1. Senkron ve Asenkron iletişim nedir örneklerle açıklayın?

Senkron iletişimde başka bir işe geçmeden önce mevcut işi bitirmek gerekliliği vardır. Senkron bir api mimarisi düşündüğümüzde response dönene kadar o threadde farklı bir işlem yapılamaz. Bir diğer örnek ise meşhur dosya okuma örneği verilebilir. Program bir dosyayı okurken işlemini tamamlamadan başka bir işlem yapamaz. Yine HTTP protokolü senkron çalışır.

Asenkron iletişimde ise tam tersi birden fazla işi eş zamanlı yapabiliriz. Aslında gerçekte bilgisayarımız tek çekirdekten oluşuyorsa aynı anda birden fazla işi yapmamız mümkün olmuyor. Aklıma bu noktada Round-robin gibi algoritmalar geliyor. Örnek vermek gerekirse, anlık mesajlaşma uygulamaları iyi bir örnek olacaktır. Sonuçta karşı taraftan bir cevap gelmesini beklemene gerek bulunmamaktadır. HTTP protokolünün aksine Websocket asenkron modeldir.

1. RabbitMQ ve Kafka arasındaki farkları araştırın?

RabbitMQ’de mesaj sağlayıcı mesajı gönderdikten sonra bu dinleyecek olan yere doğru şekilde ulaşıp ulaşmadığını izler/kontrol eder. Kafka ise hiç farketmeksizin mesajı kuyruğa gönderir ister bu mesaj alınsın ister alınmasın. İkisi de aynı amaca hizmet ediyor gibi görünsede farklı kullanım alanları için dizayn edilmiştir. RabbitMQ’da tüketici pasif rol alır, mesajın kuyruğa gönderilmesini bekler. Banka uygulama örneği veriliyor burada bir işlem için sms mesajının beklenmesi gibi. Kafka da ise tüketiciler daha aktif sadece okumakla kalmıyor aynı zamanda takipte ediyor. Son mesajın sırasını tutar ve yeni mesaj geldiğinde onu okur ve bir arttırır. RabbitMQ de mesaj bir kez tüketiciye ulaştığında tüketici, mesajı gönderene onaylama (ACK) gönderir ve kuyruktan mesaj silinir. Kafkada ise mesajlar bir log dosyasına eklenir bir süre orada tutulur. Olumsuz bir durumla karşılaşıldığında oradan tekrar okumak için zamanı vardır ve bunun yanında Kafka daha çok gerçek zamanlı uygulamar tarafından RabbitMQ’ye göre daha çok tercih edilir çünkü saniyede çok daha fazla mesaj gönderebilir.

1. Docker ve Virtual Machine nedir?

Sanal makine, fiziksel bir bilgisayardır aslında ve yine sanal olarak ayrılan bir işletim sistemi ortamını temsil eder. Her sanal makine kendi işletim sistemine ve uyuglamalarına sahiptir. Bağımsız çalışabilirler bu sebepten dolayı. Örneğin bir sunucuda windows işletim sistemine sahip bir makine varken aynı sunucuda Linux işletim sistemine sahip bir makine de olabilr.

Docker ise, bir uygulamanın yazılım paketlemesi, dağıtımı ve çalıştırılması için bir olanak sağlar. Konteyner teknolojisi kullanır ve bunu hafif taşınabilir izole birimlere ayırır ve paketler. Işletim sistemi seviyesinde izolasyon sağlar.

1. Docker ile RabbitMQ ve PostgreSQL veya MySQL kurulumu yapın?

RabbitMQ, PostgreSQL gibi teknolojilerin imajları Dockerhub’da mevcuttur. DockerHub’a kendi yaptığımız uygulamalarıda atabiliriz. Docker’ın sunduğu bulut tabanlı bir hizmettir. Oradan imajları öncelikle pull etmemiz gerekmektedir.

