yayınlanması (publish) push, olayların dinlenmesi (subscribe) pull işlemidir.

- ❑ Push modeli: Belirli bir olay meydana geldiğinde, bu olayı diğer bileşenlere otomatik olarak iletilir.
 - ❖ Örneğin bir kullanıcı bir formu doldurduğunda, bu olay otomatik olarak arka planda bir servise iletilir. Servis, bu olayı alır ve gerekli işlemleri başlatır.
- ❑ Pull Modeli: Bileşenler, belirli aralıklarla olayları kontrol eder; gerektiğinde bu olayları alır.
 - ❖ Örneği Bir uygulama, belirli bir zaman diliminde sunucudan yeni verileri kontrol eder. Eğer yeni bir veri varsa, bu veriyi alır ve işler.
- ❑ Push modelinde gereksiz veri iletimini azaltılabilir , pull modeli her zaman veri kontrolü yaptığı için kaynak kullanımını artabilir.

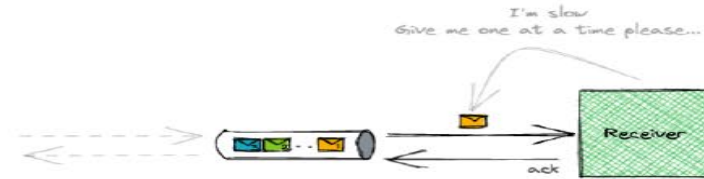
Mesajlaşma Şablonları : Point to Point Pattern:

Bir Producer mesajı yalnızca bir kuyruğa gönderir ve bu mesaj yalnızca bir Consumer tarafından tüketilir.



Asynchronous processing

Sender passes message to channel
Sender can move on with its life



Control messages downstream

Channel can provide buffer for messages
Downstream consumer controls the rate of consumption

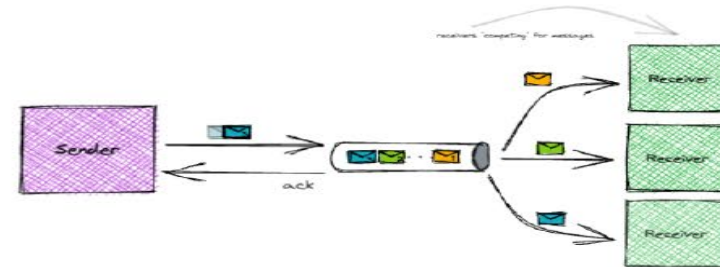
Point-to-point messaging

Sending messages across channels for asynchronous processing



Communicate with messages

Messages put onto queue by sender
Receiver consumes messages from queue



Concurrent processing

Messaging pattern is highly scalable
Many consumers can process queue

Mesajlaşma Şablonları: Publish / Subscribe Pattern

□Yayınla/Abone Ol Şablonu: Producer mesajı bir değişim noktasına gönderir.

❖Bu nokta mesajı birden fazla kuyruğa yönlendirir.

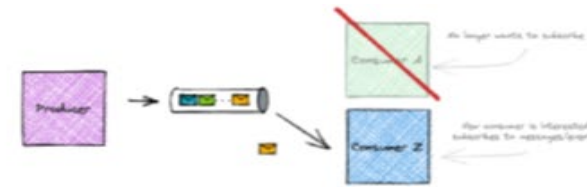
Her kuyruk bir veya daha fazla tüketiciye sahip olabilir.

Producer publishes ve consumers subscribe.



Push model

Events are pushed out to consumers
Channel pushes events to subscribers

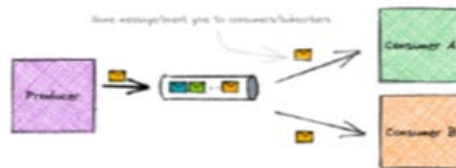


Flexible

Flexible to add new consumers and remove
Can gives agility to teams/business

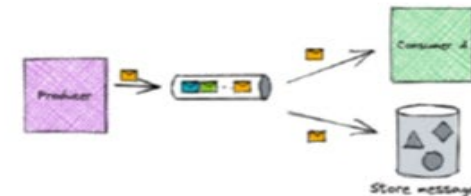
Understanding Publish/Subscribe (Pub/Sub)

