T.C. SANAYİ VE TEKNOLOJİ BAKANLIĞI

#MİLLİ TEKNOLOJİ HAMLESİ

TÜBİTAK

RABBITMQ TEKNOLOJİSİ

ARAŞTIRMA SERİSİ - SAYI 15

BİLGEM

YAZILIM TEKNOLOJİLERİ ARAŞTIRMA ENSTİTÜSÜ

Simge ve Kısaltmalar

Kısaltmalar	Açıklama
TÜBİTAK	Türkiye Bilimsel ve Teknolojik Araştırma Kurumu
BİLGEM	Bilişim ve Bilgi Güvenliği İleri Teknolojiler Araştırma Merkezi
YTE	Yazılım Teknolojileri Araştırma Enstitüsü
AMQP	Advanced Message Queuing Protocol (Gelişmiş Mesaj Sıralama Protokolü)
DLQ	Dead Letter Queue (Ölü Mesaj Kuyruğu)
DLX	Dead Letter Exchange (Ölü Mesaj Yönlendiricisi)
FIFO	First In, First Out (İlk Giren, İlk Çıkar)
RabbitMQ	Rabbit Message Queue (Rabbit Mesaj Kuyruğu)
TTL	Time-to-Live (Yaşam süresi)

Yazar
Girayhan YILDIRIM

Yayın Koordinatörü
Kübra ERTÜRK

Editörler
Ahmed Enis ERKAYA
Beyza ŞENEL
Tuğçe YILMAZ

Tasarım
Şeyma KOÇER

İletişim: 0(312) 289 92 22 - yte.bilgi@tubitak.gov.tr

https://bilgem.tubitak.gov.tr/yte

İçindekiler

Ön Söz

TÜBİTAK BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü (YTE), 2012 yılından bu yana yazılım teknolojilerinde Ar-Ge faaliyetleri yürüten bir araştırma kuruluşudur. Araştırma faaliyetlerinde elde ettiği birikimini stratejik, hassas ve kritik projeler yürüterek kamu adına hayata geçirmekte; kurumlarımıza dijital dönüşüm, yazılım geliştirme teknolojileri ve kalite süreçleri konusunda danışmanlık vermektedir.

TÜBİTAK BİLGEM YTE tarafından hazırlanan Araştırma Serisi ile kurum içi içerik üretme çalışmalarının yaygınlaştırılması ve hazırlanan içeriklerin sektörün erişimine açılması amaçlanmaktadır. Araştırma Serisi'nde yayınlanan çalışmalar TÜBİTAK BİLGEM YTE çalışanlarının projelerde elde ettiği bilgi birikimini paylaşmak adına derlenmiştir. Bu çalışmalar ile ülkemizin yazılım sektörüne katkı sağlanılması hedeflenmektedir.

Giriş

Mesaj kuyrukları, bilgisayar bilimlerinde önemli bir kavramdır ve çeşitli sistemlerde kullanılır. Yazılım sistemlerinde yapılması gereken işlevleri mesajlar kullanarak belirli bir sıralama ile yapılmasına olanak sağlayan veri yapısı modelidir. Mesaj kuyrukları, işlenmeyi bekleyen mesajların geçici olarak saklandığı bir yapıdır. Bu mesajlar, genellikle bir kaynaktan bir hedefe belirli bir işlevi sağlayabilmek amacıyla ilgili birime iletilmek üzere gönderilir. Örneğin, bir sunucu uygulaması kullanıcı tarafından gönderilen istekleri işlemek için bir mesaj kuyruğu kullanabilir. Bu istekler, kuyruğa eklenir ve ardından sırayla işlenir.

Mesaj kuyrukları, sistemler arasında asenkron iletişimi kolaylaştırmak için yaygın olarak kullanılır. Bir sistem veya servis, diğer bir sistemle veya servisle doğrudan etkileşime girmeden önce mesaj kuyruğuna bir mesaj gönderebilir. Bu sistemler arasında bağlantı sağlamanın yanı sıra, yüksek talep dönemlerinde iş yükünü dengelemek için de kullanılabilir. Örneğin, bir uygulamada kullanıcıların isteklerini işlemek için bir mesaj kuyruğu kullanarak yoğun zamanlarda performansı koruyabilir ve sistemin yükü optimize edilebilir.

Mesaj kuyrukları genellikle FIFO (First In, First Out - İlk Giren İlk Çıkar) prensibiyle çalışır. Yani kuyruğa ilk eklenen mesajlar ilk olarak işlenir. Ancak bazı durumlarda öncelikli kuyruklar veya öncelikli mesajlar için özel düzenlemeler yapılabilir. Bu da mesaj kuyruklarını esnek bir şekilde isteğimiz doğrultusunda kullanmamızı sağlar.

