Q1) Senkron ve Asenkron iletişim

Senkron iletişimde, veri iletimi için gönderici ve alıcı arasında bir zamanlama mekanizması kullanılır. Gönderici ve alıcı, belirli bir saat diliminde veya zamanlama sinyalleri ile senkronize olur. Bu zamanlama, verinin doğru bir şekilde gönderilmesini ve alınmasını sağlar. Senkron iletişimde veri paketleri belirli aralıklarla ve belirli bir sırayla gönderilir. Örnek olarak TDM (Time Division Multiplexing-Zaman Bölmeli Çoğullama) ve FDM (Frequency Division Multiplexing - Frekans Bölmeli Çoğullama) verilebilir.

TDM

TDM, Time Division Multiplexing (Zaman Bölmeli Çoğullama), dijital veya analog bölümleme tiplerinden biridir. Bu tip bölümlemelerde birden fazla sinyalin veya bit dizilerinin tek bir kanal üzerinden gönderilmesi kullanılır. Bu işlem her kanala belirli bir zaman parçası (timeslot) ayırmakla meydana gelir.

FDM

FDM, Frequency Division Multiplexing (Frekans Bölmeli Çoğullama), hücresel ve analog sistemlerde çoğullama tekniklerinden biridir. Bu teknik, radyo vericilerinin aynı frekans spektrumunda veri göndermesi için kullanılır. Yöntem, frekans bandını 30'a bölüp, her bir bölümden ses ve data haberleşmesi gerçekleştirir. Her kanaldan aynı anda sacece 1 kişi yararlanabilir ve kanallar kullanıcı talebi olduğunda ilgili kullanıcıya tahsis edilir.

Asenkron iletişimde ise, veri gönderici ve alıcı arasında önceden belirlenmiş bir zamanlama mekanizması olmadan gerçekleşir. Veri paketleri, gönderici tarafından alıcıya doğrudan gönderilir ve alıcı, paketleri kendi hızında işler. Gönderici ve alıcı arasında bir senkronizasyon sağlanmaz, bu nedenle veri paketleri arasındaki zaman aralıkları değişebilir. Örnek olarak TCP/IP protokolü verilebilir.

TCP

TCP (Transmission Control Protocol) bilgisayarlar arasındaki iletişimin, küçük paketler hâlinde ve kayıpsız olarak gerçekleştirilmesini sağlayan bir protokoldür.

IP

IP, Türkçe açılımı ve çevirisiyle internet protokolünün kısaltmasıdır. Bilgisayarların birbirleri ile iletişiminde en önemli nokta olan ağ adreslemede kullanılan düzendir. IP (internet protokolü), iki bilgisayar arasındaki paketlerin yönlendirilmesini sağlayan bir protokol olarak tanımlanır.

Q2) RabbitMQ ve Kafka arasındaki farklar

RabbitMQ ve Apache Kafka, ikisi de popüler mesaj kuyruğu (message broker) sistemleridir.

Mesaj Dağıtım Modeli

RabbitMQ: Bir "merkezi iletişim" modeli kullanır. Mesajlar, RabbitMQ sunucusuna gönderilir ve ardından abonelere veya hedeflere iletilir.
Kafka: Bir "işlem günlüğü" (log) modeli kullanır. Mesajlar, Kafka brokerlarında saklanır ve ardından aboneler istedikleri zaman bu mesajları okuyabilirler.

Kullanım Senaryoları

RabbitMQ: İşbirliği gerektiren uygulamalar, geleneksel mesajlaşma kuyruklarıyla (örneğin, RPC çağrıları) uyumlu sistemler için daha uygundur.
Kafka: Büyük ölçekli veri akışları, olay işleme (event processing), günlük analizi ve gerçek zamanlı veri işleme gibi senaryolar için daha uygundur.

Mesaj Saklama ve Kaybı

RabbitMQ: Mesajlar, tüketici tarafından işlenene kadar RabbitMQ sunucusunda saklanır. Ancak, sunucu çökmesi durumunda mesajlar kaybedilebilir.
Kafka: Mesajlar, diskte saklanır ve belirli bir süre boyunca muhafaza edilir. Bu nedenle, tüketiciler mesajları işleyemediğinde bile veri kaybı olmaz.

İşleme Hızı

RabbitMQ: Daha hızlı mesaj teslimatı için optimize edilmiştir, ancak Kafka kadar yüksek işleme hızlarına sahip değildir.
Kafka: Büyük ölçekli veri akışlarını destekleyecek şekilde tasarlanmıştır ve çok yüksek işleme hızlarına ulaşabilir.

