Senkron ve Asenkron iletişim nedir örneklerle açıklayın? (10 PUAN)

Senkron İletişim:
Senkron iletişimde, iletişimde yer alan tüm tarafların işlemleri eşzamanlı olarak gerçekleşir. Gönderici, iletiyi gönderirken alıcı da aynı anda bu iletiyi alır. Bu iletişim şekli, gönderici ve alıcının birbirlerini beklemesini gerektirir ve belirli bir zamanlama ve senkronizasyon gerektirir. Telefon görüşmeleri ve gerçek zamanlı sohbet uygulamaları (örneğin, Skype, Zoom) senkron iletişim örnekleridir.

Asenkron İletişim:
Asenkron iletişimde, iletişimde yer alan tarafların işlemleri eşzamanlı değildir. Gönderici, iletiyi gönderir ve alıcı daha sonra bu iletiyi alır ve yanıt verir. Bu durumda, gönderici ve alıcı aynı anda etkileşimde bulunmazlar. E-posta gönderme ve alımı, metin mesajları gönderme ve sosyal medya platformlarındaki iletişim asenkron iletişim örnekleridir.

RabbitMQ ve Kafka arasındaki farkları araştırın? (10 PUAN)

İkisi de mesaj kuyruğu sistemleridir.

Farkları:

RabbitMQ: AMQP standardını uygular, tipik olarak işlem sırasında iletişim kurmak, mikroservisler arasında veri alışverişi yapmak için kullanılır.
Kafka: Bir akış işleme platformudur, yüksek hacimli veri akışı ve veri işleme için tasarlanmıştır.

RabbitMQ mesajları genellikle kısa süreli saklar, Kafka ise verileri kalıcı olarak depolar.

RabbitMQ, işlemleri genellikle en az bir kez veya tam olarak bir kez semantiğiyle garanti eder. Kafka ise genellikle en az bir kez işleme semantiği sağlar. Bu, bazı senaryolarda çift girişlere neden olabilir, ancak Kafka'nın ölçeklenebilirliği ve yüksek performansı bu durumla başa çıkabilir.

RabbitMQ, genellikle dikey olarak ölçeklendirilir. Yüksek hacimli veri akışlarına veya büyük depolama gereksinimlerine dayanıklı değildir.

Kafka, yatay olarak ölçeklenebilir, büyük ölçekli veri akışlarını kolayca yönetebilir.

Docker ve Virtual Machine nedir? (5 PUAN)

Docker
Docker uygulamaları hafif, taşınabilir ve ölçeklenebilir konteynerlarda çalıştırmak için kullanılan bir platformdur.

Docker, bir uygulamanın tüm gerekli bağımlılıklarını (kütüphaneler, ortamlar, dosyalar vb.) bir konteyner içinde paketler. Bu konteyner, işletim sistemi çekirdeğiyle birlikte çalışan ve diğer Docker

konteynerleriyle izole edilmiş bir ortam sağlar.

Docker konteynerleri, hızlı başlatma, düşük bellek tüketimi ve taşınabilirlik gibi avantajlar sağlar. Aynı uygulamayı farklı ortamlarda (geliştirme, test, üretim) aynı şekilde çalıştırmak kolaydır.

Virtual Machine
Sanal makine, bir fiziksel bilgisayarın üzerinde sanal olarak çalışan bir bilgisayar ortamı sağlar. Bu, birden fazla işletim sisteminin aynı fiziksel donanım üzerinde çalışmasına olanak tanır.

Her sanal makine, kendi işletim sistemi yüklü olan ve bağımsız bir bilgisayar gibi davranan sanal bir “host” olarak işlev görür.

Sanal makineler, tam bir işletim sistemi yükleyerek ve her biri için ayrı bellek, depolama ve işlemci kaynakları tahsis ederek çalışır. Bu, bazen kaynak israfına neden olabilir.

