1-Senkron İletişim:Katılımcıların aynı anda gerçek zamanlı olarak iletişim kurduğu bir iletişim yöntemidir.Bu yöntemle katılımcılardan bir taraf veri gönderirken,diğer taraf ise bu veriyi anında alır ve işler.

Senkron iletişime örnek olarak telefon görüşmeleri,video ile yapılan anlık sohbetler veya uygulamalar aracılığı ile yapılan mesajlar örnek olarak verilebilir.

Asenkron iletişimKatılımcıların farklı zamanlarda iletişim kurduğu bir yöntemdir.Yani bir taraf veriyi gönderirken,diğer taraf bu veriyi hemen almak zorunda değildir.Gönderen taraf veriyi gönderir ve işlemi tamamlar,alıcı olan diğer taraf ise veriyi eşzamanlı olarak alıp işlemek zorunda değildir.

Asenkron iletişime örnek olarak sosyal medyada yapılan yorumlar,E-posta göndermek çünkü E-posta ile mesajlar anlık olarak gitmez ve internette bulunan forumlar örnek olarak verilebilir.

2-RabbitMQ;Yazılım sistemlerinin birbirleriyle iletişim kurmasını,veri alışverişi yapmasını ve iş akışlarını kordine etmesini sağlayan bir mesajlaşma ve akış yöneticisidir.

Kafka:Akış analitiği,veri entegrasyonu ve kritik uygulamalar için binlerce şirket tarafından kullanılan açık kaynaklı bir dağıtık etkinlik akışı platformudur.

Farklılıkları:

Performans;

RabbitMQ küçük ve orta ölçekli uygulamalr için daha uygundur.Daha düşük gecikme süresi vardır.

Kafka büyük ölçekli uygulamalar için daha uygun.Daha yüksek veri akışı işlemede daha iyi performans verir.

Mesaj kaybı:

RabbitMQ’da mesaj kaybı daha sık görülebilir.

Kafka mesaj kaybına karşı daha dayanıklıdır.

Geliştirme ortamları:

RabbitMQ web tabanlı bir arayüze sahiptir..

Kafka geliştirme ve test amacıyla geleneksel Unix tarzı komut satırı araçları sağlar.

Cursor’ın queue’da nerede kaldığını broker’ın belirlenmesi (broker-centric) isteniyorsa RabbitMQ tercih edilebilir.

Cursor’ın queue ‘da nerede kalındığını consumer’ın bilmesini ve yönetmesini (consumer-centric) isteniyorsa Kafka tercih edilebilir.

3-Docker: Docker open source bir ‘container’ teknolojisidir. Docker, aynı işletim sistemi üzerinde, yüzlerce hatta binlerce birbirinden izole ve bağımsız containerlar sayesinde sanallaştırma sağlayan bir teknolojidir. Her bir Docker konteyneri, izole bir çevre sağlar ve kendi dosya sistemi ve ağ bağlantılarına sahiptir

Docker'in temel bileşenleri şunlardır:

Docker Image: Docker Image, bir uygulamanın çalıştırılabilir durumunu tanımlayan ve uygulamayı çalıştırmak için gereken tüm dosyaları, bağımlılıkları ve yapılandırmaları içeren bir pakettir.

Docker Engine: Docker Engine, Docker konteynerlerini oluşturmak, çalıştırmak ve yönetmek için gereken temel çalışma zamanı bileşenidir.

Docker Container: Docker Container, bir Docker Image'ın çalışan bir örneğidir. Her bir Docker Container, izole bir çalışma ortamına sahiptir ve kendi dosya sistemi, ağ bağlantıları ve süreçleri vardır. Docker Container'lar, hafif, taşınabilir ve tutarlı bir şekilde çalışan uygulamaları sağlar.

Docker Compose: Docker Compose, birden çok Docker konteynerini bir araya getirerek çoklu konteyner uygulamalarının tanımlanmasını ve yönetilmesini sağlayan bir araçtır. Docker Compose, YAML formatında bir dosya kullanarak konteynerler arasındaki bağımlılıkları ve yapılandırmaları tanımlar.

Virtual Machine: Sanal Makine, fiziksel bir bilgisayar üzerinde sanal bir ortam oluşturarak, birden fazla işletim sistemi çalıştırmak için kullanılan bir sanallaştırma teknolojisidir. Her bir sanal makine, kendi işletim sistemine, belleğe, disk alanına ve diğer sistem kaynaklarına sahiptir.

