1) Senkron ve Asenkron iletişim nedir örneklerle açıklayın?

Senkron programlama ile geliştirilen bir yazılımda kodun veya sürecin ilerlemesi için birbirini takip eden işlemlerin bitmesi beklenir. Örneğin Quicksort gibi bir sıralama algoritmasında elimizdeki sayıların önce bir yarısında sonra da diğer yarısında sıralama yaparız. Burada aynı anda sayıların iki yarısını da sıralamayız. Gerçek hayattan bir örnek vermek gerekirse de sıralar ve kuyruklar örnek verilebilir. Önümüzdeki herkesin işlemi bitmeden biz işlem yapamayız.

Asenkron programlama ile geliştirilen kodda ise kod parçacıkları dağıtık yazılır ve birbirlerini beklemeden sadece bir çağrı ile çalışırlar. Bilgisayarımıza aynı anda iki dosya birden indirebilmemiz buna bir örnek olabilir. Gerçek hayatta ise bir lokantada kimse bir başkasının yemeğinin bitmesini beklemeden sipariş verebilir.

2) RabbitMQ ve Kafka arasındaki farkları araştırın?

Kafka, açık kaynaklı platformdur ve ham veri akışını kolaylaştırır. Java ve Scala ile yazılmış olan Kafka, verileri yeniden sunma odaklı bir publisher/subscriber mesaj kuyruğudur. Kafka mesajları belleğine ekler ve orada tutar, subscriber okuyana veya saklama sınırına ulaşana kadar da orada tutar.

Kafka, kullanıcılara belirli limitler dahilinde mesaj paketleri talep etmelerine izin veren bir "pull based" kullanım ortamı sağlar. Kullanıcılar, daha yüksek veri akışı ve etkili mesaj Kafka'yı kullanırlar

RabbitMQ ise daha karmaşık yönlendirme projelerinde verimli mesaj iletimini kolaylaştıran açık kaynaklı bir dağıtık mesaj kuyruğudur. "Dağıtık" olarak adlandırılmasının sebebi, RabbitMQ'nun genellikle node yapısı ile çalışması ve kuyrukların nodelar arasında dağıtıldığı, yüksek erişilebilirlik ve hata toleransı için, bir yapıda çalışmasıdır.

RabbitMQ, bir push modeli kullanır ve tüketiciler için ayarlanmış ön yükleme limitiyle kullanıcıları korur. Bu yapı, düşük gecikmeli mesaj iletişimi için kurulmuştur. Ayrıca, RabbitMQ queue mantığı ile iyi çalışır. RabbitMQ, bir postaneye benzetilebilir, yani posta alır, depolar ve teslim eder, Kafka ise ikili veri mesajlarını kabul eder, depolar ve iletiler.

RabbitMQ, AMQP 1.0, HTTP, STOMP ve MQTT gibi ek protokolleri destekler. Kafka ise video yayınları gibi projelerde daha yaygın kullanılır.

