# **2.Hafta Ödevi**

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

* Senkron:
  + Senkron iletişimde bir işlem başlatıldığında veya istek gönderildiğinde, o işlem tamamlanana kadar diğer işlemler bekler, bloklanır. Yani servisler arası senkron iletişim modelinde, bir servis diğer bir servise istek gönderdiğinde yanıtı alana kadar diğer işlemlerini askıya alır. İşlemler sıralı olarak ilerler. Asenkron iletişime kıyasla, işlemler yanıt almayı beklediğinden dolayı gecikmeler yaşanabilir.
  + Örneğin, bir alışveriş veya sipariş işlemi sırasında ödeme servisinin çağırılması. Sipariş servisi, ödeme servisine POST isteği gönderir ve ödeme işleminin sonucunu bekler. Ödeme işlemi tamamlandıktan sonra sipariş işlemi devam eder veya işlemin durumuna göre iptal olabilir.
* Asenkron:
  + Asenkron iletişimde ise bir istek gönderildiğinde veya işlem başladığında, o işlemin sona ermesi veya gönderilen isteğin yanıtı hemen beklenmez, paralel olarak diğer işlemlere devam edilir. Bu sayede, bir işlem diğerini bloke etmez ve uygulamalar arasında verimli bir şekilde veri alışverişi sağlanabilir. Bu modelde genellikle mesajlaşma sistemlerinden RabbitMQ veya Kafka kullanılarak mesaj kuyruklarından (message queue) faydalanılır.
  + Örneğin, bir e-posta sistemi. Kullanıcı bir e-posta gönderdiğinde, e-postanın gönderilmesi arka planda devam ederken kullanıcı uygulamayı kullanmaya devam edebilir.

1. **RabbitMQ ve Kafka arasındaki farkları araştırın? (10 PUAN)**

RabbitMQ ve Kafka, uygulamalar arasında mesajlaşma sağlayan mesaj queue sistemleridir (message broker).

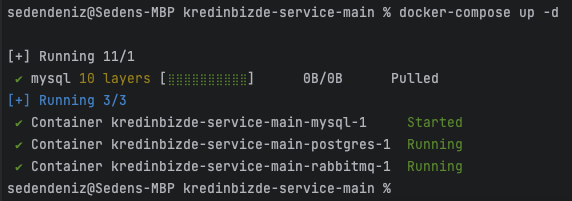
* + RabbitMQ, genellikle kısa süreli mesajlaşma ihtiyaçları için tercih edilir. Kullanımı kolaydır ve çeşitli mesajlaşma modellerini (örneğin, request/reply, publish/subscribe) destekler. RabbitMQ'da mesajlar, doğrudan alıcılara veya bir konu üzerinden yayınlanarak dağıtılabilir.
  + Kafka ise, yüksek hacimli veri akışlarını işlemek için tasarlanmıştır. Büyük datasetler üzerinde çalışan sistemlerde tercih edilir. Kafka, verileri tutarlı bir şekilde saklayabilen ve büyük miktarda verinin hızlı bir şekilde işlenip analiz edilmesini sağlayan bir yapı sunar. Kafka'nın en önemli özelliklerinden biri, mesajların uzun süre disk üzerinde tutulabilmesidir.
  + Kafka Java, Python, Node.js ve Ruby’yi destekler. RabbitMQ Java, Ruby, JavaScript, Go, C, Swift, Spring, Elixir, PHP ve .NET'i destekler.

1. **Docker ve Virtual Machine nedir? (5 PUAN)**

Docker, uygulamaları container içerisinde paketleyip, izole ederek çalıştırmayı sağlayan bir platformdur. Konteynırlar, uygulamanın çalıştırılması için gerekli tüm bağımlılıkları içerir, bu da uygulamanın farklı ortamlarda sorunsuz bir şekilde çalışabilmesini sağlar. Docker, kaynak kullanımı açısından verimlidir çünkü konteynırlar işletim sistemi çekirdeğini paylaşır ve hafif yapılıdırlar.

