**RABBITMQ**

MessageQueue ve MessageBroker bu kısımdaki önemli kavramlardandır .

**MessageQueue**

Yazılım sistemlerinde kullanılan bir yapıdır.

Birbirinden bağımsız sistemlerde veri alışverişi yapmak için kullanılmaktadır . İki farklı yapı ya da iki farklı dilde yazılmış kod olabilir .

Temelde gönderilen mesajları bu kuyrukta saklayıp sonradan bu mesajların işlenmesini sağlamaktadır.

Kuyruğa mesaj gönderene Producer(Publisher)(Yayıncı) denirken kuyruktaki mesajları işleyene ise Consumer(Tüketici) denir.Kuyruktaki mesajların olduğu kısmada MessageQueue denmektedir.

Message queue yapısı mimaride asenkron birdavranış sergilenmesini sağlamaktadır.

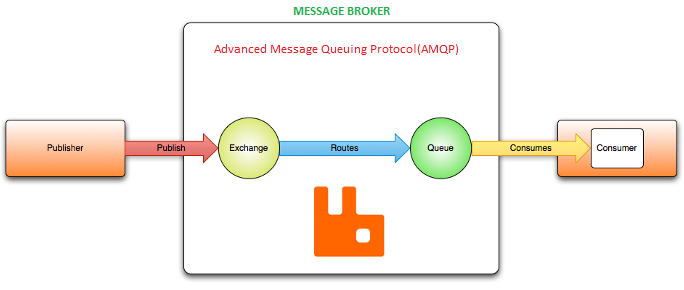
En çok kullanılan örnekleri e ticaret durumlarında kullanılmaktadır

**MessageBroker**

Message Queue yapısını barındaran ve bu queue üzerinden publisher ve consumer arası iletişimi sağlayan genel sistemin adıdır.

Birden fazla MessageQueue bulunabilmektedir

**RabbitMq**

OpenSource yapıdadır.

Farklı işletim sistemlerinde cross platform olduğu için kullanılabilir.

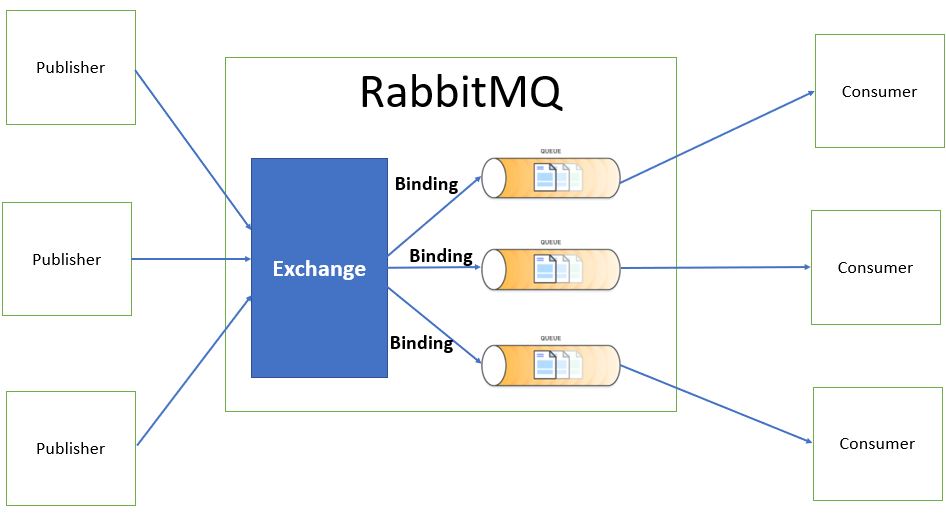
Cloud hizmetide mevcutdur. Genel olarak kullanıı ücretsizdir

**RabbitMq kurulumu ve kullanımı**

Dockerize ederek ve Cloud yapısı kullanarak iki farklı şekilde kullanılabilir

**Exchange**

Publisher dan gelen verilerin nasıl yönetileceğini ve hangi route lara yönlendirileceği konusunda control sağlar

Birden fazla kuyrukta hangisine gönderileceğini exchange üzerinden belirlenecektir