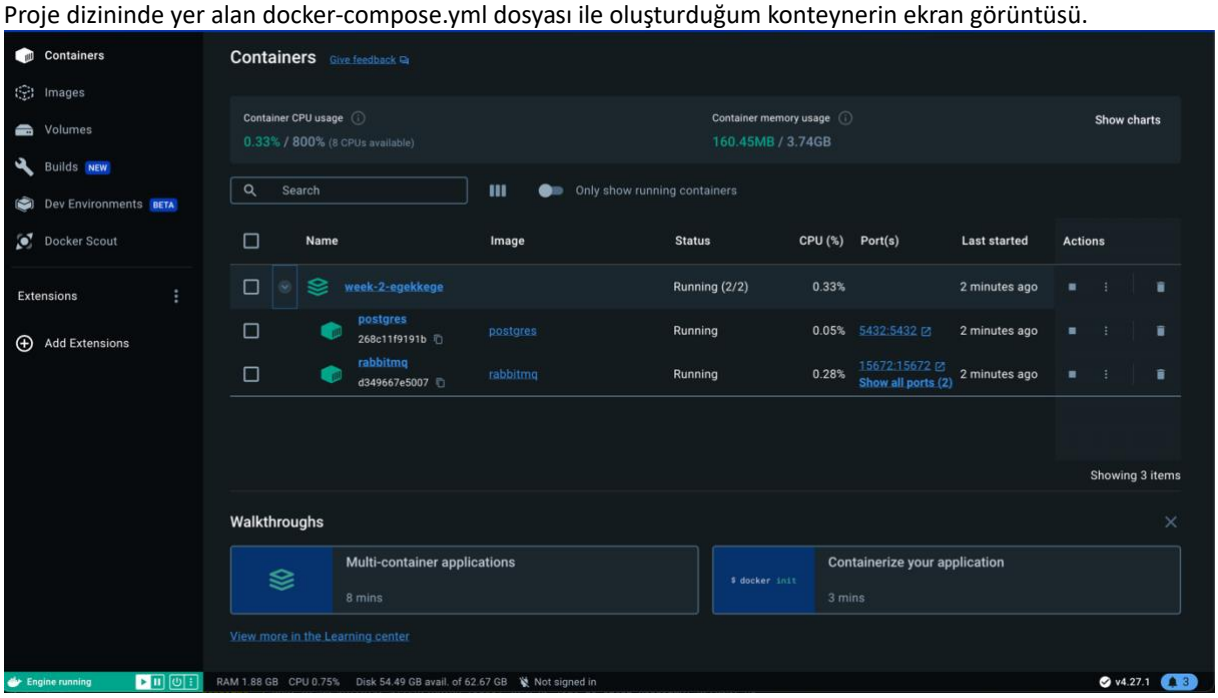
3) Docker ve Virtual Machine nedir?

Virtual machine tıpkı gerçek bir bilgisayar gibi giriş/çıkışları olan, ram ve işlemcisi , sanal olarak verilmiş olsa da, olan kendi içinde tamamen bir işletim sistemi çalıştırabilen sanal bir bilgisayardır. Gerçekten de bilgisayarımızın içinde bir bilgisayar daha varmış gibi düşünülebilir.

Docker ise VM'in aksine gerçek bir bilgisayarı taklit etmez. Konuk olduğu işletim sisteminin kernelini kullanarak sadece kendisine atanmış özellikleri gösterir. Örneğin bir java projesi için docker kullanacaksak, ilgili Docker imajında sadece java projemizi çalıştırmak için gerekli özellikleri ve programları tanımlayabiliriz. Docker'ın avantajı ise, projeyi başarılı bir şekilde üzerinde çalıştırdıktan sonra Docker konteyneri oluşturup projemizi Docker kurulu olan her yerde çalıştırabiliriz. Ama benim bilgisayarımda çalışıyordu gibi problemlerden kurtuluruz. Konteynerler tıpkı gerçek konteynerler gibi sadece bizim için gerekli olan programları ve kodları içinde tutan dosyalardır. Onları Docker engine ile çalıştırmak mümkündür.

4) Docker ile RabbitMQ ve PostgreSQL ve ya MySQL kurulumu yapın?

Proje dizininde yer alan docker-compose.yml dosyası ile oluşturduğum konteynerin ekran görüntüsü.



5) Docker komutlarını örneklerle açıklayın

- Run: Konteynerleri başlatmak ve başlatırken konfigürasyonları girmeyi sağlar.
  docker run -d -p 8080:80 nginx // -d konteyneri detached modda arkaplanda çalıştırır. -p host ve konteyner arasındaki port geçişini sağlar, 80 portundan 8080 portuna

- Pull: Konteynerleri çalıştırmadan önce ilgili konteyner imajını çekmek için kullanılır.
  docker pull ubuntu:latest // Son sürüm ubuntu docker imajını indirir.

- PS: Aktif konteynerleri ve onların isim, ID, durumları ve port numaralarını listeler.
  docker ps -a // -a komutu kapalı konteynerleri de listeler

- Start ve Stop: konteynerin adı ile birlikte girildiğine ilgili konteyneri başlatır ya da durdurur.

- Logs: konteynerin adı ile girildiğinde ilgili konteynerin loğlarını gösterir.