5-Docker Komutları

Temel Docker Komutları:

docker pull: Bir Docker imajını Docker Hub'dan indirir.

docker pull nginx

docker run: Bir Docker imajını çalıştırır.

docker run nginx

docker stop: Çalışan bir konteyneri durdurur.

docker stop <container\_id>

docker rm: Bir konteyneri siler.

docker rm <container\_id>

docker ps: Çalışan konteynerlerin listesini gösterir.

docker ps

docker images: Yerel depoda bulunan Docker imajlarının listesini gösterir.

docker images

docker build: Bir Dockerfile kullanarak bir Docker imajı oluşturur.

docker build -t my-image .

6-Monolithic mimaride, bir uygulama genellikle tek bir kod tabanında, tek bir veritabanında ve tek bir kullanıcı arayüzünde bulunur. Tüm uygulama modülleri birbirine bağlıdır ve genellikle aynı teknoloji yığını ve dil kullanılarak geliştirilir.

1. Mimarilerin Yapısı:

Monolithic: Tüm uygulama tek bir bütün olarak geliştirilir ve dağıtılır. Tek bir kod tabanı, tek bir veritabanı ve tek bir kullanıcı arayüzü vardır.

Microservice: Uygulama, küçük ve bağımsız hizmetlere bölünür. Her hizmet, belirli bir işlevi gerçekleştirir ve kendi veritabanına, kod tabanına ve hatta kullanıcı arayüzüne sahip olabilir.

2. Bağımlılıklar:

Monolithic: Modüller birbirine sıkıca bağlıdır. Bir modülde yapılan bir değişiklik, diğer modülleri etkileyebilir ve tüm uygulamanın yeniden dağıtılmasını gerektirebilir.

Microservice: Hizmetler birbirinden bağımsızdır. Bir hizmette yapılan bir değişiklik, diğer hizmetleri etkilemez ve sadece ilgili hizmetin yeniden dağıtılmasını gerektirir.

3. Teknoloji Yığını ve Dil Çeşitliliği:

Monolithic: Genellikle, tüm uygulama aynı teknoloji yığını ve dil kullanılarak geliştirilir.

Microservice: Her hizmet, kendi teknoloji yığınına ve diline sahip olabilir. Bu, farklı hizmetlerin farklı teknolojileri ve dilleri kullanmasına izin verir.

4. Ölçeklenebilirlik:

Monolithic: Tüm uygulama tek bir bütün olarak ölçeklenir. Yani, tüm uygulama aynı anda ölçeklendirilir.

Microservice: Her hizmet bağımsız olarak ölçeklenebilir. Bu, yalnızca belirli hizmetlerin ihtiyaç duyulduğunda ölçeklendirilebileceği anlamına gelir.

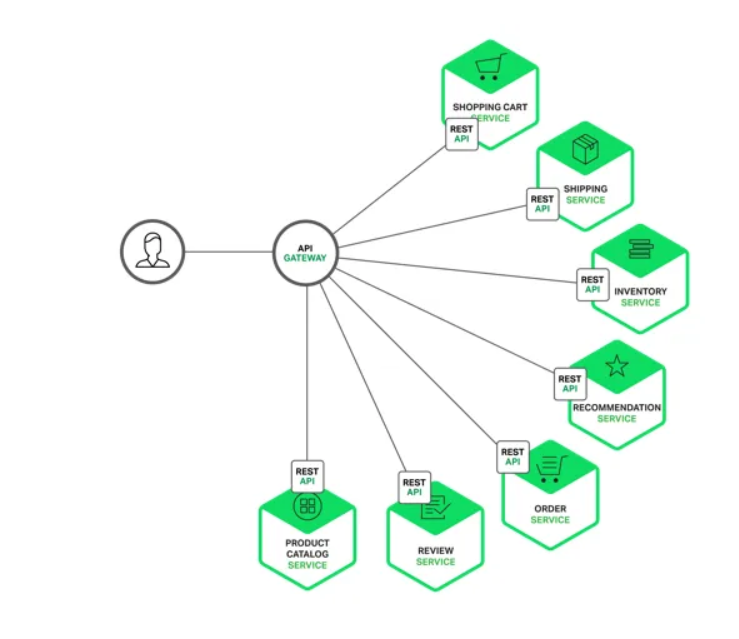
5. Geliştirme ve Dağıtım Süreçleri:

Monolithic: Tüm geliştirme, test ve dağıtım süreçleri tek bir uygulama içinde gerçekleşir.

