AHMET BERA KANSU

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

Senkron iletişim iki partinin birbirleriyle direk olarak eş zamanlı iletişime geçmesine verilen addır. Senkron iletişimde partiler mesajın işlenip işlenmediğinin bilgisine mesajı gönderdikten hemen sonra sahip olurlar. Bir web uygulamasının başka bir servise istek atıp ardından cevap alması senkron iletişime örnek gösterilebilir.

Asenkron iletişim iletişime geçen partiler arasında üçüncü bir yapı bulundurması ile gerçekleşir. Bu yapıda gönderilmek istenen mesaj üçüncü bir partiye gönderilir. Alıcı ise kendi zamanlamasına göre istediği bir zaman mesajı alır ve işler. RabbitMQ ve Kafka ile kurulan mikroservis yapılarında asenkron iletişime dair örnekler bulunabilir. Gönderici parti alıcının üye olduğu bir kanala mesajı yollar. Alıcı da kendi periyodik dilimine göre kanalı yeni mesaj için sorgular ve mesajları işler.

2 - RabbitMQ ve Kafka arasındaki farkları araştırın? (10 PUAN)

1. Mesaj Sıralama Mekanizması:
    - RabbitMQ: RabbitMQ, bir kuyruk tabanlı mesaj sıralama mekanizması kullanır. Mesajlar, belirli bir sıraya göre kuyruklara alınır ve tüketiciler bu kuyruklardan mesajları alır. Kuyruk tabanlı bir mimari, mesajların işlenme sırasını garanti eder.
    - Kafka: Kafka, bir log tabanlı mesaj sıralama mekanizması kullanır. Mesajlar, bir log olarak depolanır ve her bir mesajın bir ofseti vardır. Tüketiciler, bir ofsetten sonraki mesajları okuyabilir, böylece Kafka, yüksek oranda paralel işleme ve düşük gecikme sağlar.
2. Veri Saklama:
    - RabbitMQ: RabbitMQ, kısa süreli mesaj geçici saklama için kullanılır. Kuyruklar genellikle belirli bir süre boyunca mesajları saklar ve tüketiciler tarafından alınana kadar geçici olarak depolanır.
    - Kafka: Kafka, verilerin kalıcı olarak saklanmasını sağlar. Mesajlar, diskte bir log olarak kalıcı olarak depolanır. Bu, veri kaybını önlemek ve verileri geri alabilmek için kullanılır.
3. İşleme Modeli:
    - RabbitMQ: RabbitMQ, yayılma ve abonelik modeline dayalı bir mesajlaşma modeli sağlar. Bir yayıncı, belirli bir kuyruğa mesajları yayınlar ve aboneler bu kuyruklardan mesajları alır.
    - Kafka: Kafka, akış işleme modeline dayalı bir mesajlaşma modeli sağlar. Mesajlar, akışlar halinde işlenir ve genellikle gerçek zamanlı veri işleme senaryoları için kullanılır.
4. Ölçeklenebilirlik:
    - RabbitMQ: RabbitMQ, dikey olarak ölçeklendirilebilir. Yani, daha fazla kapasiteye ihtiyaç duyulduğunda, daha güçlü bir tek sunucu kullanılabilir.
    - Kafka: Kafka, yatay olarak ölçeklenebilir. Kafka kümesi, birden çok broker düğümü üzerinde çalışabilir ve iş yükü arttığında daha fazla broker eklenerek ölçeklendirilebilir.
5. Kullanım Senaryoları:
    - RabbitMQ: RabbitMQ, yaygın olarak işlem kuyrukları, RPC (Uzak Yöntem Çağrısı) ve dağıtılmış sistem entegrasyonu gibi senaryolar için kullanılır.
    - Kafka: Kafka, büyük ölçekli veri akışı işleme, gerçek zamanlı veri analizi, log işleme ve olay kayıtları gibi senaryolar için kullanılır.