Bu yapılar sistemlerin ölçeklenebilirliğini ve güvenilirliği artırır. Ek olarak, farklı bileşenler arasında esnek, dayanıklı bir iletişim sağlar. Özetle mesaj kuyrukları, dağıtık sistemlerde iletişimi kolaylaştıran ve iş yükünü düzenleyen önemli bir araçtır. Bu araştırmada günümüzde yaygın şekilde kullanılan mesaj kuyruğu teknolojilerinden biri olan **RabbitMQ** teknolojisinin; yapısı, mimarisi, bileşenleri, bu bileşenlerin amaçları ayrıca bu teknolojinin avantaj ve dezavantajlarından bahsedilecektir.

Nedir, Niçin Kullanılır?

1. RabbitMQ Nedir?

RabbitMQ, popüler bir açık kaynaklı mesaj kuyruğu yazılımıdır. AMQP (Advanced Message Queuing Protocol - Gelişmiş Mesaj Kuyruğu Protokolü) standardını uygular. AMQP, istemci uygulamalarının sunucuyla konuşmasına ve etkileşime girmesine olanak tanıyan, kullanılan platformdan bağımsız olarak sistemler arasında haberleşmeyi ve birlikte çalışabilirliği sağlayan standart bir protokoldür. Temel olarak protokolü kullanan RabbitMQ dağıtık sistemler arasında mesajların güvenli bir şekilde iletilmesini sağlar.

2. Temel Kullanım Alanları

Mikroservis Mimarileri (Microservice Architecture): Mikroservis mimarileri, uygulamaları küçük ve bağımsız hizmetlere böler. RabbitMQ, bu hizmetler arasındaki iletişimi sağlar, böylece sistemlerin esnek, ölçeklenebilir ve bağımsız olmasına olanak tanır.

Olay Tabanlı Mimariler (Event Driven Architecture): RabbitMQ, olay tabanlı mimarilerde kullanılabilir. Bir olay meydana geldiğinde, bu olayı alan ve işleyen farklı bileşenler arasında iletişimi kolaylaştırır.

Kuyruk Sistemi (Queue System): RabbitMQ, işlenmek üzere bekleyen işleri kuyruğa alır ve bu işleri tüketicilere dağıtarak işleri paralel hale getirir. Bu, yüksek iş yüklerini dengelemek ve işleri etkili bir şekilde yönetmek için kullanılabilir.

Uygulama Entegrasyonu (Application Integration): RabbitMQ, farklı uygulamalar arasında iletişimi kolaylaştırır. Örneğin, bir web uygulaması mesaj göndererek arka uç servisine veri aktarabilir. Aynı şekilde başka bir uygulama veya servisten aldığı mesaj aracılığıyla süreçleri işletebilir.

RabbitMQ Bileşenleri

1. Producer (Üretici)

RabbitMQ'da "Producer (Üretici)", mesajları RabbitMQ Broker'ına (Aracısına) gönderen bölümü ifade eder. Producer'lar , mesajları RabbitMQ'ya ileterek bu mesajların bir veya birden fazla kuyruğa yönlendirilmesini sağlarlar. Producer'ların temel görevi, belirli bir konsept veya olayla ilgili mesajları üretmek ve bu mesajları RabbitMQ'ya göndermektir. Örneğin, bir sistem içindeki bir uygulamada bir olay gerçekleştiğinde (alışveriş siteminde bir sipariş verildiğinde), bu olaya ilişkin bir mesaj üretebilir. Bu mesaj, RabbitMQ Broker'ına iletilir.

2. Consumer (Tüketici)

RabbitMQ'da "Consumer (Tüketici)", RabbitMQ Broker'ından mesajları alan ve bu mesajları işleyen bölümü ifade eder. Consumer'lar, belirli bir kuyruğa abone olarak, RabbitMQ Broker'ından gelen mesajları alır ve bu mesajları işleyerek sistem içinde bir eylem gerçekleştirirler. Örneğin, bir kullanıcıya asenkron olarak e-posta gönderilmelidir. Bu durumda e-posta için kullanılan servis Consumer olarak bu e-posta gönderimi için üretilen (yayınlanan) mesajın bulunduğu kuyruğu dinleyerek (yani o kuyruğa ait mesajı tüketerek) ilgili mesajları alır ve e-posta göndermek üzere ilgili mesajı tüketir.