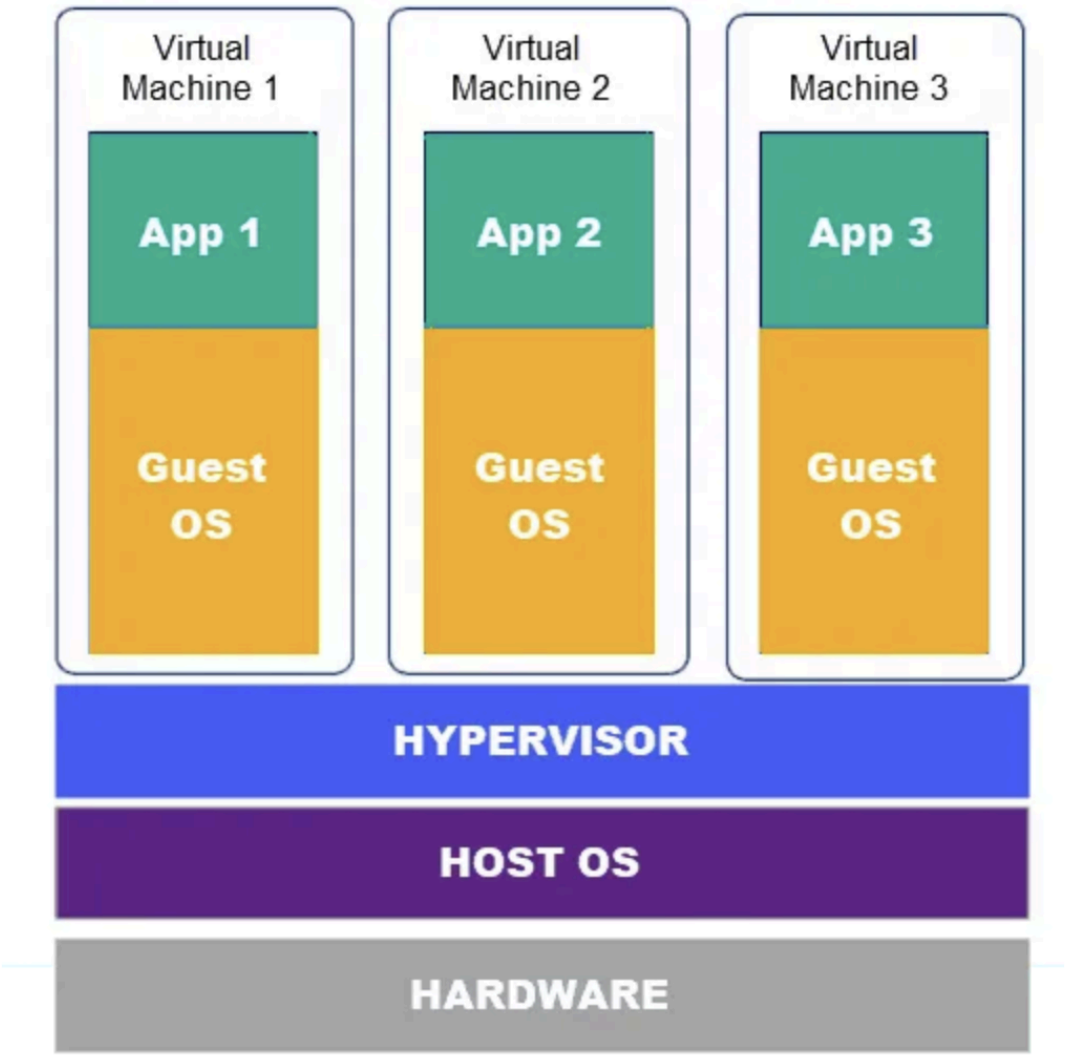
Ölçeklenebilirlik

RabbitMQ: Dikey ölçeklenebilirlik (daha fazla kaynak eklemek) sağlar.
Kafka: Yatay ölçeklenebilirlik (daha fazla sunucu eklemek) sağlar.

Q3) Docker ve Virtual Machine

Virtual Machine

Sanal makine en basit tabir ile, bir bilgisayar sistemi üzerinde birden fazla işletim sistemi çalıştırmamıza olanak sağlayan bir sanallaştırma teknolojisidir.



Sanal Makinenin Yapısı — Mimarisi

Docker

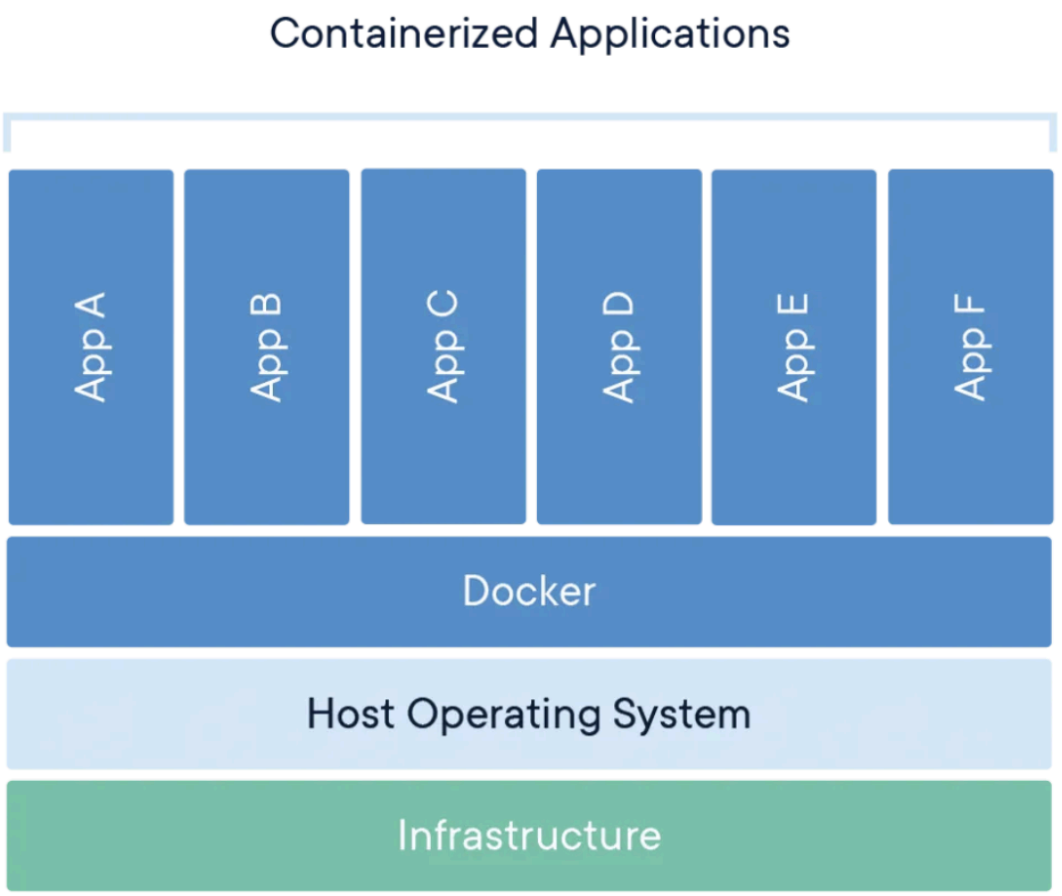
Docker, yazılım uygulamalarını hızlı bir şekilde geliştirmek, dağıtmak ve çalıştırmak için kullanılan bir platform ve açık kaynaklı bir yazılım ürünüdür. Docker, yazılım uygulamalarını birimler halinde paketlemek için kullanılan bir konteynerleştirme teknolojisini temel alır. Bu sayede uygulamalar, bağımlılıkları ve konfigürasyonlarıyla birlikte kapsülleme edilir ve hemen hemen herhangi bir ortamda konsolide bir şekilde çalıştırılabilir.