- Exec: Çalışan bir konteynerin içinde komut çalıştırmaya yarar.

  docker exec {konteyner adı} bash // konteynerde bash Shell açılır.

- Build: Kendi docker imajımızı oluşturmak için kullanılır.

  docker build -t imaj_adı

- Images: Hostta hazır bulunan imajları listeler.

- RMI: İmajları silmek için kullanılır. Yanına -imaj_ismi eklenirse sadece o imajı siler.

- Network: Docker kendi içinde ayrı bir network oluşturabilir. Bu network'ü kontrol etmek için kullanılır. Ls, create, add gibi ilave komutlarla kullanılır .

6) Microservice ve Monolith mimarilerini kıyaslayın

- Monolit programlarda tüm uygulama tek elden yazılır ve yönetilir. Bütün parçalar birbirine bağlıdır ve işlevselliğe direkt olarak katılırlar. Mikroservis mimari ile hazırlanmış programlar ise küçük ve birbirine bağımlılığı olmayan parçalardan oluşur. Her bir parça kendi veritabanı ile kendi işlevselliğinden sorumludur.
- Mikroservis mimarinin en öne çıkan avantajı ölçeklenebilir olmasıdır. Her bir servis talep arttıkça daha fazla kaynakla beslenebilir ve gerekli noktalarda en yüksek performansı gösterir.
- Mikroservis mimarisinde her bir servis başka teknolojiler ve programlama dilleri ile geliştirilebilir. Bu esneklik sayesinde bir gereksinim için en doğru dil ve teknoloji seçilebilir. Öte yandan bu esneklik monolitik mimariye kıyasla daha zor bir geliştirme sürecine sebep olur.
- Monolitik programlar, bir önceki maddede belirttiğim üzere, daha hızlı geliştirilebilr. Tek bir dil kullanılarak geliştirme yapılacağı için daha küçük takımlar daha hızlı geliştirmeler ve testler yapabilir.
- Programın belirli bölümlerinde talep dalgalanmaları veya değişken gereksinimler olacaksa mikroservis yapısı daha uygunken, görece küçük uygulamalar ve daha hızlı geliştirme süreçleri için monolitik yapılar daha uygundur.

7) API Gateway, Service Discovery, Load Balancer kavramlarını açıklayın

- Load balancer: Bir uygulamaya gelen istekleri karşılayan sunucudur. Özellikle yoğun talep olan durumlarda gelen istekleri belirli bir algoritmaya göre (round robin ya da sistem kaynaklarının gücüne göre bazı sunuculara 1 istek gönderirken bazılarna 3 istek göndermek gibi) uygulamanın sunucularına dağıtma görevini üstlenirler. Benzer şekilde talebin geldiği ülkelere göre istekleri dağıtabilirler ya da engelleyebilirler.
- API Gateway: Mikroservis mimarisi ile geliştirilmiş uygulamalarda hizmetler arası bağlantıyı veya gelen istekleri düzenlemek için kullanılan sunucudur. Mikroservis mimarisi ile hazırlanan uygulamalarda, hizmetler arasında iletişimi sağlarken bazı dönüştürme ve biçimlendirme görevlerini de üstlenebilir.
- Service Discovery: API gateway mikroservise dışarıdan gelen isteklerde daha fazla kullanılırken, Service Discovery mikroservislerin birbiri ile iletişimini IP adresi gibi bilgiler olmadan sağlar.

8) Hibernate, JPA, Spring Data framework'lerini örneklerle açıklayın

- JPA, Java Persistence API, Java ile geliştirilen programlarda ilişkisel veritabanları ile ,Object Relational Mapping mantığı ile , iletişimi sağlayan kurallar bütünü olarak özetlenebilir.
- Hibernate, JPA kullanarak veri tabanlarını uygulamanın ihtiyaçlarına göre yönetmek için kullanılan bir frameworktür. Int, String gibi değişken türleri için veri tabanında ilgili tabloları yönetir.
- Spring data, spring'in bir alt teknolojisidir. JPA kullanarak veri tabanlarını yönetir. Spring data ile veri tabanı işlemleri yapmak için interface'ler kullanılır.