RabbitMQ'da bir Consumer'ın yapabileceği temel olaylar şu şekildedir:

- RabbitMQ Broker'dan mesajları almak için belirli bir kuyruğa veya Exchange'e (Yönlendiriciye) abone olmak.
- Mesajları alıp işlemek.
- Mesajların başarıyla işlendiğini onaylamak (acknowledge).
- Gerekirse, bir hata durumunda işlenmeyen mesajları kuyruğa geri koymak.

3. Broker (Aracı)

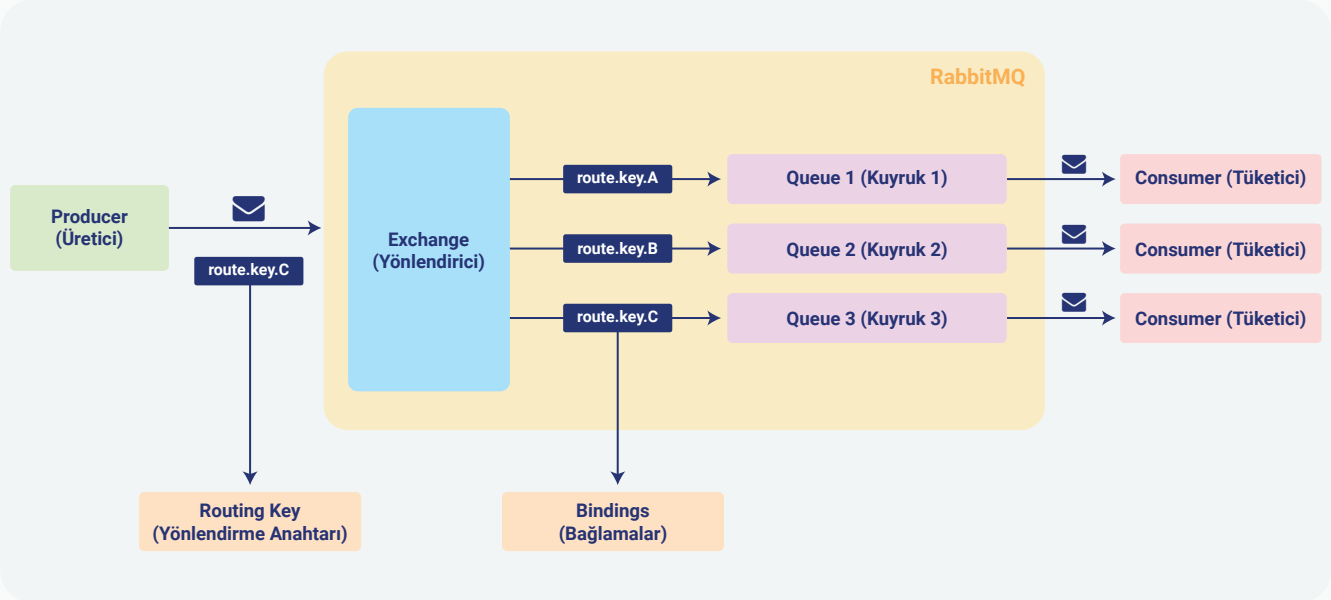
Broker, mesajların alışverişini kolaylaştırmak için gerekli mekanizmayı sağlar. Üretici mesajı gönderir, sonrasında tüketici mesajı alır. Broker, üretici tarafından gönderilen mesajları kabul eder ve gerekirse çevirir. Ardından mesajları tüketici(ler) tarafından alınana kadar bir mesaj kuyruğunda düzenler ve tutar. Kısaca özetlemek gerekirse, bir mesajın yayınlamasından tüketilmesine kadar ki süreçte bileşenler arası düzenlemeyi sağlar ve yönetir.

4. Exchange (Yönlendirici)

Exchange, producerin ürettiği mesajı alıcıya veya kuyruğa yönlendirme işlemini gerçekleştiren yapıdır. Producer ürettiği mesajı Exchange'e iletir, Exchange gelen mesajı içerdiği bilgiler ile birlikte ilgili kuyruğa ekler ve dinleyen bir Consumer varsa kuyruktan sıradaki mesajı işlemek üzere alır. Kısaca yaptığı görev, belirtilen Routing Key'e (Yönlendirme Anah-tarına) göre aldığı mesajı ilgili Queue'ye (Kuyruğa) iletmektir.

4.1. Direct Exchange (Doğrudan Yönlendirme)

Mesaj gönderilirken belirtilen Routing Key ile eşleşen Binding (Queue tanımlanırken Exchange'e atanmış key) ile ilgili Queue'ye aktarılır. Aynı zamanda varsayılan olarak kullanılan Exchange türüdür, her Queue otomatik olarak bu Exchange türüne Routing Key ile atanır.