3 - Docker ve Virtual Machine nedir? (5 PUAN)
Docker uygulama için gerekli olan kütüphane, dosya vs. parçaları içinde barındıran bir konteynırdır. Bütün gereksinimleri sağlanan uygulama sorunsuz bir şekilde çalışır. Docker imajları işletim sistemi barındırmadığı için çalıştırılan işletim sistemine bir nebze bağımlı haldedir.

Sanal makine kendi işletim sistemini barındıran bir yazılımdır. Herhangi bir bilgisayar üzerinde kurulan sanal makine üzerinde çalıştırıldığı makineden bağımsız kendi işletim sistemine göre içinde bulunan dosyaları çalıştırır.

4 - Docker ile RabbitMQ ve PostgreSQL ve ya MySQL kurulumu yapın? (5 PUAN)
5 - Docker komutlarını örneklerle açıklayın. (5 PUAN)
docker run: Bir konteyneri çalıştırmak için kullanılır.
docker run -d -p 8080:80 nginx
Bu komut, nginx adlı bir Docker imajını kullanarak bir web sunucusu konteynerini başlatır. -d bayrağı, konteynerin arka planda çalışmasını sağlar. -p bayrağı, ana makine ile konteyner arasında port yönlendirmesi yapar.
docker ps: Çalışan konteynerleri listelemek için kullanılır.
docker images: Docker imajlarını listelemek için kullanılır.
docker build: Docker imajı oluşturmak için kullanılır.
docker build -t myimage .
Bu komut, Dockerfile adlı bir dosyadan Docker imajı oluşturur. -t bayrağı, oluşturulan imaja bir etiket ekler.
docker stop: Çalışan bir konteyneri durdurmak için kullanılır.
docker stop mycontainer
Bu komut, mycontainer adlı bir konteyneri durdurur.
docker rm: Bir konteyneri silmek için kullanılır.
docker rm mycontainer
Bu komut, mycontainer adlı bir konteyneri siler.
docker exec: Bir çalışan konteyner içinde bir komut çalıştırmak için kullanılır.
docker exec -it mycontainer bash
Bu komut, mycontainer adlı bir konteyner içinde etkileşimli bir bash kabuğu başlatır.

6 - Microservice ve Monolith mimarilerini kıyaslayınız. (15 PUAN)
Monolith Mimarisi:
Tek Parça: Monolith mimarisi, genellikle tüm uygulamanın tek bir büyük ve bütünleşik bir kod tabanında bulunduğu bir yaklaşımı ifade eder.
Teknolojik Yığın: Monolith uygulamalar, genellikle tek bir teknolojik yığın kullanır. Tüm bileşenler, genellikle aynı programlama dilinde yazılır ve aynı veritabanını paylaşır.
Birlikte Dağıtım: Monolith uygulamalar, genellikle birlikte dağıtılır. Tüm servisler ve bileşenler birlikte güncellenir ve dağıtılır.
Büyük ve Karmaşık: Monolith uygulamalar, zamanla büyüyebilir ve karmaşık hale gelebilir. Bu durum, geliştirme, test ve dağıtım süreçlerini karmaşık hale getirebilir.
Microservice Mimarisi:
Parçalara Ayrılmış: Microservice mimarisi, bir uygulamanın farklı işlevlerinin küçük ve bağımsız hizmetlere (servislere) ayrıldığı bir yaklaşımı ifade eder. Her servis, tek bir işlevi yerine getirir.

Servis Odaklı Teknoloji Yığını: Microservice uygulamaları, farklı servisler arasında iletişimi sağlamak için çeşitli teknolojik yığınları kullanabilir. Her servis kendi teknolojik gereksinimlerine göre geliştirilebilir.
Servis Bağımsızlığı ve Dağıtım: Her servis bağımsız olarak geliştirilebilir, test edilebilir, dağıtılabilir ve ölçeklendirilebilir. Bu, bir servisin değişikliklerinin diğer servislere etki etmemesini sağlar.
Esneklik ve Ölçeklenebilirlik: Microservice mimarisi, esneklik ve ölçeklenebilirlik sağlar. Her bir servis, kendi ölçeklenebilirlik gereksinimlerine göre bağımsız olarak ölçeklenebilir.
Karşılaştırma:

- Bakım ve Güncelleme: Monolith uygulamalar, tüm servislerin bir arada olması nedeniyle güncellemeleri daha karmaşık hale getirebilirken, microservice mimarisi güncellemeleri daha kolay hale getirir.
- Esneklik ve Dağıtım: Microservice mimarisi, servisler arasında bağımsızlığı ve ölçeklenebilirliği teşvik ederken, monolith uygulamalar tek bir birim olarak dağıtılır.
- Karmaşıklık: Monolith uygulamalar zamanla karmaşık hale gelebilirken, microservice mimarisi servislerin daha modüler ve bakımı daha kolay hale gelmesini sağlar.

7 - API Gateway, Service Discovery, Load Balancer kavramlarını açıklayın. (10 PUAN)
API Gateway:
API Gateway, bir uygulamanın dış dünyayla iletişimini yöneten ve bir dizi mikroservis arasındaki iletişimi yönlendiren bir servis veya yazılım bileşenidir. API Gateway, gelen istekleri alır, yetkilendirme, kimlik doğrulama, oturum yönetimi gibi güvenlik kontrollerini gerçekleştirir ve ardından ilgili mikroservislere yönlendirir. Ayrıca, gelen isteklerin şeklini dönüştürebilir ve birden çok mikroservis çağrısını tek bir istekte birleştirebilir.
Service Discovery:
Service Discovery, bir mikroservis mimarisinde, bir mikroservisin IP adresi veya konumu gibi değişkenlerinin yerine, hizmetlerin dinamik olarak bulunmasını ve adreslerinin belirlenmesini sağlar. Bu, hizmetlerin otomatik olarak birbirlerini bulmasına ve iletişim kurmasına olanak tanır. Genellikle DNS tabanlı bir servis sağlarlar.
Load Balancer:
Load Balancer, bir ağdaki gelen istekleri birden çok sunucuya dağıtan bir ağ cihazı veya yazılım bileşenidir. Bu, yükü paylaştırarak, sunucu performansını artırır ve tek bir sunucunun aşırı yük altında kalmamasını sağlar. Load Balancer ayrıca sunucu erişilebilirliğini artırabilir ve yüksek kullanılabilirlik sağlayabilir.

8 - Hibernate, JPA, Spring Data framework'lerini örneklerle açıklayın. (10 PUAN)
Hibernate:
Hibernate, Java nesnelerini ilişkisel veritabanlarıyla eşleştirmek için kullanılan bir ORM (Nesne İlişkisel Haritalama) framework'üdür. Hibernate, veritabanı işlemlerini yönetmek için kullanılır ve geliştiricilere SQL sorguları yerine Java nesneleriyle çalışma imkanı sağlar.

```
@Entity
@Table(name = "employees")
public class Employee {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   @Column(name = "name")
```

```
    private String name;

    @Column(name = "salary")
    private double salary;
}
```

JPA (Java Persistence API):
JPA, Java platformu için bir ORM standartıdır. JPA, Java nesnelerini ilişkisel veritabanlarına bağlama ve yönetme işlemlerini tanımlayan bir API seti sunar. JPA, ORM framework'leri için bir spesifikasyon olarak hizmet eder ve Hibernate gibi birçok ORM framework'ü JPA spesifikasyonunu uygular.

```
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.transaction.Transactional;

@Transactional
public class EmployeeService {

    @PersistenceContext
    private EntityManager entityManager;

    public void saveEmployee(Employee employee) {
        entityManager.persist(employee);
    }

    public Employee findEmployeeById(Long id) {
        return entityManager.find(Employee.class, id);
    }
}
```

Spring Data:
Spring Data, veri erişim katmanı işlemlerini kolaylaştıran ve standartlaştıran bir projedir. Spring Data, JPA, MongoDB, Redis, Cassandra ve diğer veritabanları için genel veri erişim işlemlerini sağlar. Spring Data, geliştiricilere tekrar eden kodları azaltır ve CRUD işlemleri için hazır ara yüzler sağlar.

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {
    // Özel sorgular buraya eklenebilir
}
```