1)

Senkron ve asenkron iletişim, bilgi alışverişinde temel işbiriği biçimlerini tanımlar.

**Senkron:**

Bu iletişim türünde tarafların zamanlaması ve hızları bellidir, alıcı ve verici arasında tam bir eşzamanlılık vardır, gecikme(ping) önemsiz derecede çok küçüktür.

Örnek olarak telefon konuşmaları, gerçek zamanlı iha drone görüntüleri ve anlık mesaj uygulamaları buna örnektir.

**Asenkron:**

Senkrona göre alıcı ve verici arasında zaman farkının olması veya anlık iletişimden bağımsız mesaj yapılarıdır. Atılan emailin okunması veya atılan sms mesajları buna örnektir.

**Özet:**

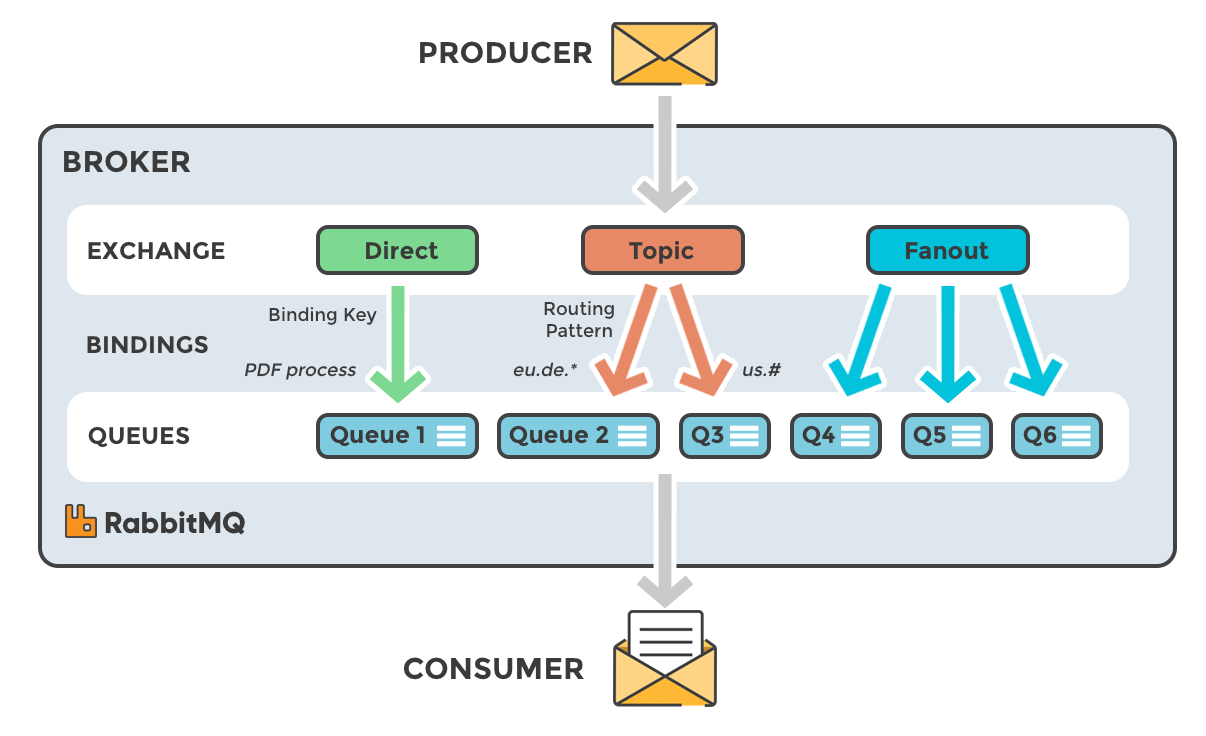
Senkron ve asenkron iletişim, iletişim gereksinimlerine ve ortamına bağlı olarak tercih edilir. Hızlı ve doğrudan tepkiler gerektiren durumlarda senkron iletişim, zaman farkının veya hızın önemli olmadığı durumlarda ise asenkron iletişim tercih edilir.