Şekil 1 – Direct Exchange

Şekil 1'deki örnek incelendiğinde Producer tarafından üretilen mesaj "route.key.C" Routing Key ile Exchange'e gönderildiğinde, eşleşen Binding Key'e (Bağlama) ait Queue'ye aktarıldığı görülecektir. Burada Routing Key ile Bindig birebir eşleşmelidir.

4.2. Topic Exchange (Etiket ile Yönlendirme)

Topic Exchange, belirli bir kalıp veya doğrudan yönlendirme gibi Routing Key ile Queue'ya erişebilir. Direct Exchange'in aksine birden fazla Queue'ya mesaj iletebilir. Yine aynı şekilde gönderilen mesajdaki Routing Key, eşleşen Binding Key'e sahip Queue'ya aktarılır. Ancak Binding Key ile Routing Key noktalarla ayrılmış şekilde kelimelerden oluşmalıdır. Direct Exchange'te böyle bir zorunluluk yoktur. Kelime 255 byte'a kadar herhangi bir metin olabilir.

Binding Key tanımlanırken iki özel durum söz konusudur.

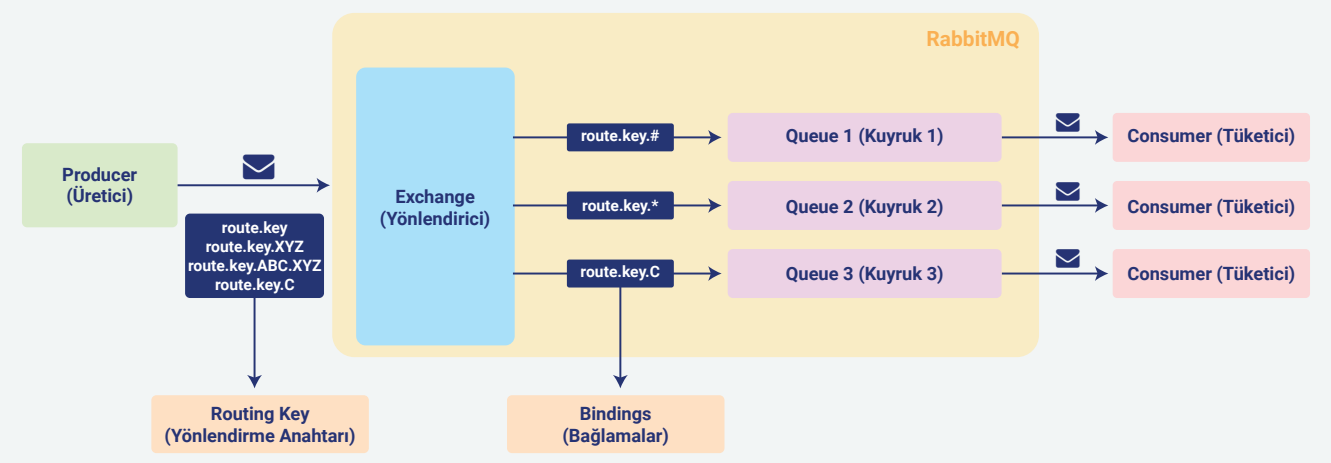
- *** (Yıldız):** Kullanılan yerde yerine **bir** tane belirteç geleceği garanti edilir.
- **# (Hash):** Kullanılan yerde **sıfır veya daha fazla** belirteç gelebilir.

Routing Key'in **"route.key.*"** olduğu durumda bazı örnekler:

- route.key.XYZ → ✓
- route.key.ABC → ✓
- route.key → ✗
- route.key.ABC.XYZ → ✗

Routing Key'in **"route.key.#"** olduğu durumda bazı örnekler:

- route.key.ABC → ✓
- route.key → ✓
- route.key.ABC.XYZ.QWE → ✓



Şekil 2 – Topic Exchange

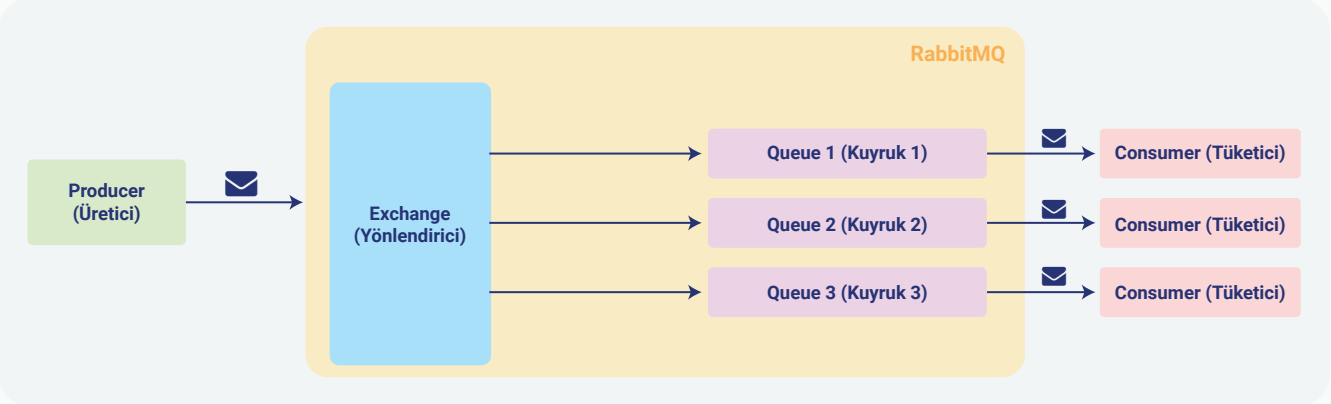
Şekil 2'de belirlenen Routing Key'lerin hangi Queue'lara iletileceği incelendiğinde:

- route.key → Sadece Kuyruk 1'e iletilir.
- route.key.XYZ → Kuyruk 1 ve 2'ye iletilir.
- route.key.ABC.XYZ → Kuyruk 1 ve 2'ye iletilir.
- route.key.C → Kuyruk 1 ve 3'e iletilir.

Ayrıca Topic Exchange için herhangi bir Route Key belirtilmediği sürece Fanout Exchange gibi çalışır.

4.3. Fanout Exchange (Yayarak Yönlendime)

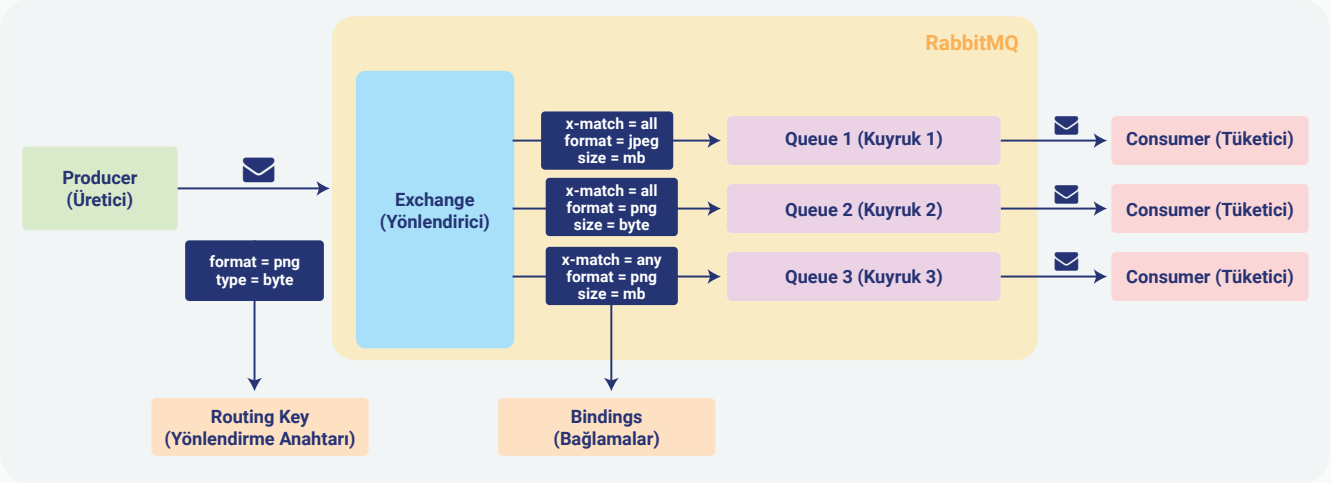
Fanout Exchange'de gönderilen mesajlar kopyalanarak yönlendirilebilecek bütün Queue'lara, Routing Key'e bakılmaksızın gönderilir. Bu Exchange türünde Routing Key'in bir önemi yoktur.



Şekil 3 – Fanout Exchange

4.4. Header Exchange (Başlık ile Yönlendirme)

Header Exchange, iletişimi Routing Key yerine mesajın başlık kısmında bulunan **header key-value**'larına göre yapılmaktadır. Exchange, gelen mesajları hangi Queue'ya yönlendireceğine karar verebilmesi için Binding Key ayrıca **"x-match"** başlığını barındırmalıdır. Bu argüman **all** veya **any** değerini alabilir. **all** değerini alması durumunda tüm değerlerinin eşleşmesi, **any** olması durumunda bir değerin eşleşmesi yeterlidir.