Dört farklı exchange türü mevcuttur

**Route**

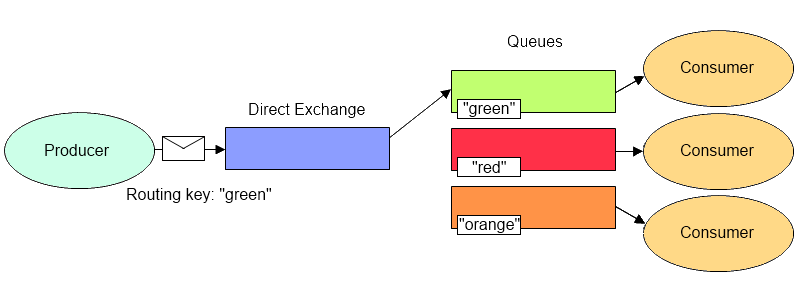
Mesajların exchange üzerinden kuyruklara nasıl gönderileceğini belirleyen yapıdır

Exchange de bulunana routing key değerleri kullanılmaktadır

**Binding**

Exchange ile kuyruklar arasındaki olan bağlantıya verilen isimdir

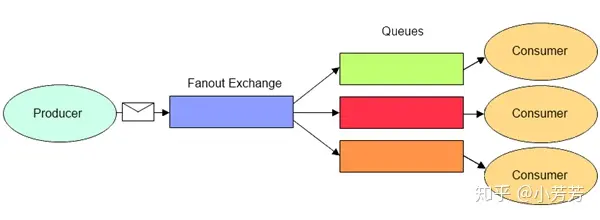
**Direct Exchange (Exchange Types)**

En sadesi denilebilir ve Default olarak gelendir

Direct olarak belirli kuyruğa gönderilmesini sağlayan tip tir

Mesaj routing key e uygun kuyruklara gönderilir

Genellikle hata mesajlarının işlendiği seneryolarda kullanılabilir

**Fanout Exchange(Exchange Types)**

Exchange tarafına bind oluşturulmuş tüm kuyruklara gönderim yapılmasını sağlar.

Publisher kuyruk isimlerini önemsemeden direk gönderim sağlamaktadır

Tüm servislere ortak bildiriminde bulunmak için kullanılabilir



**Topic Exchange(Exchange Types)**

Routing key in bir kısmına yapısına formatına göre kuyruklara mesaj gönderilir

Kuyruklar sadece ilgili routing key e göre consume edilecektir

Log sistemi seneryolarında kullanılmaktadır

**Header Exchange(Exchange Types)**

Routing key yerine header ları kullanarak yönlendirme yapmaktadır

Topic deki tüm seneryolarak kullanılabilmektedir

**Gelişmiş Kuyruk Mimarisi**

RabbitMq teknolojisinin genel olarak işleri-kaynakları-görevleri ölçeklendirilebilecek ve asenkron bir şekilde hazırlanmasını sağlayan yapılardır

Mesajların kalıcılığını göre farklı seneryolar farklı durumlar oluşacaktır, bunların configure edilmesini sağlayan yapılardır

**Round-Robin Dispatching(Sıralı Gönderim)**

RabitMq default olarak tüm consumer lara mesajın sırayla döngüsel olarak mesaj gönderilmesi durumudur

**Message Acknowledgment(Mesaj Onaylama)**

Default olarak consumer tarafına gönderilen mesajı ister başarılı olsun ister olmasın hemen kuyruktan silinmesi üzerine işaretleme yapmaktadır

E ticaret anlamında örneklendirirsek bu mesajlar için siparişler gerçekten consumer tarafından alındığına ait bir eminlik oluşmalı ki bu böyle silinmelidir aksi halde siparişler consume edilemediği için sorunlar oluşmaktadır

RabbitMq tarafında consumer edilen veri başarılı olduğuna dair kuyruğa bir veri iletilmeliki bu zaman kuyruktan bu mesaj silinmeli böylece veri kaybı önlenmiş olucaktır

Ayrıca Message Acknowledgment özelliği kullanılıyorsa bu kısımda , RabbitMq tarafına bu verinin silinmesi konusunda haber verilmesse , başka bir consumer tarafından tekrar tüketilecektir ve veri tutarsızlığı oluşacaktır .Ayrıca mesajlar onaylanarak silinmesse kuyruğun yoğunlaşması ve şişmesine sebeb olucaktır

Geri bildirim default için RabbitMq da 30 dk dır, Eğer bu süre dolar hala biro nay bildirimi gelmesse RabbitMq mesajı tekrar yayınlamaya devam edicektir

**Message Durability(Mesaj Dayanıklılığı/Sürekliliği)**

Üst tarafta consumer da sıkıntı yaşaması durumu ile iligli sıkıntıların giderilmesine bakmıştık , burada ise RabbitMq sunucu tarafında bir sorun olursa nasıl yol izlenmeli burada bu kısma bakılacaktır