Docker Engine: Docker'ın temel çalışma motorudur. Docker Engine, Docker konteynerlerinin oluşturulması, çalıştırılması ve yönetilmesinden sorumludur.

Docker Image: Bir Docker konteynerinin çalıştırılabilir bir örneğini oluşturmak için kullanılan şablon. Docker Image'ler, uygulama kodunu, çalışma zamanı, kütüphaneleri, bağımlılıkları ve diğer gerekli bileşenleri içerir.

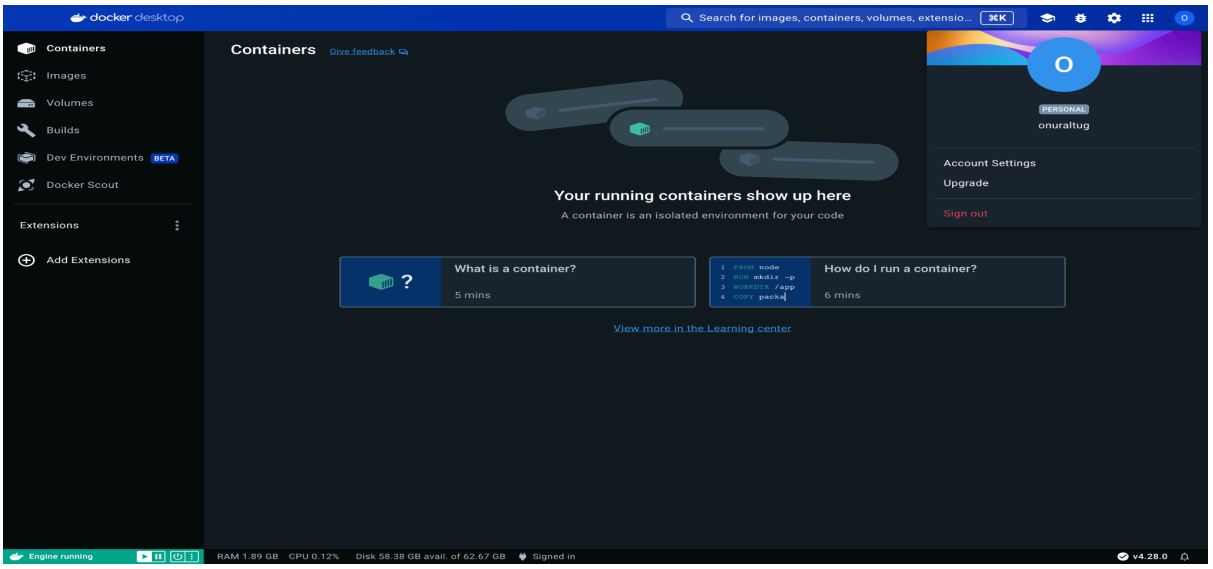
Docker Container: Bir Docker Image'in çalıştırılabilir bir örneği. Docker Container'lar, Docker Image'lerinden türetilir ve izole bir ortamda çalıştırılır.

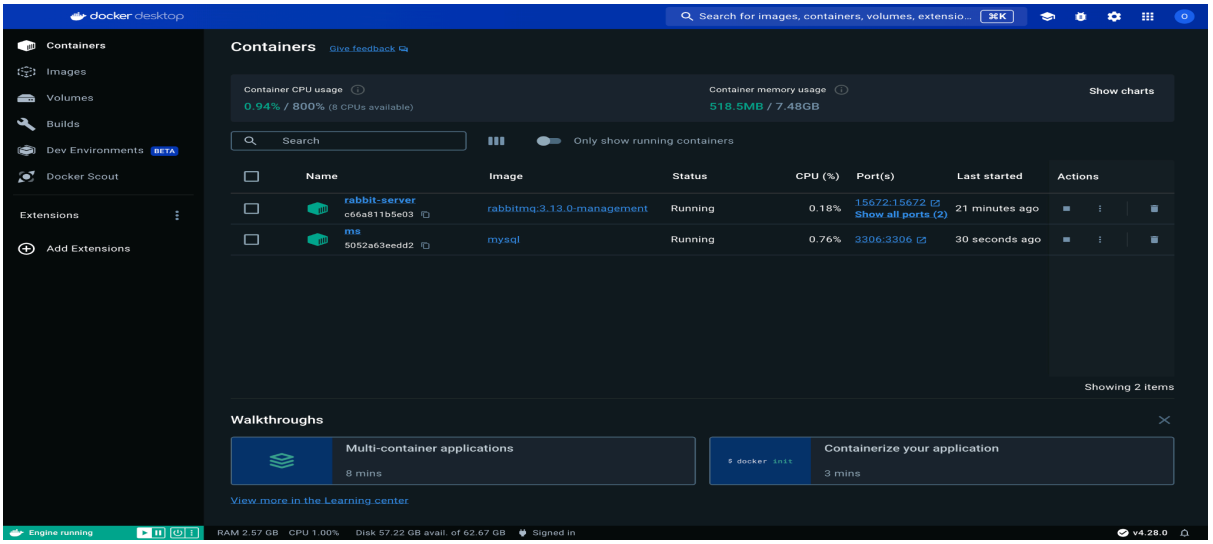
Dockerfile: Bir Docker Image'in nasıl oluşturulacağını tanımlayan metin tabanlı bir dosya. Dockerfile, bir uygulamanın yapılandırılması, bağımlılıklarının tanımlanması ve Docker Image'in oluşturulması için kullanılır.