Şekil 4 – Header Exchange

Şekil 4'te tanımlanmış olan Header Exchange ile ilgili **header-key-value** bilgisine bakıldığında ilgili mesaj Kuyruk 2 ve 3'e iletilir.

5. Queue (Kuyruk)

RabbitMQ'da bir "Queue", mesajların geçici olarak depolandığı ve bir veya daha fazla **Tüketici** tarafından işlenmek üzere bekletildiği bir yapıdır. Kuyruklar, dağıtık sistemler arasında mesajların güvenilir bir şekilde taşınmasına olanak tanır. Kuyruklar, **Üretici** tarafından gönderilen mesajları depolar ve bu mesajları tüketen uygulamalara veya diğer kuyruklara ileterek asenkron bir iletişim modeli sağlar. Her kuyruk belirli bir amaç için kullanılır ve bu amaç uygulamanın ihtiyaçlarına bağlı olarak değişebilir.

RabbitMQ kuyruklarının temel özellikleri şunlardır:

- **Mesaj Depolama:** Kuyruklar, gelen mesajları geçici olarak depolar. Bu, bir mesajın üretilip hemen tüketici tarafından işlenmediği durumlarda aktif olarak kullanıma hazır halde bekletilir.
- **FIFO Sıralama:** Kuyruklar genellikle FIFO prensibine göre çalışır, yani ilk gelen mesaj ilk çıkar. Bu, mesajların sırasının korunmasını sağlar. Eğer mesaj sırasının önceliklendirme üzerinden yönetilmesi isteniyorsa o şekilde de kullanılabilir.
- **Multiple Consumers (Çoklu Tüketiciler):** Bir kuyruğa birden fazla tüketici bağlanabilir. Bu sayede, aynı mesaj birçok Consumer tarafından işlenebilir, böylece iş yükü paylaşılabilir ve sistem ölçeklenebilir hale gelir.

Queue, mesajların depolandığı kuyruktur. RabbitMQ'da 3 tür kuyruk vardır:

- **Durable Queue (Dayanıklı Kuyruk):** Kuyruk, RabbitMQ sunucusu yeniden başlatıldığında bile korunur. 'Durable' değeri **false** olarak ayarlanırsa yeniden başlatıldığında silinecektir.
- **Exclusive Queue (Özel Kuyruk):** Kuyruk, sadece oluşturulan bağlantıya özeldir. Bağlantı sonlandırıldığında kuyruk da silinir.
- **Auto-Delete Queue (Otomatik Silinen Kuyruk):** Kuyruk, en az bir Consumer bağlı değilken son mesajı aldıktan sonra otomatik olarak silinir.

6. Routing Keys (Yönlendirme Anahtarları)

RabbitMQ'da, "Routing Key" bir mesajın belirli bir kuyruğa nasıl yönlendirileceğini belirlemek için kullanılan bir etikettir. Mesaj Producer tarafından Exchange'e gönderildiğinde Routing Key, bu mesajın hangi kuyruklara iletileceğini belirlemek için Exchange tarafından kullanılır. Exchange ve Queue arasındaki ilişki, Exchange türü ve Routing Key kullanımıyla belirlenir. Routing Key, esnek bir iletişim modeli sağlar çünkü mesajlar belirli kriterlere göre yönlendirilebilir. Bu, sistem içinde farklı Consumer'ların belirli türdeki mesajları almasını ve işlemesini sağlar.

7. Bindings (Bağlamalar)

Exchange ile Queue arasında kurulacak bağlantının yönlendirme kuralıdır. Exchange aldığı mesajları bu kurala göre ilgili kuyruklara dağıtır.

