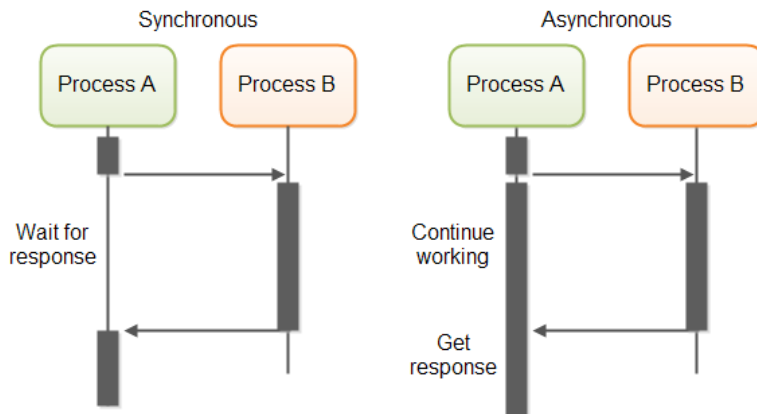


Soru1-Senkron ve Asenkron iletişim nedir örneklerle açıklayın?

Senkron iletişimde, programlar birbirleriyle etkileşim halindeyken eşzamanlı olarak çalışır. Bu, bir programın diğerinden bir yanıt beklerken, diğer programın isteği karşılamak için hazırda beklemesini ifade eder. İletişim gerçekleşene kadar her iki taraf da bekler, örneğin bir kullanıcı bir düğmeye tıkladığında ve diğer işlemler ilgili işlem tamamlanana kadar bekler. Bu, senkron iletişimi örneklendirir ve tüm tarafların belirli bir süreci tamamlaması gereken işbirliği gerektiren durumlar için faydalıdır.

Buna karşılık, asenkron iletişim, programların birbirlerine bağımlı olmadan işlemlerini bağımsız olarak gerçekleştirdiği bir süreci ifade eder. Bir program bir istek gönderir ve diğer program isteği aldığı anda işlemlerine devam eder. Taraflar, iletişim sırasında birbirlerini beklemek zorunda değildir, örneğin bir kullanıcı bir web sayfasındaki bir düğmeye tıkladığında ve tarayıcı sayfayı yenilemeden önce sunucudan yanıt beklemeksizin isteği sunucuya gönderir. Bu asenkron iletişim, kullanıcıların tarayıcıda aynı anda diğer işlemleri yapmasına olanak tanır. İşbirliği gerektirilmeyen durumlar ve tarafların bağımsız olarak çalışabileceği durumlar için uygundur.

Her iki iletişim yöntemi de farklı amaçlar için hizmet eder ve programlar arasında etkili işbirliğini kolaylaştırır. Senkron iletişim, taraflar arasında belirli bir sıra veya düzen gerektiren senaryolar için idealdir, asenkron iletişim ise tarafların bağımsız olarak işlem yapmasına izin verir. Sonuç olarak, her iki yöntem de programlar arasında daha iyi işbirliği ve performans sağlar.



Hocamın da derste verdiği örnekteki gibi asenkron çalışan bir fırında bir müşteri almak istediği ekmek için sırada beklemek yerine başka işleri varsa halleder ve fırıncıda bir sorun çıkması durumunda mağdur olup onca zaman sırada beklemesin. Senkron çalışmayı daha çok birbiri ardına gelen ilişkili işlerin yapılması olarak yorumluyorum. Mesela bir yemeğin hazırlanması gibi malzemeler doğranmadan kızartılmaz yahut başka işlere girilmez her şey

sırayladır

Soru2-RabbitMQ ve Kafka arasındaki farkları araştırınız?

RabbitMQ ve Apache Kafka arasında seçim yapmak zorlu bir karar olabilir, çünkü her platformun benzersiz güçlü ve zayıf yönleri vardır. Bir tercih yapmadan önce proje gereksinimlerinizi dikkatlice değerlendirmek önemlidir.

RabbitMQ'da mesajlar, bir kuyrukta bekleyen mektuplara benzerken, Kafka'da bunlar bir günlüğe kaydedilen girişlere benzer. RabbitMQ, AMQP ve MQTT gibi çeşitli dilleri (protokoller) desteklerken, Kafka kendi benzersiz diyalektiğine sahiptir.

Mesajların işlenmesi konusunda, RabbitMQ bir kez teslimat hizmeti gibi işlerken, Kafka gerektiğinde mesajların yeniden oynatılmasına izin veren bir makbuz sağlar.

Mesaj yönlendirme açısından, RabbitMQ doğru teslimatı sağlayan etkili bir postacı gibi işlev görürken, Kafka mesajları farklı kutulara kategorize ederek organizasyon sağlar.

RabbitMQ acil teslimatları önceliklendirerek mesaj öncelik seviyeleri atayabilirken, Kafka her mesajı eşit olarak işler, tek sıralı bir kuyrukta beklemeye benzer.

RabbitMQ ve Kafka, güvenli mesaj iletimini garanti etse de, biraz farklı onay yöntemleriyle çalışırlar.

RabbitMQ hafif ve çevik kuyrukları tercih ederken, Kafka ağır yükleri kolayca yönetmek için tasarlanmıştır.