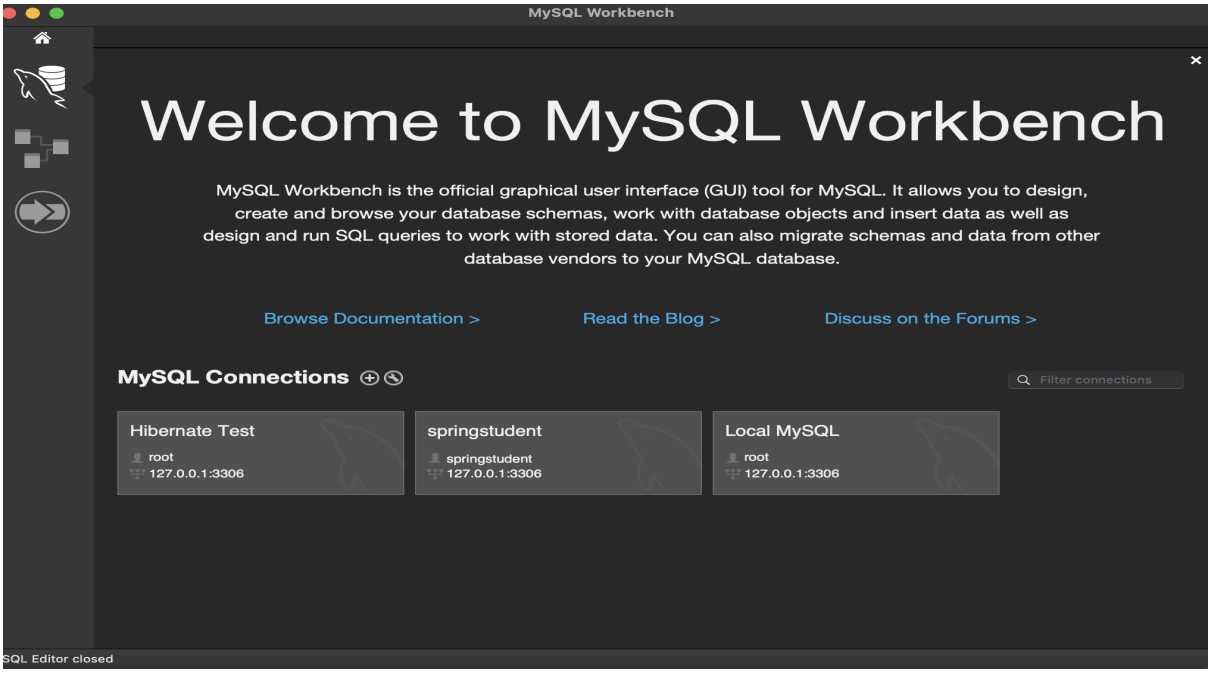


Konteynerların Yapısı — Mimarisi

Q4) Docker ile RabbitMQ ve MySQL kurulumu







Q5) Docker komutları

Image indirme
docker pull redis
docker pull kullaniciRegistirysi/nginx localhost:5000/myadmin/nginx

Image yükleme
docker push kullaniciRegistirysi/redis
docker push kullaniciRegistirysi/nginx localhost:5000/myadmin/nginx

Volume oluşturma
docker volume create volumeAdi

Container adını değiştirme
docker rename containerAdi yeniContainerAdi

Container güncelleme
docker update --cpu-shares 512 -m 300M containerAdi

Container bloklama
docker wait nginx

SIGKILL göndererek containerı durdurma
docker kill nginx

Container içeriğini görüntüleme
docker inspect containerAdi
docker inspect --format '{{ .NetworkSettings.IPAddress }}' $(docker ps -q)

Container olaylarını görüntüleme
docker events containerAdi

Public olan portları listeleme
docker port containerAdi

Container içinde çalışan işlemleri görüntüleme
docker top containerAdi

Container kaynak kullanımını görüntüleme
docker stats containerAdi

Containerın barındırdığı dosya veya klasörlerde yapılan değişiklikleri görüntüleme
docker diff containerAdi

Image build etme
docker build .
docker build github.com/creack/docker-firefox
docker build - < Dockerfile
docker build - < context.tar.gz
docker build -t kullaniciRegistrysi/containerAdi .
docker build -f digerDockerFile .
curl example.com/remote/Dockerfile | docker build -f - .

Tar ile sıkıştırılmış depodaki container imagelerin Docker'a yüklenmesi
docker load < archlinux.tar.gz
docker load --input alpine.tar

Image'in tar dosyası haline getirilmesi
docker save busybox > centos.tar

Image'e tag eklenmesi
docker tag cassandra kullanici Registry/cassandra

Ağ oluşturulması
```
docker network create -d overlay MyOverlayNetwork
docker network create -d bridge MyBridgeNetwork
docker network create -d overlay \
  --subnet=192.168.0.0/16 \
  --subnet=192.170.0.0/16 \
  --gateway=192.168.0.100 \
  --gateway=192.170.0.100 \
  --ip-range=192.168.1.0/24 \
  --aux-address="my-router=192.168.1.5" --aux-address="my-switch=192.168.1.6" \
  --aux-address="my-printer=192.170.1.5" --aux-address="my-nas=192.170.1.6" \
  MyOverlayNetwork
```

Ağların silinmesi
docker network rm MyOverlayNetwork

Ağların listelenmesi
docker network ls

Ağ hakkında detaylı bilgilerin listelenmesi
docker network inspect MyOverlayNetwork

Çalışan containerın ağa bağlanması
docker network connect MyOverlayNetwork nginx

Containerın çalıştırıldığı anda ağa bağlanması
docker run -it -d --network=MyOverlayNetwork nginx

Containerın ağdan çıkarılması
docker network disconnect MyOverlayNetwork nginx

Kullanılmayan volumeların hepsini silme
docker volume prune volumeAdi

Containerı volume ile beraber silme
docker rm -v redis

Kapatılmış bütün containerların silinmesi
docker rm $(docker ps -a -f status=exited -q)