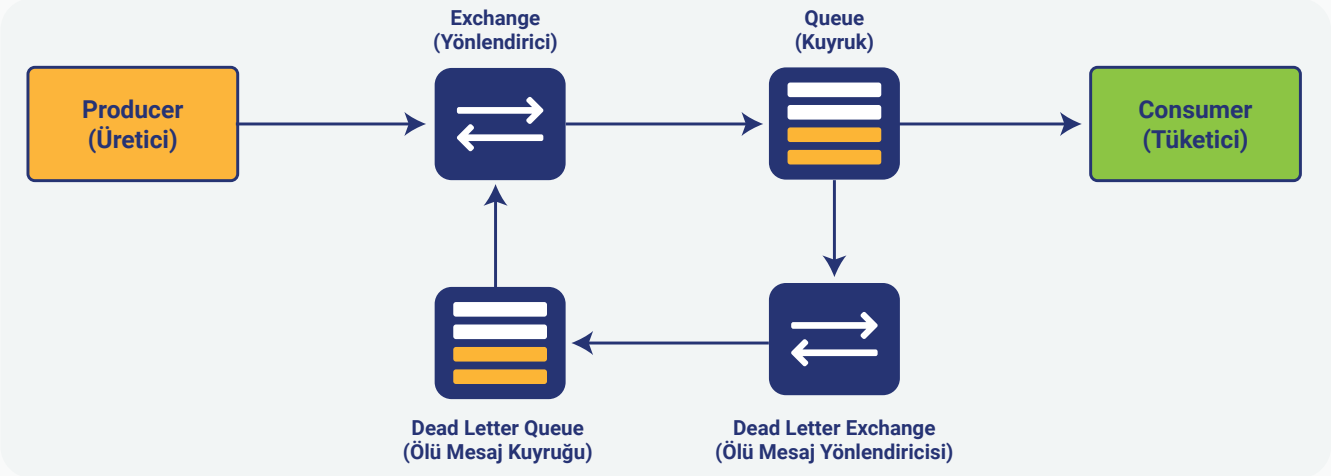
Dead Letter Exchange ve Queue (Ölü Mesaj Yönlendiricisi ve Kuyruğu) Mekanizması

Dead Letter Exchange (DLX), mesajın belirli nedenlerle tüketilemediği durumlarda yönlendirildiği özel bir Exchange türüdür. Dead Letter Queue (DLQ) ise bu Exchange üzerinden mesajları alan özel bir kuyruktur.

Bu mekanizmaya hangi durumlarda ihtiyaç duyulur:

- Bir mesajın TTL (Time-To-Live) süresi dolmuşsa.
- Kuyruk kapasitesi dolmuşsa ve yeni mesajlar için yer yoksa.
- Mesaj, Consumer tarafından herhangi bir şekilde reddedilirse (Nack or Reject).

Bu gibi durumlarda, mesajlar doğrudan silinmek yerine DLX ve DLQ üzerinden işlenir, böylece bu mesajlar üzerinde daha sonrasında farklı işlemler gerçekleştirilebilir. RabbitMQ kendi içerisinde bu mekanizmayı sunmaktadır. Yapılması gereken tek şey Queue tanımlaması yapılırken argüman olarak **"x-dead-letter-exchange"** etiketi verilmesidir. Bu etiketin verildiği Queue'larda yukarıdaki koşullardan birisi gerçekleştiğinde DLX aracılığıyla DLQ'ya yönlendirilecektir. DLQ aracılığıyla iletilemeyen mesajlar amacına yönelik olacak şekilde daha sonra kullanılabilir (Şekil 5).



Şekil 5 – DLX ve DLQ mekanizması

Avantajları ve Dezavantajları

1. Avantajlar

- **Güvenilirlik:** RabbitMQ, sistem arızaları veya ağ sorunları durumunda bile ileti teslimatını sağlar. Mesaj onayı ve kalıcılık gibi özellikler, mesajların kaybolmamasını sağlar. Kuyruğa veya RabbitMQ Broker'ına herhangi bir sebepten dolayı zarar gelmesi durumunda, sistem

içerisinde mesajların disk üzerinde kaydedilerek veya yedeklenerek mesaj kaybının önüne geçilebilmesine olanak tanır.

- **Esneklik:** AMQP, MQTT ve STOMP gibi birden fazla mesajlaşma protokolünü destekler, bu da farklı türde uygulamalar ve entegrasyon senaryoları için çok yönlülük sağlar. Örnek olarak bir servis var ve bu servis birçok uygulama ile kullanılan mesaj kuyrukları üzerinden iletişim kurarak diğer uygulamalara istekler yollamaktadır. Diğer uygulamaların kendi içerisinde farklı mesaj protokollerini kullandığı düşünüldüğünde, bu durumda o uygulamaya özgü kuyruklar tarafından, uygulamanın kullandığı mesaj protokolü özelinde mesajlar gönderilerek kolay bir şekilde servislerin adaptasyonu sağlanır.
- **Ölçeklenebilirlik:** RabbitMQ büyük mesaj hacimlerini yönetebilir ve kümelere (cluster) daha fazla düğüm ekleyerek yatay olarak ölçeklendirilebilir. Bu ölçeklendirme, sistem içerisindeki yük dağılımını sağlayarak daha performanslı bir şekilde kuyrukların çalışmasını sağlar.
- **Yönlendirme ve Filtreleme:** İleri düzeyde mesaj yönlendirme ve filtreleme yetenekleri sunar. Daha önce bahsedilmiş olan yönlendirme metotlarında olduğu gibi mesajların başlıkları, yönlendirme anahtarları veya mesaj içeriği gibi kriterlere göre belirli kuyruklara yönlendirilmesine olanak tanır. Bu sayede mesajlar servislere ve işlevlerine göre ayrılarak anlaşılmasını kolaylaştırır ve amaçları doğrultusunda kullanımı sağlar.
- **Yönetim ve İzleme:** RabbitMQ, web tabanlı bir yönetim arayüzü ve kapsamlı izleme yetenekleri sağlar. Bu sayede mesajlaşma altyapısını yönetmek ve izlemek kolay bir hale gelmiş olur.