Docker komutlarını örneklerle açıklayın. (5 PUAN)

docker run => Docker konteynerini oluşturur ve çalıştırır.
Örnek: docker run -it --rm ubuntu:latest /bin/bash

docker ps => Çalışan Docker konteynerlerinin listesini görüntüler.

docker images => Lokaldeki Docker konteynerlerinin listesini görüntüler.

docker build => Dockerfile kullanarak yeni bir Docker image yaratır.
Örnek: docker build -t myapp .

docker pull => Remote bir docker deposundan bir image’ı lokale çeker.
Örnek: docker pull nginx:latest

docker stop => Çalışan bir docker konteynerini durdurur.
Örnek: docker stop mycontainer

docker start => Durdurulmuş bir docker konteynerini çalıştırır.
Örnek: docker start mycontainer

docker exec => Çalışan bir docker konteynerinde komut çalıştırır.
Örnek: docker exec -it mycontainer /bin/bash

docker rm => Bir docker konteynerini siler.
Örnek: docker rm mycontainer

docker rmi => Bir docker image’ını siler.
Örnek: docker rmi myimage

Microservice ve Monotlith mimarilerini kıyaslayın. (15 PUAN)

Monolith mimari, tüm uygulamanın tek bir kod tabanında yaratıldığı mimari türüdür. Tüm işlevler tek bir kod tabanında geliştirilir ve dağıtılır. Bütün bileşenler birbirine doğrudan bağımlıdır. Ölçeklenebilirliği sınırlıdır. Tek bir kod tabanı bulunduğu için geliştirme ve bakım kolaydır.

Mikroservis mimarisi, uygulamanın bir dizi bağımsız servise dağıtıldığı mimari türüdür. Her servis belirli bir işlevi temsil eder. Her servis kendi veri tabanına, kendi teknolojilerine sahip olabilir. Servisler birbirinden bağımsız olduğu için güncellenmeleri kolaydır. Her servis ayrı ayrı ölçeklendirilebilir. Çok sayıda hizmetin geliştirilmesi ve koordine edilmesi gerektiği için geliştirme hızı düşebilir.

API Gateway, Service Discovery, Load Balancer kavramlarını açıklayın. (10 PUAN)

Api Gateway: Bir uygulamanın dış dünyayla olan iletişimini yöneten bir ara katmandır. Gelen istekleri yönlendirir, doğrular, güvenlik kontrolleri uygular. Farklı servislerin sunduğu birden fazla API'yi birleştirebilir.

Service Discovery: Mikroservis mimarilerinde, hizmetlerin IP adresleri ve bağlantı noktaları değişebilir, yeni hizmetler eklenebilir veya var olanlar kaldırılabilir. Service Discovery bu değişen ortamda hizmetlerin birbirini bulmasını ve iletişim kurmasını sağlar.

Load Balancer: Gelen istekleri birden fazla sunucu ve servis arasında dağıtır. Tek bir sunucunun veya hizmetin aşırı yük altında kalmasını önler.

Hibernate, JPA, Spring Data framework'lerini örneklerle açıklayın.

Hibernate: Java tabanlı bir ORM (Object-Relational Mapping) framework'üdür. Veritabanı işlemlerini gerçekleştirmek için nesne yönelimli programlama (OOP) ile ilişkisel veritabanı arasında bir köprü görevi görür. Hibernate, OOP modeli ve ilişkisel veritabanı arasında eşleme yapar ve geliştiricilere SQL queryleri yazmak zorunda kalmadan veritabanı işlemleri yapma imkanı sağlar.

JPA: Java EE standartlarının bir parçasıdır ve Java nesnelerini ilişkisel veritabanı tablolarına ve sorgularına eşlemek için bir API sağlar.

Spring Data: Spring Framework'te veri erişim katmanını kolaylaştırmak için bir dizi modül ve altyapı sağlar.

```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
}
```

Örnekte Spring Data'nın Jpa modulünün kullanımı gösterilmiştir. Repository beanleri oluşturmaya yardımcı olur.