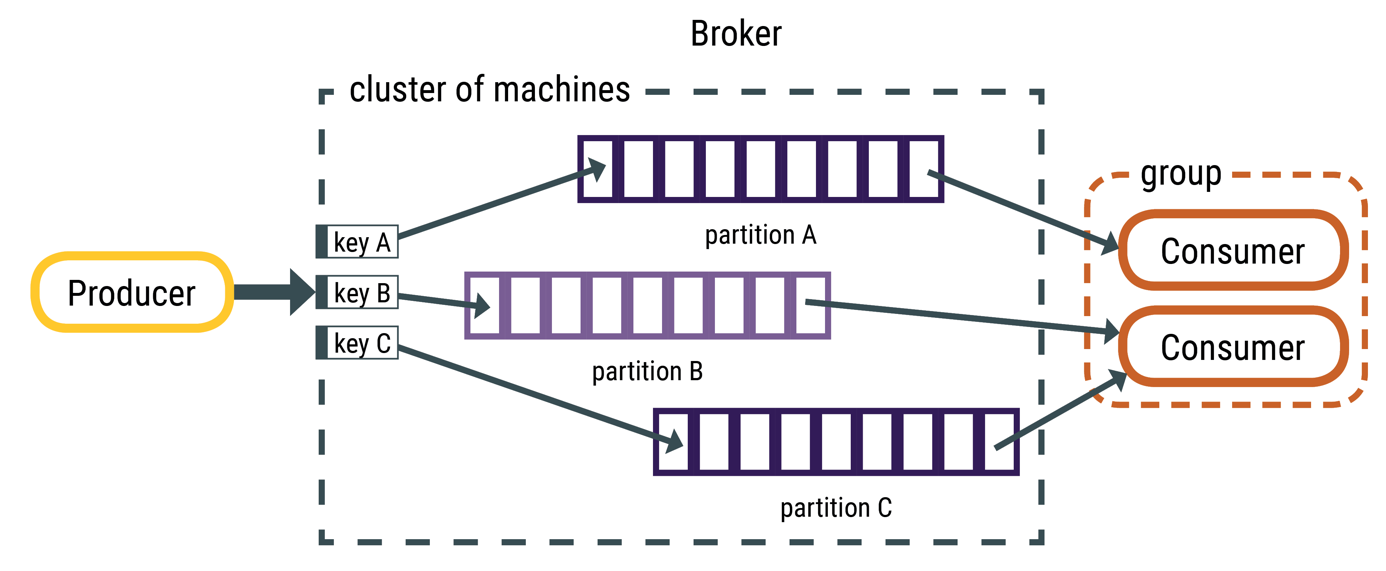
2)

RabbitMQ, mesaj kuyrukları üzerinden geleneksel mesajlaşma modelini benimseyen bir sistemdir. Mesajlar üreticiden kuyruğa gönderilir ve tüketiciye iletilir. Genellikle dağıtık sistemler arasında mesajlaşma için tercih edilir.

Kafka ise yayın-abonelik modelini kullanan bir platformdur. Veriler "konular" aracılığıyla yayınlanır ve aboneler bu konulara abone olarak verileri alırlar. Yüksek performans ve ölçeklenebilirlik sağlar ve genellikle gerçek zamanlı veri akışı ve büyük veri analitiği gibi uygulamalarda tercih edilir. Büyük veri kütlelerinin işlenmesi ve analizi için kullanılır.

Bu nedenle, RabbitMQ genellikle geleneksel mesajlaşma senaryoları için tercih edilirken, Kafka daha çok büyük ölçekli uygulamalar için uygundur ve performansıyla öne çıkar.





3)

Docker ve Sanal Makine (Virtual Machine - VM), yazılım uygulamalarını farklı ortamlarda çalıştırmak için kullanılan teknolojilerdir.

Docker, uygulamaları hafif ve taşınabilir konteynerlere paketleyerek çalıştırmayı sağlayan bir platformdur. Her konteyner, uygulamanın gereksinim duyduğu tüm bağımlılıkları ve kaynakları içerir. Bu sayede uygulamalar farklı ortamlarda aynı şekilde çalışabilir ve dağıtılabilir hale gelir.

Sanal Makine ise, fiziksel bir bilgisayarda birden fazla sanal işletim sistemi çalıştırmayı sağlayan bir teknolojidir. Her sanal makine, kendi işletim sistemine ve kaynaklara sahiptir. Bu sayede farklı işletim sistemleri ve uygulamalar aynı fiziksel makinede birbirlerinden izole olarak çalışabilir.

Dolayısıyla, Docker uygulamaları hafif ve taşınabilir konteynerlerde çalıştırırken, Sanal Makine fiziksel bir makinede birden fazla izole ortam oluşturarak farklı işletim sistemleri ve uygulamaları çalıştırır.

VM örnek 🡪 Virtualbox bu uygulamada sanal makine kurup burda işlemler yapabiliriz.

4)

ekran görüntüsü, yazılım, metin, bilgisayar simgesi içeren bir resim

Açıklama otomatik olarak oluşturuldu

5)

docker pull: Docker Hub veya başka bir Docker Registry'den bir görüntüyü (image) çekmek için kullanılır.

Örnek: docker pull ubuntu:latest

docker build: Bir Docker dosyasından (Dockerfile) Docker görüntüsü oluşturmak için kullanılır.

Örnek: docker build -t myapp .

docker run: Bir Docker görüntüsünden bir konteyner başlatmak için kullanılır.