Virtual Machine (Sanal Makine) ise, fiziksel bir makinenin donanım kaynaklarını taklit eden işletim sistemleri çalıştırır. Her sanal makine kendi işletim sistemi örneğine sahiptir ki bu durum onları daha ağır yapar ve daha yüksek miktarda kaynak tüketmesine neden olur. Sanal makineler, uygulamalar arası tam izolasyon sağlamak ve farklı işletim sistemlerini aynı fiziksel donanım üzerinde çalıştırmak için kullanılır.

1. **Docker ile RabbitMQ ve PostgreSQL ve ya MySQL kurulumu yapın? (5 PUAN)**



1. **Docker komutlarını örneklerle açıklayın. (5 PUAN)**

* **docker run**: Bir Docker konteynırı başlatmak için kullanılır. Örneğin, “docker run hello” komutu, Docker Hub'dan "hello" imajını çeker (eğer localde yoksa) ve bir konteynırda çalıştırır.
* **docker build**: Dockerfile dosyasını kullanarak bir Docker imajı oluşturur. Örneğin, “docker build -t my-app:1.0 .” komutu, mevcut Dockerfile'ı kullanarak "my-app" adında bir imaj oluşturur ve bu imaja "1.0" etiketi verir.
* **docker images**: Localdeki Docker imajlarını listeler. docker images komutu, sistemde mevcut olan tüm Docker imajlarını ve bunların detaylarını gösterir.
* **docker ps**: Çalışan Docker konteynırlarını listeler. “docker ps” komutu aktif konteynerleri, “docker ps -a” komutu ise hem aktif hem de durmuş konteynerleri listeler.
* **docker stop**: Çalışan bir konteynırı durdurur. “docker stop konteynır\_id” şeklinde kullanılır, (“konteynır\_id “durdurmak istenilen konteynırın ID'si)

1. **Microservice ve Monotlith mimarilerini kıyaslayın. (15 PUAN)**

Microservice ve Monolith mimarileri, yazılım geliştirme süreçlerinde uygulamaları tasarlama ve dağıtma yöntemleridir.

Microservice mimarisi, uygulamanın bağımsız servisler olarak ayrılmasını sağlar, bu da her bir mikroservisin ayrı ayrı ölçeklendirilebilmesine olanak tanır. Monolith mimaride ise uygulama tek bir büyük blok olarak geliştirilir ve ölçeklendirme genellikle uygulamanın tamamını ölçeklendirme gerektirir.

Microservice mimarisi, ekiplerin minimal bağımlılıkla ve farklı teknolojiler kullanarak paralel çalışabilmesini sağlıyor. Monolith mimaride ise kod tabanı büyüdükçe, uygulamanın geliştirilmesi ve bakımı zorlaşabilir.

Microservisler bağımsız olarak dağıtılabilir, bu sayede belirli bir servis üzerinde yapılan güncellemeler tüm uygulamayı etkilemez. Monolith mimaride ise küçük bir değişiklik bile tüm uygulamanın yeniden dağıtımını gerektirebilir.

Monolith mimarisi, daha basit uygulamalar ve hızlı başlangıçlar için uygun olabilirken, microservice mimarisi büyük, ölçeklenebilir ve uzun vadeli projeler için tercih edilebilir.