Publishing messages to many downstream subscribers/consumers



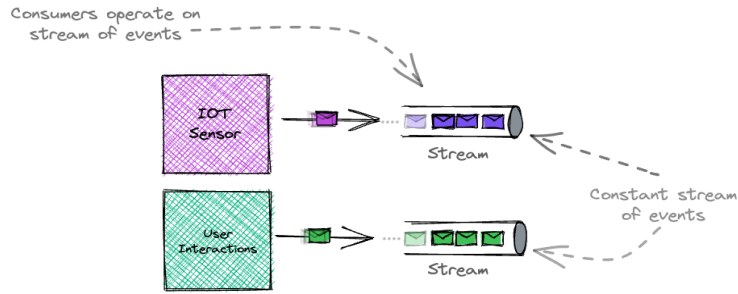
Publish to many consumers

Publish messages/events to many subscribers
Each subscriber gets copy of event to process



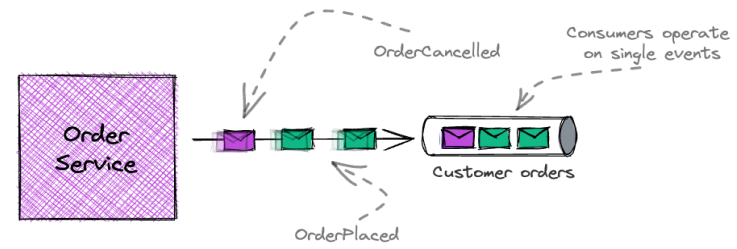
Listening to your events (debugging)

Create listeners to debug see messages/events
See events without impacting consumers



Streaming events

Unbounded series of events
Common examples include user click streams / IOT / transactions

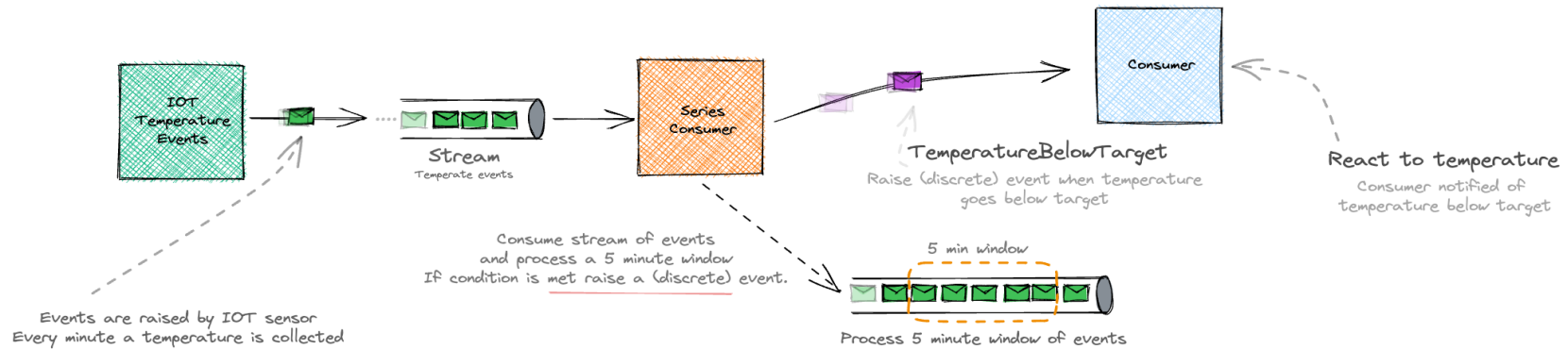


Discrete events

Distinct or individual events that are significant facts.
Common examples would be OrderPlaced,

Understanding stream and discrete events

What is the difference between streaming and discrete events? How can you use them?