Durdurulmuş bütün containerların silinmesi
docker rm `docker ps -a -q`
docker rm $(docker ps -a -q)

Dangling olan bütün imagelerin silinmesi
docker rmi $(docker images -f dangling=true -q)

Bütün imagelerin silinmesi
docker rmi $(docker images -a -q)

Bütün containerların durdulması ve silinmesi
docker stop $(docker ps -a -q) && docker rm $(docker ps -a -q)

Dangling olan bütün volumeların silinmesi
docker volume rm $(docker volume ls -f dangling=true -q)

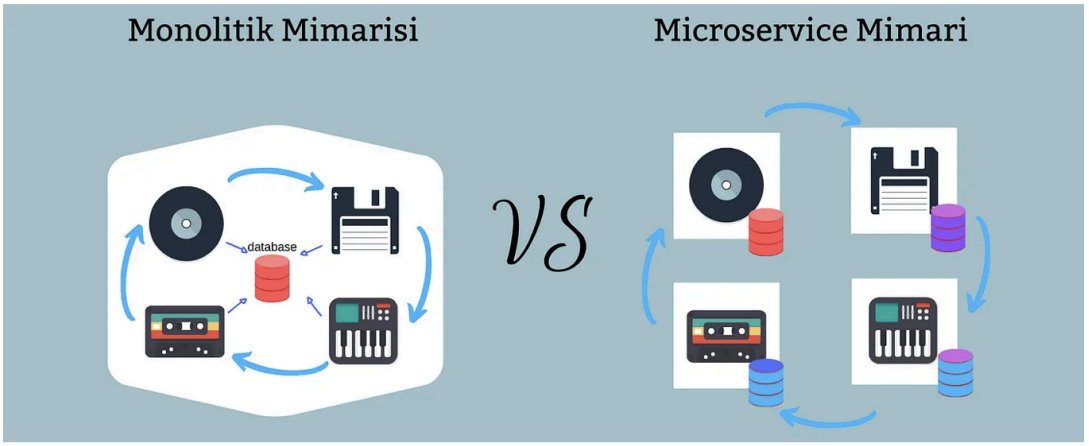
Q6) Microservice ve Monolith mimarileri

Monolitik bir uygulama, birden fazla modül içeren tek bir kod tabanına sahiptir. Modüller, fonksiyonel veya teknik özelliklerine göre ayrılmıştır. Tüm uygulamayı build eden tek bir derleme sistemine sahiptir. Ayrıca tek bir çalıştırılabilir veya deploy edilebilir dosyaya sahiptir.

Monolitik yaklaşım bir yazılım uygulaması oluşturmak için varsayılan bir yöntemdir. Yine monolitik bir yaklaşımı benimsemek devasa bir kod tabanını kullanımını da beraberinde getirmektedir. Bunun yanı sıra yeni bir teknoloji benimsemek, ölçeklendirmek, deployment süreçleri, yeni değişiklikler uygulamak gibi birtakım zorlukları mevcuttur.

Microservice, tek işe odaklı uygulamanın işlevsel küçük bir parçasını ifade eder. Microservice olarak tasarlanan bir servis, ihtiyaca bağlı olarak birden fazla projede(uygulamada) veya farklı projelerde tekrar kullanılabilir. Servisler arasındaki bağımlılıklar, loose-coupled ilkesine uygun olarak en aza indirgenir. Tek sorumluluk odaklı servisler bir standarda uygun olunca, diğer servisler üzerindeki değişikliklerden etkilenmezler.

Microservice yaklaşımı ölçeklenebilirlik, esneklik, çeviklik gibi önemli avantajlar barındırır. Bu yüzdendir ki birçok önemli ve büyük şirket Netflix, Google, Amazon, Spotify gibi teknoloji liderleri, monolitik mimarilerinden microservice mimarisine geçiş yapmayı başarıyla tamamlamışlardır. Bu arada, pek çok şirket bu yöntemi kullanmaya başlamakta ya da var olan sistemlerini bu mimariye evirmektedir.



Monolitik vs Microservice Mimarisi

Q7) API Gateway, Service Discovery, Load Balancer kavramları

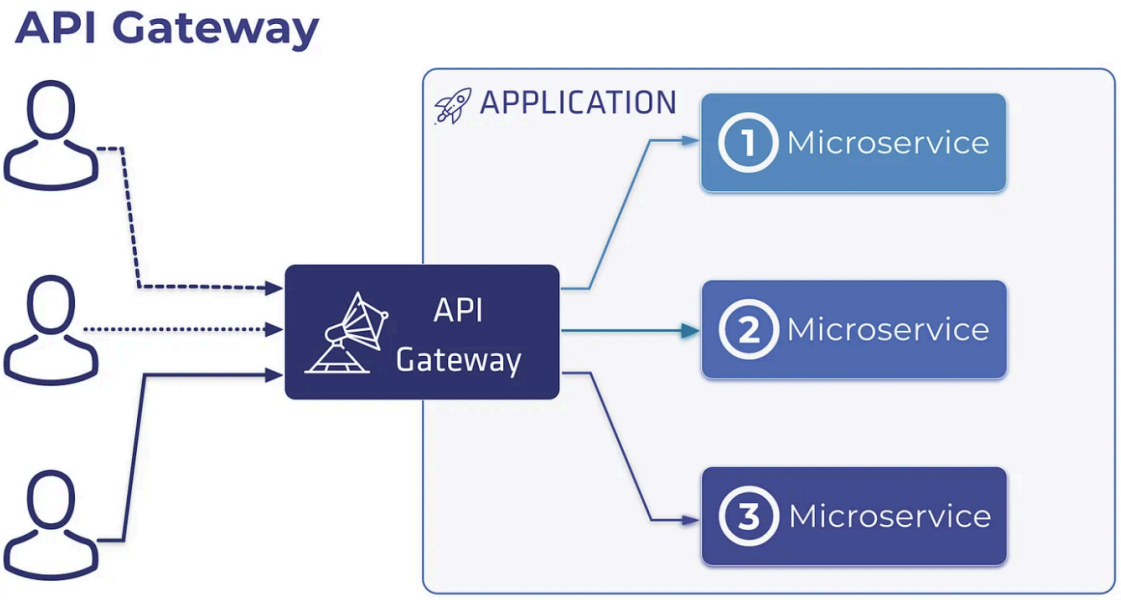
API Gateway

Gateway, client tarafından servislerimize gelen istekleri karşılayan yapılar olarak nitelendirebiliriz. API Gateway temel işlevi client tarafından istekleri alıp gerekli servislere iletmektir. Bu sayede client tarafından servisler için herhangi bir port bilgisinin bilinmesine gerek kalmamaktadır.