Microservice: Her hizmet, bağımsız olarak geliştirilir, test edilir ve dağıtılır. Bu, paralel geliştirme ve dağıtım süreçlerini mümkün kılar.

7-API Gateway (ağ geçidi), sisteme tek giriş noktası olan bir sunucudur. Bir apinin (Uygulama Programlama Arayüzü- kullanıcıların kullandığı uygulama yazılımları) önünde bulunan uygulamaya tek bir giriş noktası sağlar. Bu sayede ağ geçidi, koruyucu görevi görerek güvenliği sağlar.

API ağ geçidinin bir mimariye nasıl uyduğunu gösteren şema



Service Discovery

Service Discovery, bir mikroservis mimarisindeki hizmetlerin dinamik olarak bulunmasını ve erişilmesini sağlayan bir süreç veya mekanizmadır. Bu, uygulama içindeki farklı hizmetlerin IP adreslerini veya konumlarını manuel olarak ayarlamak yerine, otomatik olarak bulunmasını ve erişilmesini sağlar. Service Discovery, hizmetlerin dinamik olarak ölçeklendirilebilmesi ve yüksek kullanılabilirlik sağlanabilmesi için önemlidir.

Load Balancer

Load Balancer, yük dengeleyici veya yük dengeleyici olarak da bilinir. Bu, bir ağ üzerindeki trafik yükünü dağıtmak için kullanılan bir cihaz veya yazılımdır. Genellikle sunucu grubu veya hizmetler arasında gelen istekleri dengeler ve bu sayede her bir sunucunun veya hizmetin aşırı yük altında kalmasını önler.

8-Hibernate: Hibernate, Java tabanlı bir ORM (Object-Relational Mapping) framework'üdür. Veritabanı işlemlerini gerçekleştirmek için nesne yönelimli bir yaklaşım sunar, böylece veritabanı tablolarını Java sınıflarıyla eşleştirir. Bu, geliştiricilere SQL sorguları yazmak yerine doğrudan Java nesneleri üzerinde işlem yapma imkanı sağlar. Örneğin:

// Bir Hibernate Entity sınıfı örneği

@Entity

@Table(name = "users")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(name = "username")

private String username;

@Column(name = "email")

private String email;

// Getter ve Setter metotları

}

Yukarıdaki örnek, "users" tablosunu temsil eden bir Hibernate Entity sınıfıdır.

JPA (Java Persistence API): JPA, Java tabanlı bir ORM standardıdır ve Hibernate gibi ORM framework'lerinin temelini oluşturur. JPA, Java EE (Enterprise Edition) standartlarından biridir ve veritabanı işlemlerini gerçekleştirmek için bir API sağlar. JPA, Hibernate gibi ORM framework'lerinin altında kullanılabileceği gibi, diğer ORM framework'leriyle de uyumlu olarak çalışabilir. Örneğin:

// JPA Entity sınıfı örneği

@Entity

@Table(name = "products")

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(name = "name")

private String name;

@Column(name = "price")

private BigDecimal price;

// Getter ve Setter metotları

}

Yukarıdaki örnek, "products" tablosunu temsil eden bir JPA Entity sınıfıdır.

Spring Data: Spring Data, Spring Framework'ün bir parçası olarak geliştirilmiş bir veri erişim katmanı framework'üdür. Spring Data, JPA, MongoDB, Redis, Cassandra gibi çeşitli veri depolama teknolojileri için genel veri erişim operasyonlarını destekler ve bu operasyonları kolaylaştırır. Örneğin:

// Spring Data repository örneği

public interface ProductRepository extends JpaRepository<Product, Long> {

List<Product> findByName(String name);

}

Yukarıdaki örnek, "Product" entity sınıfı için CRUD işlemlerini gerçekleştiren bir Spring Data repository interface'ini temsil eder. Bu interface, JpaRepository arayüzünden türetilmiştir ve temel CRUD işlemlerinin yanı sıra özel sorguları destekler.