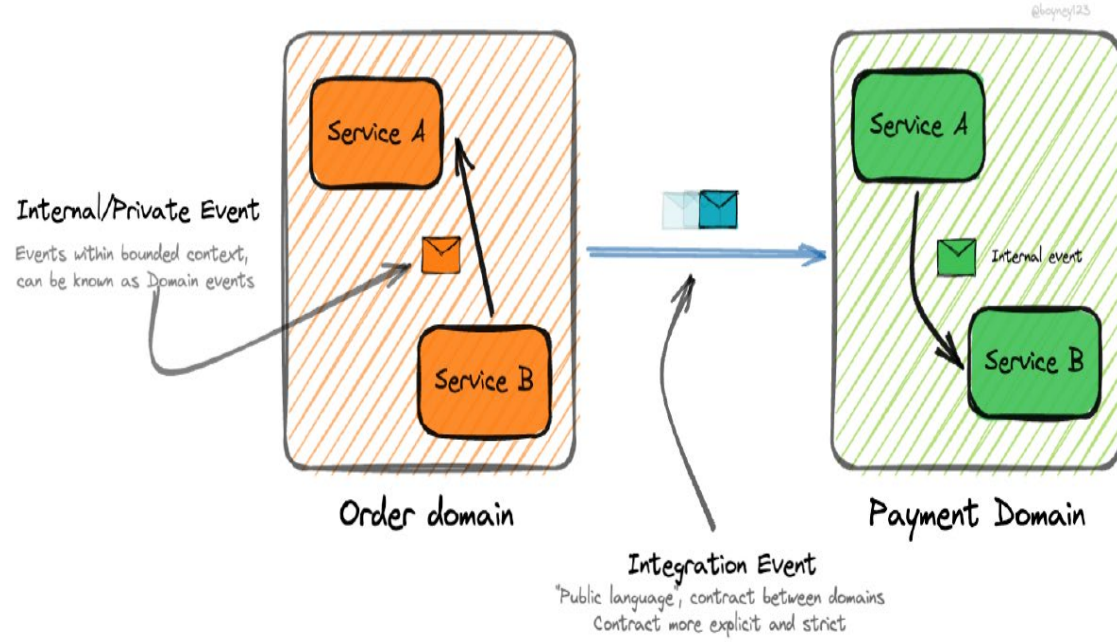


Using streaming and discrete events

Use both streaming and discrete events together.
Consume streams of events, process them and raise events from them.

Internal vs External Events

What's the difference? Private vs Public events?



Private / Internal: Uygulama detayları açıktır.

Alan (domain) tarafından anlaşılacak dil kullanılır.

Sözleşme (contract) önemlidir.

Servislerin sınırlarına göre kurallar gevşetilebilir.

Public/ External: Alanın uygulama detayları açığa çıkamaz.

Olaylara sınırlar ölçüsünde paylaşılan bir dil hakimdir.

Tahminde bulunmak mümkün değildir.

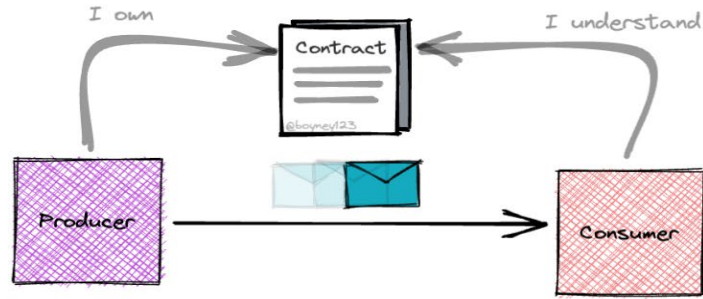
Sözleşme önemlidir.

Mevcut alan dışında olayları kullananlar hakkında fazla bilgi içermez.

Explicit vs Implicit Events

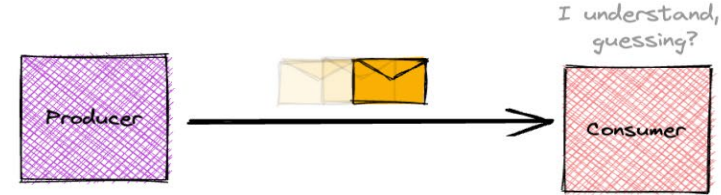
Clear and unclear understanding of events

@boyney123



Explicit Event

Both parties understand contract and intent



Implicit Event

Understood, but not clearly expressed intent or contract

Explicit Events: Olay ve amacı net olmalıdır.
Açık adlandırma kuralları uygulanır.
Sözleşmeler şematik olarak tanımlanır.
Producers ve Consumers için yapılacaklar bellidir.