2. Dezavantajlar

- **Karmaşıklık:** RabbitMQ'yu kurmak ve yapılandırmak, özellikle mesaj kuyruğu sistemlerine aşina olmayan kullanıcılar için karmaşık olabilir. Değişimleri (Exchange), kuyrukları, bağlamaları yapılandırmak ve yönlendirme anahtarlarını anlamak yeni başlayanlar için zorlayıcı olabilir. Bu da etkili kullanılması için geliştirici tarafında bir efora sebebiyet verecektir.
- **Ölçeklenebilirlik:** Avantajlı bir durum olmasının yanı sıra RabbitMQ, bir kümeye daha fazla düğüm eklenerek yatay olarak ölçeklendirilebilirken, ölçeklendirmenin yönetimi zor olabilir ve yüksek kullanılabilirlik ve performans sağlamak için dikkatli planlama gerektirir. Ayrıca bu durum altyapıdaki karmaşıklığı artırarak problem çözümünde ekstra kaynağa veya efora olan ihtiyacı artırabilir.
- **Kaynak Yoğunluğu-Performans:** RabbitMQ'nun çalıştırılması özellikle mesaj yükü arttıkça işlemci ve bellek gibi kaynakları gerektirir. Optimal performans için yapılandırma, ince ayar ve izleme gereklidir. Mesaj büyüklüğü ve yoğunluğunun artması performans ve yönetim konusunda iyi bir plan ve altyapı ile desteklenmediği sürece büyük bir problemdir.

Sonuç ve Öneriler

Bu çalışmada, RabbitMQ teknolojisinin genel olarak amacı, mimarisi, kullanım şekli, avantaj ve dezavantajları anlatılmıştır. Çalışmada bahsedildiği üzere bu teknolojinin etkin bir şekilde kullanılması projelerin veya sistemlerin daha etkili bir şekilde çalışmasına olanak sağlayacaktır. Mevcut sistemlere kolayca uyum sağlayabildiği için entegrasyon konusunda bir problem yaşanmayacaktır. Servislerin veya sistemlerin birbirinden bağımsız bir şekilde asenkron olarak iletişim kurmasını ve sistem içerisinde yük dağılımını sağlayarak uygulama performansının daha yüksek seviyede olmasını sağlayacaktır. Fakat etkin bir şekilde kullanılması için projenin bu teknolojiyle uyumluluğu, altyapının yeterliliği, geliştiricilerin bu teknoloji hakkında bilgileri ve bu teknoloji kullanılırken uçtan uca kullanmış olduğu izleyici araçlarıyla yönetilmesi sağlanmalıdır. Bu koşullar sağlandığı sürece bu teknoloji uzun vadede kullanılan alanda olumlu etkiler ve gelişmeler sağlayacaktır.

Kaynakça

RabbitMQ Documentation, URL: https://www.rabbitmq.com/docs, (Erişim zamanı; 2, 15, 2024).

Martin Toshev, Learning RabbitMQ, Chapter 1-2, 2015

RabbitMQ Tutorial, URL: https://www.youtube.com/watch?v=iQ4kENLfaNI&list=PLalrWAGybp-B-UHbRDhFsBgXJM1g6T4IvO (Erişim zamanı; 2, 15, 2024).

RabbitMQ: Concepts and Best Practices, URL: https://medium.com/cwan-engineering/rabbit-mq-concepts-and-best-practices-aa3c699d6f08 (Erişim zamanı; 3, 14, 2024).

RabbitMQ ve Exchange Türleri, URL: https://medium.com/devopsturkiye/rabbitmq-ve-exchan-ge-t%C3%BCrleri-46c04cf81ee3 (Erişim zamanı; 3, 15, 2024).

T.C. SANAYİ VE
TEKNOLOJİ BAKANLIĞI

#MİLLİ TEKNOLOJİ HAMLESİ

TÜBİTAK

İşçi Blokları Mahallesi Muhsin Yazıcıoğlu Caddesi No:51/C 06530 Çankaya/ANKARA

+90 (312) 289 92 22 - yte.bilgi@tubitak.gov.tr

TÜBİTAK - BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü (YTE)