Bunun için terminale -> docker pull rabbitmq yazmamız yeterlidir. Bunu yaparak aslında RabbitMQ imajını çekmiş oluyoruz. İmaj ise bir programın/uygulamanın çalışması için gereken tüm kod ve araçları içerir.

Sonraısnda bu imajı çalıştırmak için aşağıdakini terminalde çalıştırmak yeterlidir.

docker run -d --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq

Bu noktadans sonra localhost:15672 den rabbitmq’nun arayüzüne erişebiliriz ve yine rabbitMQ ile iletişime geçmek istiyorsak da 5672 portunu kullanırız.

PostgreSQL için ise yine aynı adımları izleyebiliriz ancak burada biraz farklılık oluşacaktır.

Docker pull postgres ile yine dockerhubdan ilgili imajı çekeriz. Sonrasında konteryniri ayağa kaldırmak için aşağıdaki gibi bir komut yazmamız yeterlidir.

docker run --name my\_postgres -p 5432:5432 -e POSTGRES\_PASSWORD=mysecretpassword -d postgres

1. Docker komutlarını örneklerle açıklayın.

Docker run -> bir konteryneri çalıştırmak için kullanılır.

Docker pull -> bir imajı hub’dan çekmek için kuıllanılır.

Docker build –t -> bir docker dosyasından imaj oluşturmak için kullanılır.

Docker ps -> ayakta olan konteynerları listeler.

Docker images -> Yereldeki imajları listeler.

Docker stop <konteyner numarası> -> çalışan konteyneri durdurur.

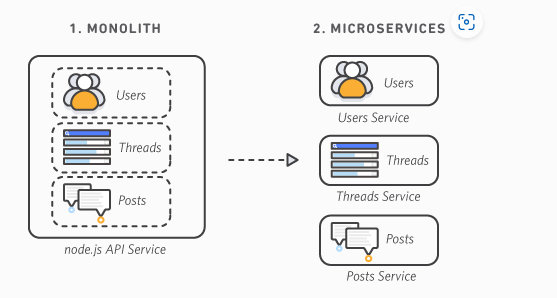
Docker rm <konteyner numarası> -> bir konteyneri silme için kullanılır.

Docker rmi <imaj numarası> bir imajı silmek için kullanılır.

Docker-compose up -> çoklu konteyner başlatmak için kullanılır.

1. Microservice ve Monotlith mimarilerini kıyaslayın.

Monolit mimari, tüm bir uygulamanın tek bir kod tabanında ve tek bir dağıtımda birleştiği yapıdır. Genellikle tek bir veritabanı uygulama tarafından paylaşılır. Potansiyel olarak risklidir ancak hala günümüzde neredeyse yarısından fazla şirket monolit uygulama mimarisi benimsemektedir. Riskli olmasının sebebi bu projelerde istenilen kadar abstraction kullanılsada birbirine bağımlıdırlar. A noktasında yaptığınız bir değişiklik F noktasında bir yerleri ciddi şekilde etkileyebilir ve proje büyüdükçe bu bağımlılıkların ve yan etkilerin önüne geçmek zorlaşır. Monolit mimarinin bazı kolaylıkları vardır özellikle yeni bir projeye başlandığında mikroservis mimarisine göre yapılacak ön planlama çok daha azdır. Hızlı ilerleme kaydedilebilir. Uygulamanın dağıtımı (deployment) basittir. Monolit uygulamalarda debuggin çok daha kolaydır.