Response Caching: Gateway, servislere gelen isteklerden dönen sonuçları cacheleyebilir, böylece maliyet düşürülebilir.

Authentication ve Authorization: Daha servise varmadan eğer kullanıcının, isteğin gönderildiği serviste yetkisi yok ise 401(Unauthorized) hatası daha servise ulaşmadan, gateway tarafından döndürülebilir.

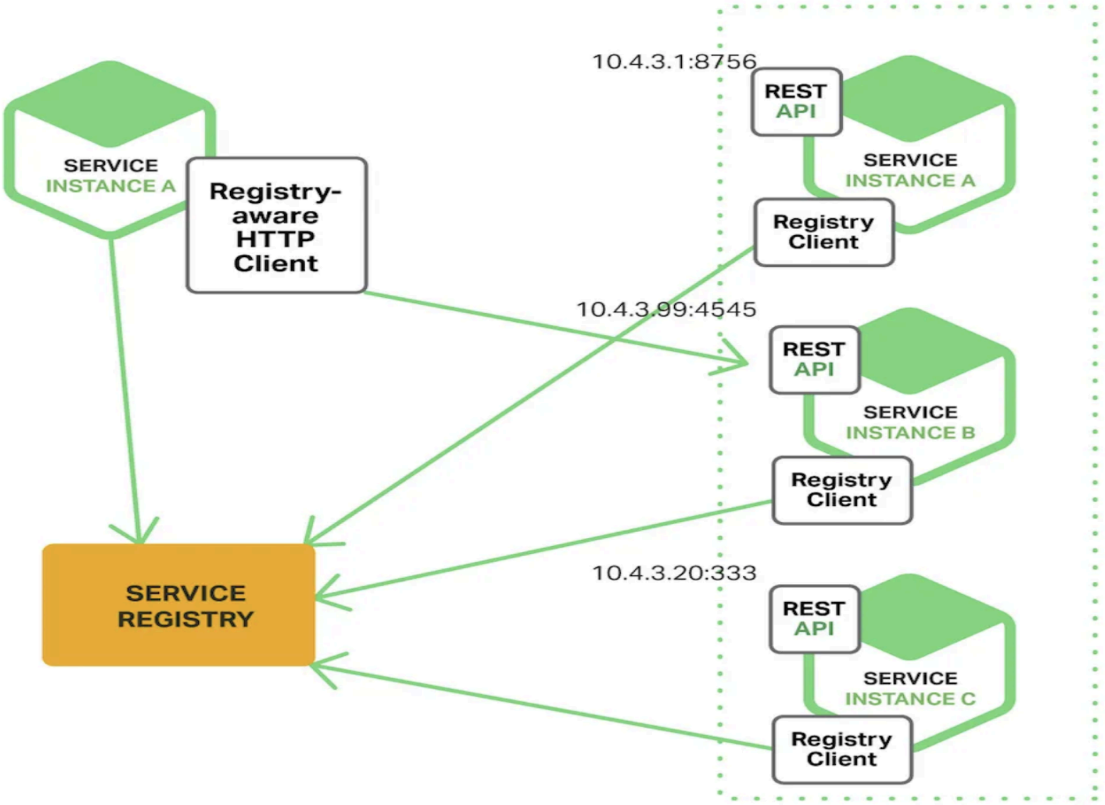
Logging: Servislere gelen istekleri loglayabiliriz, bununla beraber hangi servise çok fazla istek geliyor gibi sorulara yanıt verebiliriz.



