2.Hafta Ödevi

Senkron ve Asenkton iletişim:

- **Senkron İletişim:**

Senkron iletişimde, bir işlemin tamamlanması diğer işlemlerin beklemesine neden olur. Yani, bir işlem başladığında, işlem tamamlanana kadar diğer işlemler bekler. Senkron iletişimde işlemler sırayla ve birbirini bekleyerek gerçekleşir.

- **Asenkron İletişim:**

Asenkron iletişimde ise, bir işlem başlatıldıktan sonra beklemeye gerek kalmadan diğer işlemler devam edebilir. Yani, bir işlem başlatıldığında, sonucunu beklemek yerine diğer işlemlere devam edilir. İşlem tamamlandığında sonucu elde etmek için özel bir yöntem kullanılır.

Java Spring'de, özellikle asenkron iletişim oldukça önemlidir çünkü bu, uygulama performansını artırmak ve kullanıcı deneyimini iyileştirmek için kullanılır. Örneğin, bir web uygulamasında bir kullanıcının bir dosyayı yüklemesi uzun sürebilir. Senkron bir yaklaşımla, kullanıcı dosyanın yüklenmesini beklerken hiçbir şey yapamaz. Ancak asenkron bir yaklaşımla, kullanıcı dosyayı yükledikten sonra diğer işlemlerine devam edebilir ve dosyanın yüklenme sürecini arka planda işleyebilirsiniz.

RabbitMQ ve Kafka arasındaki farkları araştırın?

- **Mesaj Sıralama Modeli**:

RabbitMQ: RabbitMQ, tipik bir mesaj sıralama (message queuing) sistemidir. Yani, bir mesaj yayınladığınızda, bu mesaj bir kuyruğa eklenir ve aboneler bu kuyruğu dinleyerek mesajları alırlar.
Kafka: Kafka ise bir olay akışı (event streaming) platformudur. Kafka, verileri akış halinde işlemek ve depolamak için tasarlanmıştır. Mesajlar kuyruklara eklenmek yerine, bir "log" mantığıyla sıralı bir şekilde depolanır ve tüm abonelere eş zamanlı olarak iletilir.

- **Veri Saklama**:

RabbitMQ mesajları kuyruklarda saklar. Bir abone mesajı işlediğinde, RabbitMQ bu mesajı kuyruktan kaldırır.
Kafka: Kafka, mesajları bir "log" olarak adlandırılan özel bir veri yapısında saklar. Mesajlar belli bir süre veya boyuta kadar tutulabilir ve abonelerin istediği zaman bu logu okumasına izin verir.

- **Senkronluk ve Asenkronluk**:

RabbitMQ: RabbitMQ, senkron veya asenkron kullanılabilir. Mesaj yayınladığınızda, beklemek istiyorsanız senkron bir şekilde bekleyebilir veya mesaj yayınladıktan sonra devam edebilirsiniz.
Kafka: Kafka, genellikle asenkron bir şekilde kullanılır. Mesajlar hemen işlenmez, bunun yerine Kafka, mesajları loga ekler ve abonelerin bu logu istedikleri zaman okumalarına izin verir. Bu nedenle, Kafka'nın doğası gereği asenkron bir modeldir.

Docker ve Virtual Machine nedir:

- **Virtual Machine (Sanal Makine):**

Sanal makine, fiziksel bir bilgisayarda yazılım tabanlı bir ortam oluşturarak, birden çok sanal işletim sistemi çalıştırmanıza olanak tanır. Bu sanal işletim sistemleri, fiziksel donanımın üzerinde yüklü bir işletim sistemi gibi davranır. Her sanal makine kendi sanal belleği, sanal işlemcisi, sanal sabit diski ve diğer sanal cihazlarına sahiptir.

Sanal makineler genellikle hipervizör adı verilen bir yazılım katmanı aracılığıyla çalışır. Bu hipervizör, fiziksel donanım üzerinde sanal makinelere erişimi yönetir ve kaynakları paylaştırır. Örnek olarak, popüler hipervizörler arasında VMware, VirtualBox ve Microsoft Hyper-V bulunmaktadır.

Sanal makinelerin avantajları arasında farklı işletim sistemlerini aynı fiziksel donanım üzerinde çalıştırma yeteneği, izolasyon ve güvenlik sağlama, çeşitli iş yükleri için esneklik ve taşınabilirlik sayılabilir. Ancak, her bir sanal makine için ayrı bellek, işlemci ve depolama alanı tahsis edilmesi gerektiğinden kaynak tüketimi daha yüksektir.