Örnek: docker run -d -p 8080:80 myapp

docker ps: Çalışan konteynerleri listelemek için kullanılır.

Örnek: docker ps

docker stop: Çalışan bir konteyneri durdurmak için kullanılır.

Örnek: docker stop mycontainer

docker rm: Bir konteyneri silmek için kullanılır.

Örnek: docker rm mycontainer

docker rmi: Bir Docker görüntüsünü silmek için kullanılır.

Örnek: docker rmi myimage

docker exec: Bir çalışan konteyner içinde komut çalıştırmak için kullanılır.

Örnek: docker exec -it mycontainer bash

(<https://docs.docker.com/reference/cli/docker/>)

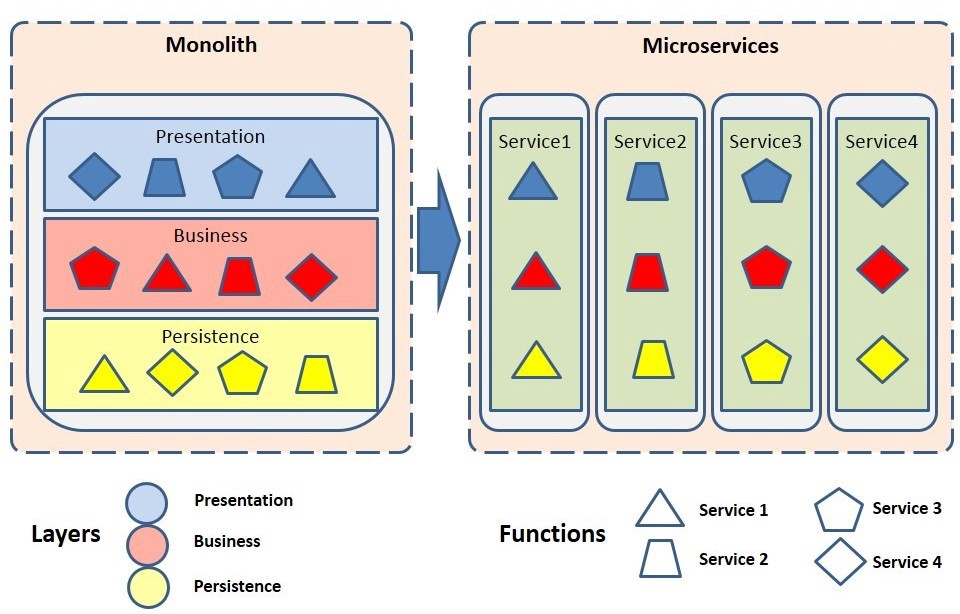
6)

Monolith mimarisi, tüm uygulama bileşenlerinin tek bir yapı içinde birleştirildiği ve birlikte çalıştığı bir yapıdır. Bu yaklaşım genellikle başlangıç aşamasındaki uygulamalar için tercih edilir. Tüm kod tabanı tek bir proje içinde bulunduğu için, başlangıçta geliştirme ve dağıtım kolay olabilir. Ancak zamanla, uygulamanın büyümesi ve karmaşıklığının artmasıyla bakımı zorlaşabilir ve geliştirme süreçleri yavaşlayabilir.

(presentation 🡪 Business 🡪 Persistence)

Microservice mimarisi ise, uygulamanın farklı işlevlerini küçük, bağımsız ve ölçeklenebilir hizmetlere bölen bir yaklaşımdır. Her hizmet kendi bağımsız kod tabanına ve veritabanına sahiptir ve bu hizmetler ayrı ayrı geliştirilebilir, dağıtılabilir ve ölçeklenebilir. Bu yaklaşım, büyük ve karmaşık uygulamaların geliştirilmesi ve yönetilmesi için daha esnek bir yapı sağlar. Ancak, mikro hizmetler arasındaki iletişim ve veri bütünlüğünü sağlama gibi bazı zorlukları da beraberinde getirebilir.

Sonuç olarak, Monolith mimarisi başlangıçta basitlik ve hız sağlayabilirken, Microservice mimarisi daha büyük ve karmaşık uygulamalar için daha esneklik ve ölçeklenebilirlik sunar. Hangi mimarinin tercih edileceği, uygulamanın gereksinimlerine, büyüklüğüne ve ekibin tecrübesine bağlı olarak değişebilir.



7)

**API Gateway:** API Gateway, bir mikro servis mimarisinde gelen istekleri yönetmek için kullanılan bir ara yazılımdır. Gelen istekleri alır, doğrular, yetkilendirir ve gerektiğinde ilgili mikro servise yönlendirir. Ayrıca, istek ve cevapları transform ederek farklı formatlarda iletişim sağlar.