Service Discovery

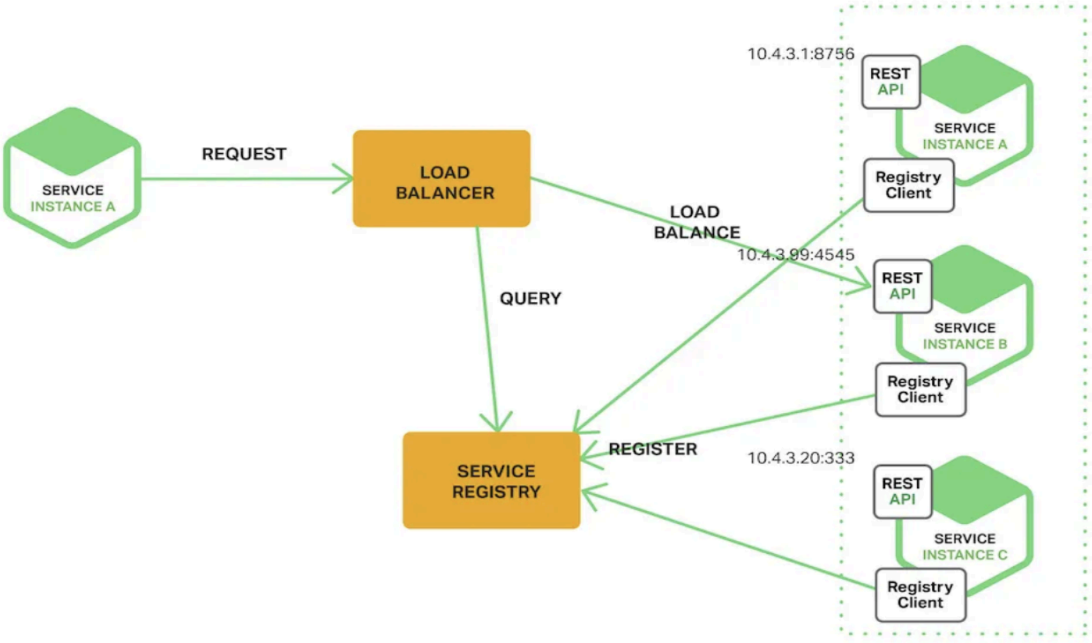
Client-Side Service Discovery

Bu yapıda client service registry ile iletişime geçer ve istediği domainin adresini sorar.Service registry üzerinde kayıtlı olan yani ulaşılabilir durumdaki microservice instance'larının ip adreslerini verir.Daha sonra client bir load-balancing algoritmasına göre bu microservicelerden en uygun olanı seçer ve microservice ile iletişim kurar.



Server-Side Service Discovery

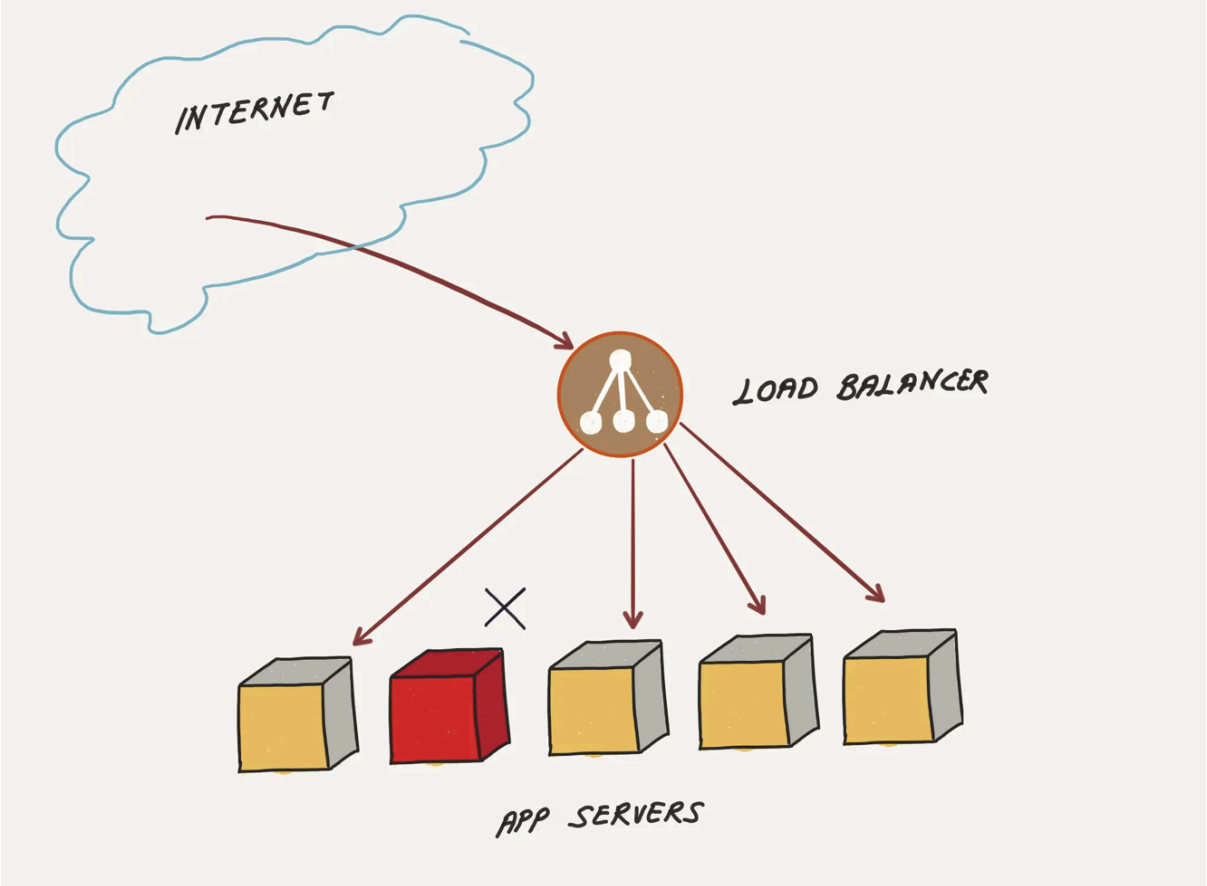
Burada ise client bir Load Balancer(Yük Dengeleyici) ile iletişim kurar.İsteğini buraya iletir.Load Balancer ise Service Registry ile konuşarak avaliable durumdaki instance'ların listesini alır.Daha sonra da arasından kendi algoritmasına göre en uygun gördüğü instance'a client'tan aldığı isteği yönlendirir.



İkisinin arasındaki fark Client-Side Service Discovery yapısında client,instance ile kendisi iletişim kurmaktadır.Çünkü o instance'ın ip adres bilgisini service registry'e sorarak öğrenmektedir.Server-Side Service Discovery örneğinde ise request load balancer aracılığı ile instancelara iletilmektedir.

Load Balancer

Yük dengeleyiciler (LB — Load Balancer), yükü dağıttıkları sistemlerin sağlıklı olup olmadığını belirli aralıklarla kontrol ederek sağlıklı çalışmayan sunuculara trafiği yönlendirmekten imtina edebilirler. Bu sayede, yük dengeleyiciler sistemlerimizin ölçeklenmesini sağladıkları gibi yüksek erişilebilirliğini (HA — High Availability) sağlamak için de kullanılırlar. Problemli sunucuya yeni istek gönderilmez ve kullanıcıların problemden etkilenmemesi sağlanır. Ek olarak, yük dengeleyiciler sayesinde uygulama güncellemeleri sistemde kesintiye sebep olmadan yapılabilir. HA durumuna benzer şekilde, öncelikle güncellenecek sunucuya LB'nin trafik yönlendirilmemesi sağlanır.