Normalde default tarafta , tüm kuyruk ve mesajlar silinecektir . Kuyruk ve mesajların kalıcı olması için bazı configure yapıların oluşturulması gerekmetedir

**Fair Dispatch (Adil Dağıtım)**

RabbitMq için tüm consumerlara eşit şekilde mesajları ilete bilmekteyizEşit şekilde mesajların iletilmesi sistemdeki performansı düzenli hale getirecektir , Böylece consumerlar arası fazla yük az yük dengesizliğide çözülmüş olucaktır

**Mesaj İşleme Konfigürasyonu(Mesaj İşleme Hızı ve Teslimat ayarları)**

Bunun FairDispatch özelliğinin configure edilmesi için kullanımı olucaktır ve **BasicQos** metodu ile bu yapılandırma sağlanabilmektedir, Ölçelenidrilebilmek için yapılmış bir durumdur

**Mesaj Tasarımları**

Aslında design pattern gibi devamlı tekrar eden olayların kavramsallaşması durumunda kullanılmakatadır .

**P2P(Point-to-Point) Tasarımı**

Seneryo gereği bir mesajın tüketici tarafından işlenmesi gerekiyorsa bu yaklaşım kullanılır.Publisher mesajı kuyruğa gönderir consumer kuyruktaki mesajı consume edicektir.

**Publish/Subscribe(Pub/Sub) Tasarımı**

Seneto gereği bir mesajın bir çok tüketici tarafından işlenmesi gerektiğinde kullanılır. Publisher mesajı exchange e gönderir , mesaj ise exchange bind edilmiş olan tüm kuyruklara consume edilmektedir.

**Work/Queue(İş Kuyruğu) Tasarımı**

Seneryo gereği publisher tarafından yayınlanan mesajın birden fazla consumer arasında yalnızca biri tarafından tüketilmesi amaçlanmaktadır , böylece consumer lar eşit iş yüküne sahip olucaktır . Work queue tasarımı parallel işleme ihtiyacı duyuan seneryolar için oldukça uygundur.

**Request/Response Tasarımı**

Bu seneryoda ise publisher request yapar gibi kuyruğa mesaj gönderir ve bu mesajı tüketen consumer dan sonuca dair kuyruktan bir response bekler. Hem publisher hem consumer seneryo gereği Request ve Response verilerinin taşınması sırasında iki karekter de olucaktır . (Bundan dolayı hem publisher hem de consumer için iki karakterinde oluşturulacağı yapı tanımlanmaktadır )

Tüm yapılan uygulamalrda ilgili kısımda cmd yi açıp seneryoya göre bir ya da birden fazla reciver oluşturarak dotnet run komutuyla testlerini yapılabilecektir

**Enterprice Service Bus(ESB)**

Servisler arası entegrasyon sağlayan companentlerin bütünüdür , farklı yazılım setlerinin birbiri ile iletişimini sağlayan bir mimari ve tool dur ESB yapılanması bugün RabitMq ile tasarlanan bu yapı yarın kafkaya geçmek istediğinde değiştirmesi konusunda da bütünlük sağlayarak geçişi kolaylaştıran bir yapıdır ,entegrasyon sırasında bir den fazla durumu abstract yapılandırması haline getirin durumdur denebilir temelde .

ESB nin kullanımında en popular kütüphane mass transit kütüphanesidir.

**Mass Transit**

.Net için geliştirilmiş olan ve dağıtık(distributed) uygulamaları rahatlıkla yönetmeyi sağlayan esb frameworküdür , farklı uygulamalar arasında message-based-comminucation sağlayan , bir transport gateway dir , özellikle microservce mimarilerinde distiributed sistemlerin oluşturulması ve haberleşmesi sürecinde ,teknolojilere olan bağlılığı soyutlamak için kullanılan bir kütüphanedir . Message Broker bağımlılığını ortadan kaldırmak için de ayrıca tercih edilebilir (Publish ve send metotları kullanılabilir kodda daha detaylı olarak açıklaması yapılmıştır )

**Transport Gateway**

Farklı sistemler arasında , farklı iletişim protokollerini kullanarak iletişim kurmayı sağlayan araçtır, bu araç iletişim protokollerinde farklılıkları elimine ederek sorunsuz bir şekilde birlikte çalışabilmelerini sağlamaktadir.