Implicit Event: Olayın amacı belirsizdir.
Adlandırma kuralları belirlenmemiştir.
Varsayımlar yapılabilir.
Sözleşme olmadan önemli değişiklikler mümkündür.

Event Partitioning /Event Batching

Olay Bölümleme / Olay Gruplama

- ❑Örneğin e-ticaret uygulamasında **sipariş bölümleme**, paralel işleme gerçekleştirerek ölçeklenebilirliği etkinleştirir.
- ❑Olayların sipariş **oluşturma, güncelleme ve iptal** gibi olayların kesin sırasını kesinleştiren önemli bir *tasarım şablonudur*.
- ❑Veriler bölümlenerek, büyük veri kümeleri bir anahtara göre daha küçük, yönetilebilir alt kümelere (parçalara) bölünür.
 - ❖Sonuçta, ilgili olayların doğru sırada işlenmesi sağlanır.
- ❑Olaylar grup halinde işlendiği (**event batching**) durumlarda, birbirine bağlı olaylarda sıra korunacaktır.
- ❑Bu şablonun bir diğer önemi mesaj sırasını korumada sistemlerin güvenilirliğini korumasıdır.

Sonuç olarak Olay Kaynaklama (sourcing) ve Olay Bölümleme(partitioning) teknikleri uygulamaya esneklik ve ölçeklenebilirlik sağlar.

Mesaj Sıralama Zorluklarının Sebebi Nedir?

□ Concurrency and Parallel Processing -Eşzamanlılık ve Paralel İşleme

- ❖ Sistem aynı anda birden fazla olayı işler.
- ❖ Ancak paralellik, işlerin sırasını bozabilir.
 - ✓ Örneğin, iki servis ilgili olayları işleyebilir, ayrı ayrı çalıştıkları için, olayların sırasız olarak tamamlanmasıyla sonuçlanabilir.

□ Ağ gecikmeleri -Network Delays

- ❖ Dağıtık (distributed) sistemlerde sistemini parçaları farklı sunucularda olduğu için ağ gecikmeleri yaşanır.
- ❖ A olayı önce, B olayı sonra gönderilse bile, ağdaki bir aksaklık nedeniyle A, B'den daha sonra gelebilir.
- ❖ Bir servis çöktüğünde ve yeniden denediğinde, aynı karışıklığa neden olabilir.

□ Mesaj Aracıları (Brokers)

- ❖ Olay aracılığı (Kafka, RabbitMQ), yükleri yönetmek için olayları birden fazla bölüme ayırır.
- ❖ Aynı varlıkla (sipariş veya kullanıcı gibi) ilgili olaylar bölümlere ayrıldığında, işleme sırası karışabilir.

□ Event Duplication –Olay Tekrarı

- ❖ Yeniden işleme veya ağ sorunları sonucunda bazen olaylar birden fazla gönderilir.
- ❖ Yinelenen bir olayı yanlış zamanda işlenmesi ile karışıklıklar meydana gelir.

Bu Problemler Nasıl Çözölmektedir?

- ❑ **Olayın Kaynaklanması (Event Sourcing):** Olayların doğru sıralanmasını sağlar.
 - ❖ Her olayı sırayla depolamak ve geçmiş olayları durumu yeniden oluşturmak üzere kullanmaktır.
 - ❖ Olaylara sürüm numaraları ekleyerek, bunların sırayla işlenmesini sağlar.
- ❑ **Eşgüçte Olay İşleyicileri (Idempotent Event Handlers)** Olay işleyici aynı olayı birden çok defa sorunsuz işleyebiliyorsa, sırayı bozmadan olay tekrarları işlenebilir.
- ❑ **Anahtara Göre Bölümleme (Partition by Key)** Kafka gibi brokerlar için, aynı anahtara sahip tüm olayların (belirli bir kullanıcı veya işlem gibi) aynı bölüme gitmesi sağlar.
 - ❖ Bir bölümdeki olaylar sırayla işlendiğinden, sıra sorunları olmaz.

Bu Problemler Nasıl Çözölmektedir?