Q8) Hibernate, JPA, Spring Data Framework

Hibernate

Hibernate Java nesnelerini bir veritabanında kalıcı hale getirmek / kaydetmek için bir frameworkdür. Hibernate, Nesneden İlişkisel Eşleme (ORM) sağlar. ORM Object Relational Mapping anlamına gelmektedir. ORM toolları nesne tabanlı programlamada bulunan objeler ile veritabanı sistemimizdeki tablolar arasında köprü görevi kurulmasını sağlar. Objeleri ilişkisel veritabanında mapping yapmaya yarar. ORM toollarının kullanımının en büyük avantajı OOP'da yer alan inheritance, polymorphism gibi konseptleri veritabanımız ile kolaylıkla kullanabilmektir. Ayrıca bazı durumlarda projelerde SQL sorguları yazılımcıya büyük yük olabiliyor. Özellikle proje büyüdüğünde bu işlemler zorlaşıyor. ORM toollarından önce yazılımcılar kendileri objeleri veritabanındaki tablolarla eşleştirecek kodlar yazıyorlardı. Hibernate gibi toollar yazılımcıları bu yükten kurtardı.

```java
@Entity
@Table(name = "Students")
public class Student {

        @Id
        @Column(name = "id")
        private int id;

        @Column(name = "first_name")
        private String firstName;

        @Column(name = "last_name")
        private String lastName;

        @Column(name = "email")
        private String email;
```

Java projemiz içerisinde yer alan Entity class

id	first_name	last_name	email
1	Will	Smith	will@test.com
2	Leonardo	DiCaprio	leonardo@test.com
NULL	NULL	NULL	NULL

Veritabanımızda yer alan Students tablosu

JPA

Daha önce Java Persistence API olarak bilinen Jakarta Persistence API (JPA) Nesneden İlişkisel Eşleme (ORM) için Standart API'dır. JPA, Java objeleri ve ilişkisel (relational) database arasında bilgi aktarımı için kullanılan bir standarttır. JPA bu ikisi arasında bir köprü görevi görür. JPA kullanabilmek için projeye implemente edilmesi gerekir ve bu yüzden Java dilindeki Hibernate, TopLink gibi çeşitli ORM araçları bilgiler konusunda bir devamlılık sağlamak için JPA kullanır. JPA aracılığıyla farklı ORM araçları ufak birkaç ayarlama dışında yazılan kod değiştirilmeden kullanılabilir.

```java
package com.example.accessingdatajpa;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Customer {

  @Id
  @GeneratedValue(strategy=GenerationType.AUTO)
  private Long id;
  private String firstName;
  private String lastName;

  protected Customer() {}

  public Customer(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }
```

```java
package com.example.accessingdatajpa;

import java.util.List;

import org.springframework.data.repository.CrudRepository;

public interface CustomerRepository extends CrudRepository<Customer, Long> {

  List<Customer> findByLastName(String lastName);

  Customer findById(long id);
}
```

Spring Data

Spring Data, Spring Framework ekosisteminde yer alan bir kütüphanedir. Interfaceler aracılığı ile veri iletişimini hazır metodlarla birlikte kullanmamıza aracılık etmektedir. Projemizde var olan entitymizin alanlarına göre dinamik olarak metodlar oluşturup SQL söz dizimine gerek kalmadan veri alabilmemiz mümkün olup veritabanı bağımsız çalışmaktadır. Çok karıştırılan bir nokta ise Hibernate'in yerini aldığı. Bu oldukça yanlış bir düşünce Spring Data Hibernate ile birlikte çalışabilir ancak yerini alan bir yapı değildir. Spring Data'nın çalışması için bir JPA aracına ihtiyaç bulunmaktadır. Örnek olarak CrudRepository sayesinde Crud operasyonları için bir kod yazmaya gerek yoktur.

```
@Entity
public class Ogrenci {

  @Id
  @GeneratedValue(strategy=GenerationType.AUTO)
  private Long id;
  private String adi;
  private String soyAdi;
  ..
  ..
  ..
}
```

```
public interface OgrenciRepository extends CrudRepository<Ogrenci, Long> {

}
```

```
ogrenciRespository.save(ogrenci);
ogrenciRespository.delete(ogrenci);
ogrenciRespository.update(ogrenci);
ogrenciRespository.findById(id);
```

References

TDM
https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/07/tdm-(time-division-multiplexing-zaman-b%C3%B6lmeli-%C3%A7o%C4%9Fullama)

FDM
https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/07/fdm-frequency-division-multiplexing-frekans-bolmeli-cogullama

TCP/IP - https://berqnet.com/blog/tcp-ip

Docker and Virtual Machines
https://medium.com/software-development-turkey/dockera-giri%C5%9F-virtual-machines-vs-containers-ea67848dfefb

Docker komutları
https://medium.com/kodgemisi/docker-etkili-kullanmak-icin-komutlar-93faf9329142

Microservice ve Monolith mimarileri
https://gokhana.medium.com/monolitik-mimari-ve-microservice-mimarisi-aras%C4%B1ndaki-farklar-bd89ac5b094a

API Gateway
https://medium.com/@yigitcanolmez/api-gateway-nedir-microservice-mimarisinde-api-gateway-kullanmak-8c7fe8e6ec58

Service Discovery
https://medium.com/i%CC%87yi-programlama/microservice-mimarilerinde-service-discovery-7a6ebceb1b2a

Load Balancer
https://medium.com/@gokhansengun/load-balancer-nedir-ve-ne-i%C5%9Fe-yarar-32d608f98ef9

Hibernate
https://tugrulbayrak.medium.com/hibernate-1-orm-kavram%C4%B1na-giri%C5%9F-c2ba2f2a3bfe

JPA - https://spring.io/guides/gs/accessing-data-jpa

Spring Data - https://blog.burakkutbay.com/spring-data-nedir.html/