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

* **API Gateway**: Uygulamalara gelen tüm istekleri kabul eden ve bu istekleri uygun mikroservislere yönlendiren bir ara katmandır. Ayrıca, yetkilendirme, izleme ve istatistik gibi işlevleri de sağlayabilir.Uygulama dışından gelen trafik için bir giriş noktası sağlar ve uygulamanın iç yapısını dış dünyadan soyutlar. Örneğin, bir mobil uygulamadan gelen bir istek, API Gateway tarafından önce kullanıcının kimlik doğrulaması yapıldıktan sonra ilgili mikroservise yönlendirilir. Bu, uygulama geliştiricilerinin güvenlik, izleme ve yönetme gibi işlemleri merkezi bir yerde ele almasını sağlar.
* **Service Discovery**: Mikroservislerin birbirini nasıl bulduğu ve iletişim kurduğu ile ilgilidir. Bu sayede, servislerin konumları ve IP adresleri gibi detayları dinamik olarak yönetilir. Mikroservisler genellikle dinamik ortamlarda çalışır ve IP adresleri veya port numaraları gibi ağ konumları değişebilir. Service Discovery, bu değişiklikleri otomatik olarak yönetir ve servislerin birbirlerine ulaşabilmesi için gerekli bilgileri sağlar. Örneğin, bir e-ticaret sitesinde bir sipariş işlemi sırasında, envanter servisi siparişin stok durumunu kontrol etmek için Service Discovery kullanarak stok yönetim servisinin güncel adresini hızlıca bulur ve siparişin gerçekleşip gerçekleşemeyeceğini belirler.
* **Load Balancer**: Gelen istekleri, birden fazla sunucu veya servis arasında dağıtan yani yükü dengeleyen bir sistemdir. Bu sayede hiçbir nokta aşırı yüklenmemiş olur. Bu hem performansı artırır hem de bir bileşinin arızalanması durumunda verimli bir kullanılabilirlik sağlar. Örneğin, bir e-ticaret sitesinde kampanya dönemlerinde kullanıcı sayısı arttığında, Load Balancer otomatik olarak trafiği, sistem üzerinde eşit bir şekilde dağıtarak tüm kullanıcılara hızlı bir şekilde yanıt verilmesini sağlar.

1. **Hibernate, JPA, Spring Data framework’lerini örneklerle açıklayın. (10 PUAN)**

Her biri aslında Java ile veritabanı işlemlerini gerçekleştirmek için çözümler sunsa da farklı özellikleri ve kullanım alanalrı vardır.

* **Hibernate:** Java için bir ORM (Object-Relational Mapping) framework’üdür. Veritabanındaki tabloları ve sütunları Java sınıfları ve alanlarına eşler, böylece geliştiriciler veritabanı sorgularını direkt olarak SQL yerine Java kodları üzerinden yönetebilirler. Örneğin, bir User classında Hibernate kullanarak bu classı veritabanındaki users tablosuna eşleyebilir ve SQL sorguları yazmadan veritabanı işlemleri gerçekleştirilebilinir.

import javax.persistence.Entity;

import javax.persistence.Id;

@Entity

public class User {

@Id

private Long id;

private String name;

private String email;

}

* **JPA (Java Persistence API):** Java platformunda veri kalıcılığı (persistence) için bir standarttır. Hibernate, JPA'nın bir uygulamasıdır ama JPA farklı ORM çözümleriyle çalışabilir. JPA, uygulamaların veritabanı işlemlerini yönetmek için bir arayüz sağlar, böylece geliştiriciler ORM frameworkünü değiştirse bile aynı API'ı kullanabilirler. Bir Book sınıfını bir books tablosuna eşlemek için JPA anotasyonları kullanılabilir, bu da kodun ORM çözümünden bağımsız olmasını sağlar.
* **Spring Data:** Veri erişim katmanını kolayca geliştirmek için yüksek seviye abstraksiyonlar ve destek sağlar. Spring Data, JPA, MongoDB, Cassandra gibi çeşitli veritabanı teknolojileri için repository destekleri sunar, böylece geliştiriciler CRUD (Create, Read, Update, Delete) işlemlerini kolayca gerçekleştirebilirler. Örneğin, bir PersonRepository arayüzü oluşturup CrudRepository'den türeterek, Spring Data bu sınıfın somut bir implementasyonunu otomatik olarak oluşturur. Bu, veritabanı işlemlerini çok daha az kodla yapmamızı sağlar.

import org.springframework.data.repository.CrudRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface UserRepository extends CrudRepository<User, Long> {

}