Zaman Damgaları ve Sıra Numaraları (Timestamps and Sequence Numbers)

Her olayı bir zaman damgası veya sıra numarasıyla etiketlemek, daha sonra yeniden sıralamalarına yardımcı olabilir.

- ❖ İki olay yanlış sırada görünürse, işleme başlamadan önce bunları düzeltmek için etiketler kullanılır.

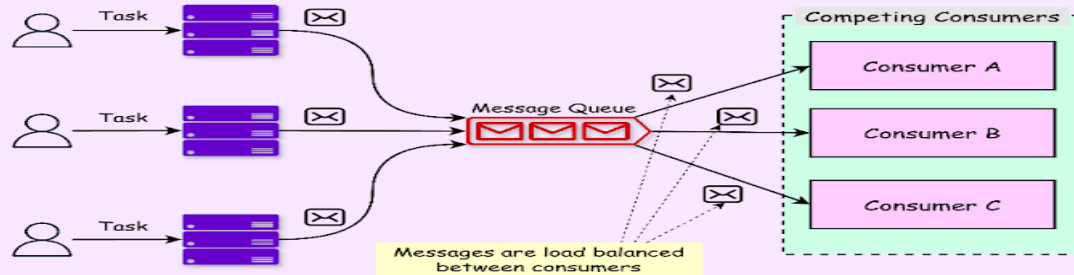
Nihai Tutarlılık – Eventual Consistency Çalışma sırasında her şeyin mükemmel bir şekilde tutarlı olması gerekmez; bu durumlarda nihai tutarlılığa güvenilir.

- ❖ Olaylar geçici olarak sırasız olsa bile, belli bir zaman sonra düzeleceği anlamına gelir.

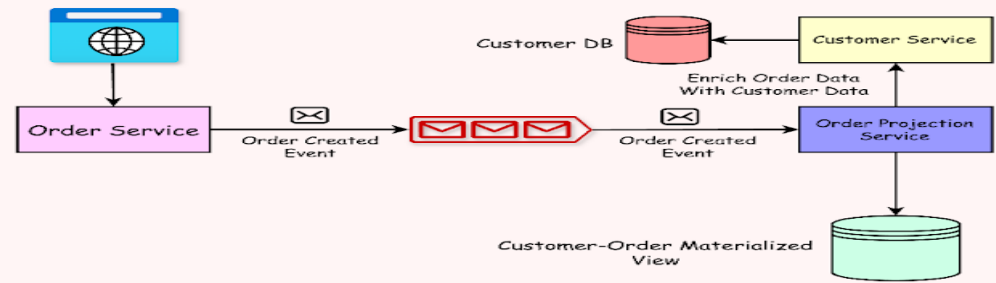
Olay Gruplama (Event Grouping) Olaylar bir grup olarak işlendiğinde, özellikle birbirine bağlı olaylar için, sıranın korunması kolaylaşacaktır.