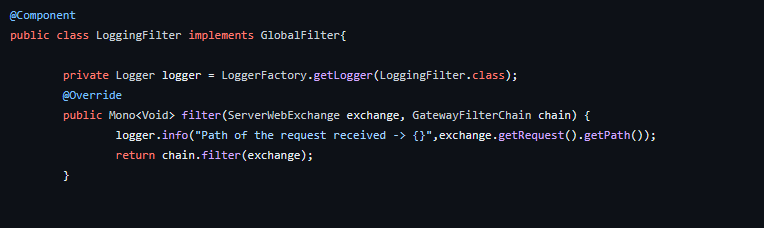
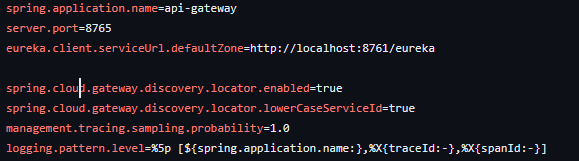
Mikroservis mimarisi, uygulamayı küçük, birbirinden bağımsız, ölçeklenebilir ve kendi işlevlerini bölen bir yapıdadır. Her mikroservis belirli bir işi yapar ve kendi veritabanları ve iş mantıkları vardır. Birlikte çalışırlar tabiki ancak genellikle ayrı olarak dağıtılır ve farklı seviyelerde ölçeklendirilir. Bir mikroservis python ile yazılmışken bir diğeri java ile yazılabilir böylece aslında bir özgürlük doğar. En büyük özelliği ise bu mimarinin her servis farklı ölçeklenebilir. Örnek vermek gerekirse, çok fazla istek alan bir e-ticaret sitesi düşünelim. İndirim yapılan bir döneme girildi ve normalde aldığı istekten çok daha fazlasını almaya başladı. Monolit bir uygulama olsaydı tüm istekler tek bir noktada toplanacağı için ölçeklenmesi çok zor olacaktı ve hızlı yanıt veremeyebilir ve hatta çökebilir. Yeni bir instance yaratabilir ancak mikroservis mimarisine göre kıyasladığımızda çok daha yüksek maliyete sahip olacaktır. Böyle bir indirim gününde hangi servislerin fazla yük alacağı bellidir ve örneğin üye olma ile ilgil olan servis sipariş ile ilgili olan servisten çok daha az istek alacağı aşikardır ve bu noktada gelen istekler de göz önünde bulundurularak hızlı şekilde ilgili servisler ölçeklenebilir. Tüm uygulamadan yeni bir instance oluşturmaktansa, bir servisten yeni bir instance oluşturmak çok daha maliyeti düşürür ve daha sağlıklı olur. Sonuçta ayağa kalkma zamanlarıda birbirinden çok daha farklı olacaktır.

1. API Gateway, Service Discovery, Load Balancer kavramlarını açıklayın.

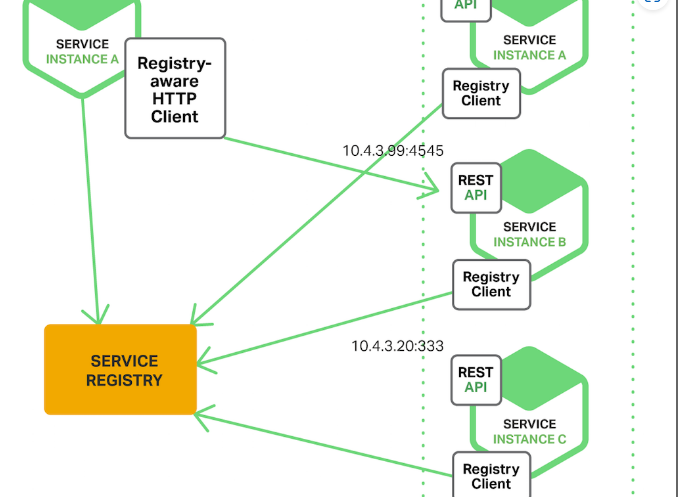
API Gateway, bir sistemdeki mikroservislerin, istemcilerin isteklerini alıp yönlendiren ve işleyen bir yazılım bileşenidir. API Gateway, gelen istekleri belirli mikroservislere yönlendirirken, yükü dengeleyebilir ve farklı mikroservis sunucuları arasında iş yükünü dağıtarak performansı artırabilir. Daha önce geliştirdiğim bir projede API Gateway kullanmıştım basit bir örnek olarak onu sağlayabilirim.



Bunun yanında eureka client kullanmıştım. Yine bu bir service discovery olarak geçiyor. Hizmet keşfi ve kaydı için kullanılır.