- **Docker:**

Docker, yazılımları hafif, taşınabilir ve kapsayıcı bir şekilde paketlemeye yarayan bir platformdur. Docker, uygulamaları çalıştırmak için izole edilmiş ortamlar olan konteynerler kullanır. Konteynerler, işletim sistemini paylaşır ancak izole bir ortamda çalışır. Bu, bir Docker konteynerinin çok daha hafif ve hızlı bir şekilde başlatılmasını ve çalıştırılmasını sağlar.

Docker, konteynerlerin oluşturulması, yönetilmesi ve dağıtılmasını kolaylaştıran bir dizi araç sağlar. Docker konteynerleri, uygulamaları ve tüm bağımlılıklarını (kütüphaneler, ortam değişkenleri vb.) paketler, böylece uygulama, herhangi bir ortamda tutarlı bir şekilde çalışabilir.

Microservice ve Monotlith mimarilerini kıyaslayın:

- **Monolithic Mimariler:**

Monolithic mimariler, genellikle tek bir büyük ve tek parça halinde geliştirilmiş bir uygulamayı ifade eder.
Bir monolithic uygulama, genellikle tek bir kod tabanında, tek bir dilde yazılmış ve tek bir veritabanı kullanılarak geliştirilir.
Tüm uygulama katmanları (örneğin, kullanıcı arayüzü, iş mantığı ve veri tabanı erişimi) tek bir dağıtılabilir dosya veya paket içinde birleştirilir.
Monolithic mimariler, başlangıçta daha hızlı geliştirme ve dağıtım sağlayabilir ancak zamanla büyüdükçe, bakımı zorlaşabilir ve esneklikte kısıtlamalara neden olabilir. Ayrıca, büyük uygulamaların ölçeklenmesi ve güncellenmesi daha zor olabilir.

- **Microservices Mimarileri:**

Microservices (mikroservisler), uygulamanın farklı işlevlerini küçük, bağımsız ve kendi kendine yeterli hizmetlere bölen bir mimari yaklaşımıdır.
Her mikroservis, belirli bir işlevi yerine getiren ve genellikle bir API aracılığıyla diğer servislerle iletişim kuran birimler halinde geliştirilir.
Her bir mikroservis, farklı bir dil veya teknoloji yığını kullanılabilir, ayrıca kendi veritabanına sahip olabilir.
Mikroservis mimarileri, ölçeklenebilirlik, esneklik ve bakım kolaylığı gibi avantajlar sunar. Ayrıca, geliştirme ekibine belirli işlevler üzerinde odaklanma ve farklı teknolojileri kullanma esnekliği sağlar.

API Gateway, Service Discovery, Load Balancer kavramlarını açıklayın

- **API Gateway (API Geçidi):**

API Gateway, gelen istekleri alır ve bu istekleri doğru hedeflere yönlendirir. API Gateway genellikle bir mikro servis mimarisinde kullanılır ve gelen istekleri farklı mikro servislere yönlendirmek için bir ara katman olarak görev yapar. Ayrıca yetkilendirme, kimlik doğrulama, güvenlik, protokol dönüşümü gibi işlevleri de gerçekleştirebilir. Böylece, bir API Gateway kullanarak, uygulama geliştiricileri birden çok servise erişebilir ve bu servislerin işlevselliğini merkezi bir noktadan yönetebilirler.

- **Service Discovery (Servis Keşfi):**

Servis Keşfi, dağıtık sistemlerde bir servisin konumunu bulmayı kolaylaştıran bir yöntemdir. Mikro servis mimarisinde, birçok küçük servis birbirine bağlanır ve dinamik olarak ölçeklenebilir. Bu durumda, her servis bir IP adresi ve port numarasıyla bulunamaz. Servis Keşfi, bu sorunu çözmek için kullanılır. Bir servis, Servis Keşfi aracılığıyla kayıt yapar ve diğer servisler de bu kayıtları sorgulayarak diğer servislerin yerlerini ve IP adreslerini öğrenirler. Böylece, herhangi bir servis, diğer servislerle iletişim kurmak için statik bir yapıya ihtiyaç duymadan dinamik olarak bağlanabilir.

- **Load Balancer (Yük Dengeleyici):**

Load Balancer, gelen istekleri birden çok sunucu veya servis arasında dengeler. Büyük ölçekli web sitelerinde veya uygulamalarda kullanılan Load Balancer, trafik yoğunluğunu paylaştırarak sistem performansını artırır ve tek bir sunucunun veya servisin aşırı yük altında ezilmesini engeller. Load Balancer ayrıca yedeklilik sağlar, yani bir sunucu veya servis başarısız olursa, trafik otomatik olarak diğer sunuculara veya servislere yönlendirilir. Bu, sistemlerin yüksek erişilebilirlik ve dayanıklılık sağlamasına yardımcı olur.