A Cheatsheet On Event-Driven Architectural Patterns

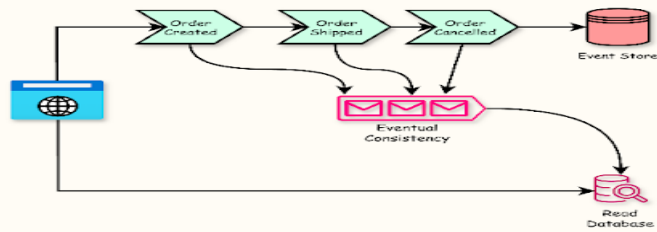
1 Competing Consumer Pattern



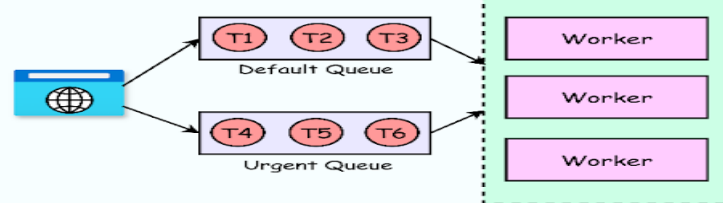
2 Consume and Project Pattern



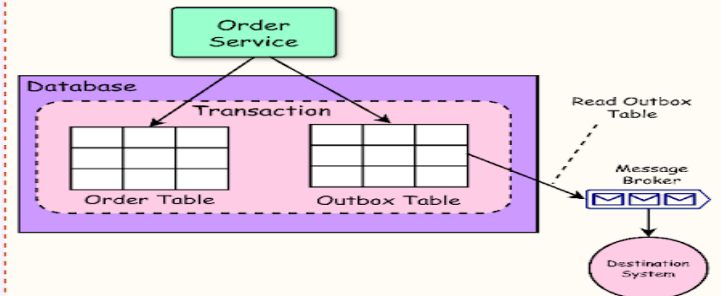
3 Event Sourcing



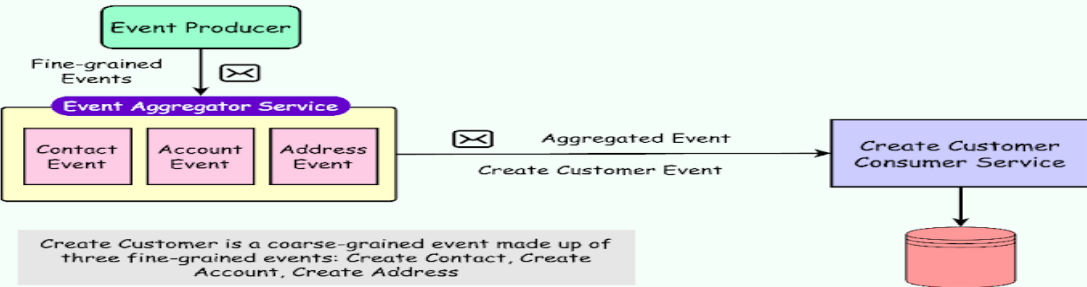
4 Async Task Execution Pattern



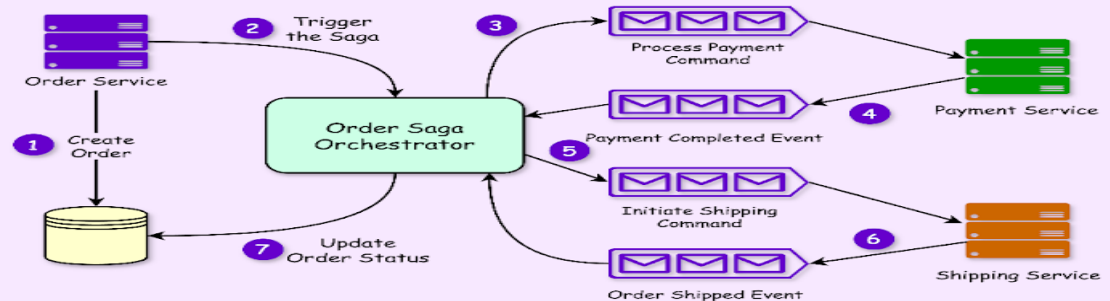
5 Transactional Outbox Pattern



6 Event Aggregation Pattern



7 Saga Pattern



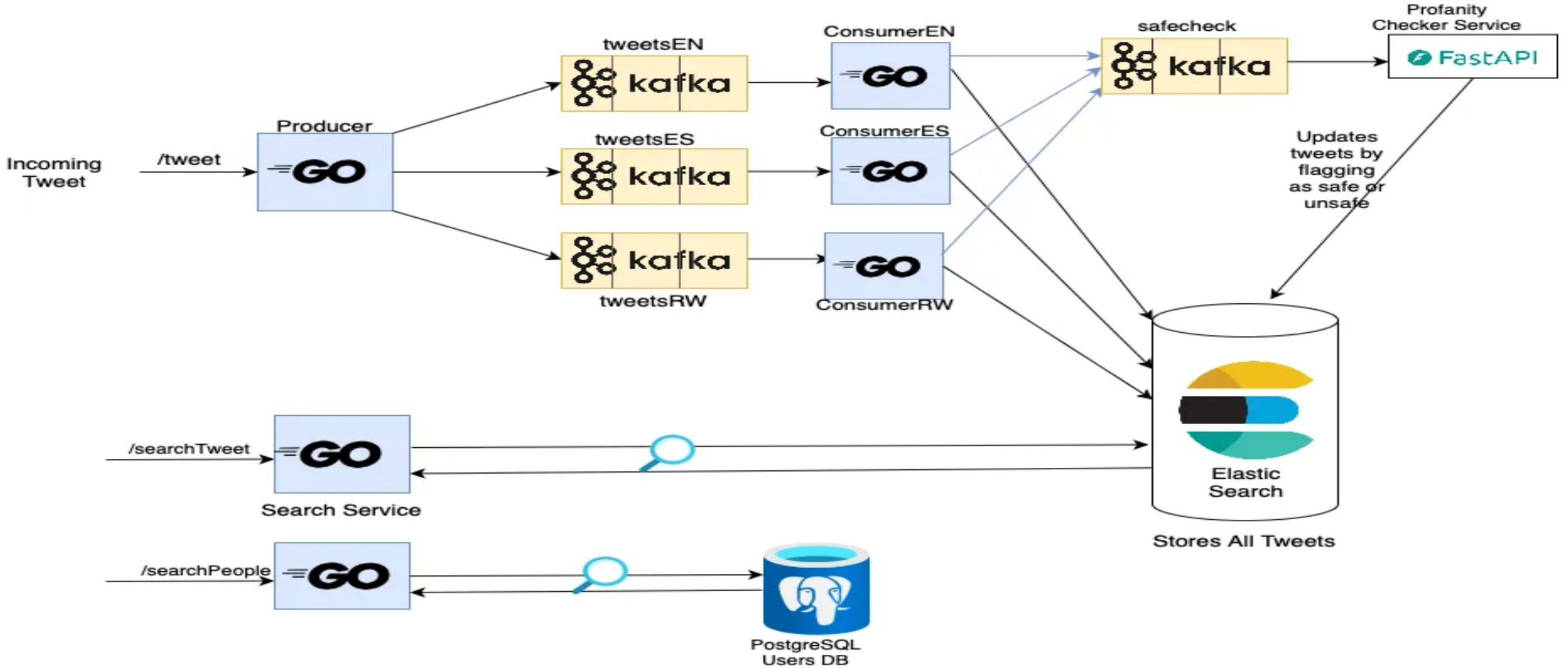
EDA Şablonlarının Avantajları

Fault Tolerant – Hata Toleransı: Sistemler bazı bileşenlerin arızalanması durumunda bile çalışmaya devam eder.

Etkin kaynak kullanımı : Servisler talebe göre bağımsız olarak ölçeklenir.

Dinamik ve esnek iş akışları İşletme değişen gereksinimlere ve pazar koşullarına uyum sağlar.

Uygulamalarda gerçek zamanlı veriler ve dağıtık servislerin kullanımı arttıkça, örneğin mesaj sırasını belirlemek çok kritik olacaktır.



- ❑ Tweet atıldığında bir olay tetiklenir.
- ❑ Tweet'i Kafka Event Broker'a göndermek ve İngilizce, İspanyolca ve Arapça gibi dillere göre sınıflandırmak **Producer Service** sorumluluğundadır.
- ❑ Kafka, **asynchronous events** için (güvenilir) akış sağlar ve **Producers** dan bağımsız olarak **Consumers** a sunar.
- ❑ Dillere göre oluşturulmuş Tweetlerin konulara göre akışı (Streaming of Tweets) Kafka tarafından etkin olarak işlenir.
- ❑ **Consumers** tweetleri indirir, içerik doğrulaması yapar ve sistemlere günceller.
Örneğin, **Profanity Checker Service** (Küfür Kontrol Servisi), bir tweetin güvenli /güvensiz olarak sınıflandırılma durumunu kontrol eder.
- ❑ İşlenen Tweetler **Elastic Search** tarafından depolanır; böylece arama sonuçlarının daha hızlı bulunması sağlanır.
- ❑ Elastic Search, tweetlere hızlı arama sağlar.
 - ❖ searchTweet uç noktası (end point)
 - ❖ PostgreSQL, /searchPeople gibi kullanıcılarla ilgili sorguları işler ve kullanıcılar hakkında yapılandırılmış verileri korur.