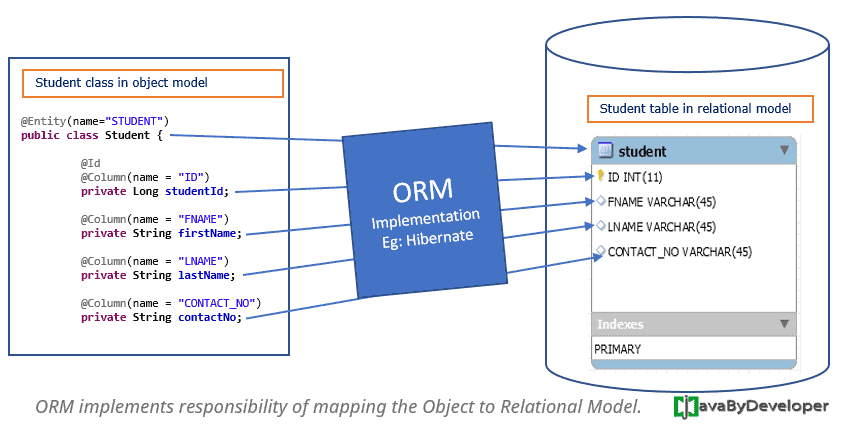
**Service Discovery**: Service Discovery, bir ağdaki servislerin dinamik olarak keşfedilmesini ve yönetilmesini sağlayan bir sistemdir. Mikro servis mimarisinde, servislerin IP adresleri ve port numaraları sıklıkla değişebilir. Bu nedenle, diğer servislerin bu değişiklikleri otomatik olarak algılayıp buna göre iletişim kurması gerekir. Service Discovery bu işlevi yerine getirir.

**Load Balancer:** Load Balancer, gelen istekleri farklı sunucular arasında dağıtarak yükü dengeler. Özellikle yüksek trafikli uygulamalarda kullanılır ve performansı artırır, sunucu yükünü dengeleyerek tek bir sunucunun aşırı yüklenmesini önler. Load Balancer ayrıca, hizmet sürekliliğini sağlamak için sunucu arızalarını algılar ve etkilenen istekleri başka sunuculara yönlendirir.

**8)**

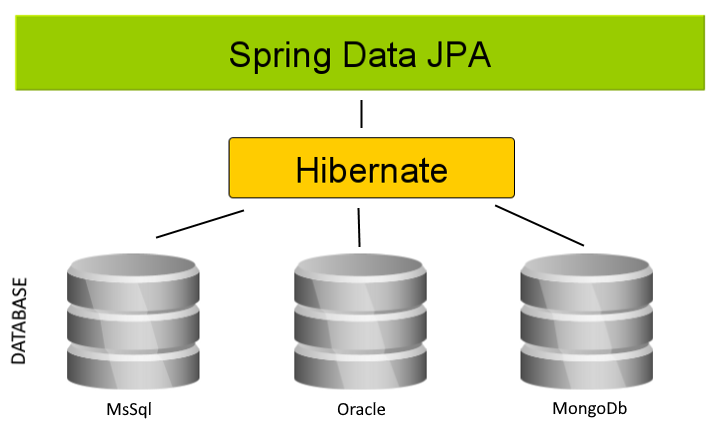
**Hibernate:**

Hibernate, Java programlama dilinde nesne ilişkisel eşleme (ORM - Object-Relational Mapping) işlevselliği sağlayan açık kaynaklı bir framework'tür. Bu framework, veritabanı tablolarını Java sınıflarıyla eşleştirmeyi kolaylaştırır. Örneğin, Hibernate kullanarak Java sınıfları oluşturabilir ve bu sınıfları veritabanı tablolarıyla ilişkilendirebilirsiniz. Hibernate, veritabanı işlemlerini gerçekleştirmek için SQL sorgularını otomatik olarak oluşturur ve yönetir, bu da geliştiricilerin veritabanıyla etkileşimini kolaylaştırır.



**JPA (Java Persistence API):**

JPA, Java EE (Enterprise Edition) için bir standart olan ve ORM işlevselliği sağlayan bir API'dir. JPA, veritabanı işlemlerini gerçekleştirmek için kullanılan Hibernate gibi ORM framework'lerinin altında çalışır. JPA, Java sınıflarını ve veritabanı tablolarını eşleştirmek için anotasyonlar sağlar ve geliştiricilere veritabanı işlemlerini daha kolay bir şekilde yönetme imkanı tanır. JPA'nın kullanımı, farklı ORM framework'leri arasında geçiş yapmayı kolaylaştırır ve uygulamaların taşınabilirliğini artırır.



**Spring Data:**

Spring Data, Spring Framework'ün bir parçası olan ve veritabanı işlemlerini gerçekleştirmek için kullanılan bir alt projedir. Spring Data, çeşitli veritabanı teknolojileri için genel ve tutarlı bir erişim katmanı sağlar.