Özellikle mikroservis gibi uygulamalarda normal uygulamalara göre statik ip veya network lokasyonu yoktur. Servisler hata alır yenileri ayağa kalkar bu gibi durumlar için asıl kullanılır. İki tipi vardır. İlki client-side tasarım deseni. Bu desende ayakta olan serverların lokasyonundan ve aralarında yükü dengelemekten sorumlu olan clienttır.

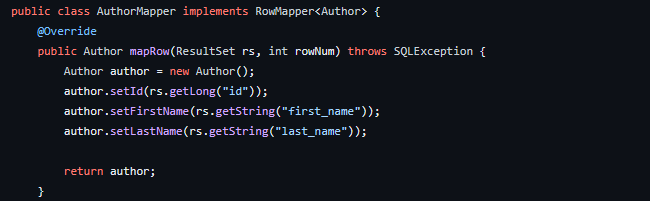
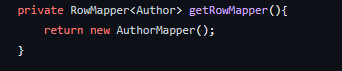


Server-side tasarım deseni ise genellikle istek load balancer’a gelir ve load balance ilgili servere/servise isteği yönlendirir. Load balancer ise adından da anlaşıldığı gibi yükü dengeler iki server olduğunu düşünelim tüm isteklerin bir servera gitmesini engeller. Fazla yük aldığı zaman bir server haliyle fazla CPU kullanımı olacaktır ve yavaş hatta çalışmaz duruma gelebilir bu ve bunun gibi durumları engellemek için kullanılır.

1. Hibernate, JPA, Spring Data framework’lerini örneklerle açıklayın.

Hibernate, ORM denilen (Object Relational Mapping) sağlayan bir çerçevedir. Bu çerçeve veritabanı tabloları ve Java sınıfları arasında ilişki kurmayı ve verileri veritabanı ile etkileşim kurmak için nesne odaklı bir yaklaşım sağlar. Sadece veritabanı tabloları ile Java sınıflarını eşleştirmekten sorumlu değildir. Veritabanına erişim ve sorgulama işlemlerini de kolaylaştırır. Örneğin SQL sorguları yazmadan veritabanı işlemleri gerçekleştirebiliriz Hibernate sayesinde. Hangi kolon’u hangi java objesinin attribute’u ile eşleşeceğini ise kullanılan annotationlarla bilir. Hibernate aslında jdbcTemplate ile yapılanları son kullanıcıdan soyutlayıp mapplemesine kadar yapar.





Bu ve bunun gibi işlemleri yapmaktansa, SessionFactory ile bir session kurulup get metodu ile sınıf tipi ve id verilerek çok basit bir şekilde yapılabilir. Yada JPA kullanılabilir.

Java Persistence API ORM standartıdır. İlişkisel veritabanları ile etkileşimi kolaylaştırmak için arayüz sağlar ve bu sayede veritabanındaki verileri Java nesneleri olarak temsil eder. Tamamen veritabanı işlemlerini gerçekleştirmek için daha yüksek seviyede API sağlar. SQL sorguları yazmaya gerek kalmadan işlemler gerçekleştirilebilir. Bileşenleri arasında JPQL bulunur. Örnek vermek gerekirse sql ile yazılan bir sorgu ile arasında çok farkı yoktur.

SELECT \* FROM Author WHERE last\_name = Baturay'; -> SQL

SELECT a FROM Author a WHERE a.lastName = Baturay; ->JPQL

Spring Data ise veritabanı için bir arayüz sağlar. CRUD işlemlerini basitleştrir. Arayüzünde findByID, save vs. Gibi hazır fonksiyonlar bulunur. Sorgu metodları için dinamik sorgular bulunur. Örneğin findByLastName(String lastName) gibi bunu çözümler ve soyadına göre bir sorgu gönderir. Sayfalama ve sıralama gibi işleri kolaylaştıran API’ler sağlar.

Yine @Query anatasyonu kullanılarak özgü sorgular yazılablir.