Ölçeklendirme durumunda, RabbitMQ artan trafiği daha fazla yardımcı ekleyerek karşılayabilir, zirve saatlerinde ek kasacılara sahip olmaya benzer. Diğer yandan, Kafka sorunsuz trafik akışı için otoyola daha fazla şerit ekleyerek genişler.

RabbitMQ ve Kafka'nın farklı lisans anlaşmaları olsa da, her iki platform da açık ve ücretsiz olarak kullanılabilir.

RabbitMQ kullanıcı dostu ve güvenilirdir, güvenilir bir arkadaşına benzer, Kafka ise karmaşıklıklarını anlamak için daha fazla zaman ve çaba gerektirebilir.

Mesajlaşma yeteneklerinin yanı sıra, Kafka, RabbitMQ'nun daha basit yaklaşımına kıyasla kapsamlı bir araç kutusu sunarak çeşitli ek özellikler sunar.

Soru3-Docker ve Virtual Machine nedir?

Sanal Makineler (VM'ler):

Sanal makineleri, fiziksel bilgisayarların dijital kopyaları olarak düşünebilirsiniz. Her sanal makine, kendi işletim sistemi ve yazılımı ile çalışır, gerçek bir bilgisayar gibi.

Bir sanal makine oluşturmak, geniş miktarda depolama ve belleği tüketebilecek tam bir işletim sistemi kurulumunu içerir.

Sanal makineler, tek bir fiziksel bilgisayarda farklı işletim sistemlerini çalıştırmanızı gerektiğinde faydalıdır. Her sanal makine, kendi bağımsız ortamını kurar ve diğerlerinden ayrı tutulur.

Docker:

Docker, bunun aksine, yazılımı ve bağımlılıklarını paketlemek için sihirli bir konteyner gibi davranır. Bu paketlere "konteyner" adı verilir ve aynı işletim sistemini paylaştıkları için son derece hafiftirler.

Docker ile bir uygulamayı çalıştırdığınızda her seferinde tam bir işletim sistemi kurmanıza gerek yoktur. Bunun yerine, uygulamanın tüm gerekli bileşenlerini bir konteynıra kapsülleyerek, taşınabilir ve herhangi bir makinede kolayca çalıştırılabilir hale getirirsiniz. Docker, uygulamaları çeşitli ortamlarda tutarlı bir şekilde geliştirmeyi ve dağıtmayı kolaylaştırır.

Benzerlikler:

Hem Docker konteynerleri hem de sanal makineler, yazılım ortamları oluşturmak için şablon görevi gören görüntülerden oluşturulur.

Her ikisi de zaman içinde yapılan değişiklikleri izlemenizi sağlar, güncellemeleri ve yapılandırmaları etkili bir şekilde yönetmeyi kolaylaştırır.

Docker'ı veya sanal makineleri tercih etseniz de, uygulamalarınızı çeşitli donanım tiplerinde çalıştırabilirsiniz, karmaşıklık yaşamadan.

Ne Zaman Hangisini Kullanmalı?

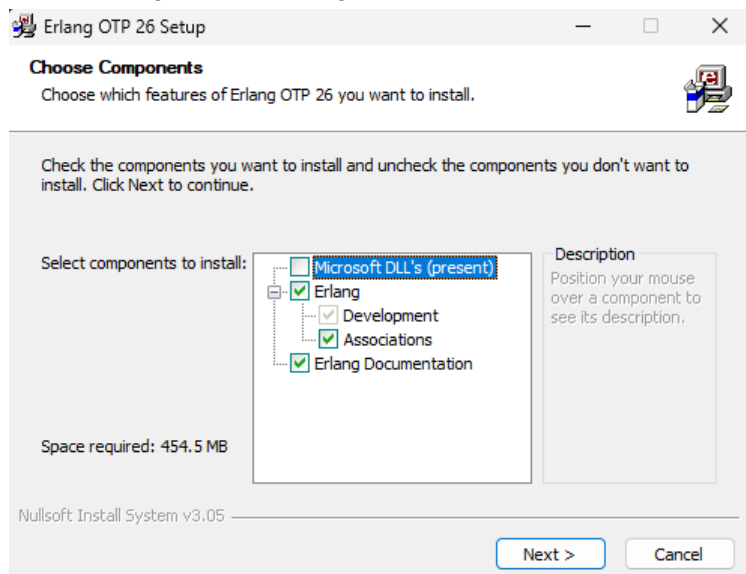
Sanal makineler, belirli işletim sistemi yapılandırmaları veya çeşitli donanım kurulumları gerektiren durumlar için idealdir.

Docker, hafif, taşınabilir ve kolayca ölçeklenebilen çözümler aradığınız senaryolarda, özellikle sık güncellemeleri olan uygulamalar veya mikro hizmet mimarisi için mükemmel bir seçenektir.

Özetle, sanal makineler tamamen izolasyon ve daha ağır yapılandırmalar sunarken, Docker yazılım uygulamalarını paketleme ve dağıtma konusunda esneklik, hız ve verimlilik sağlar. Karar, ihtiyaçlarınıza ve projelerinizin özelliklerine en uygun olanına göre verilir.

Soru4-Kurulumlar -RabbitMQ -Docker -PostgreSQL -MySQL

RabbitMQ için önce erlangı kurdum

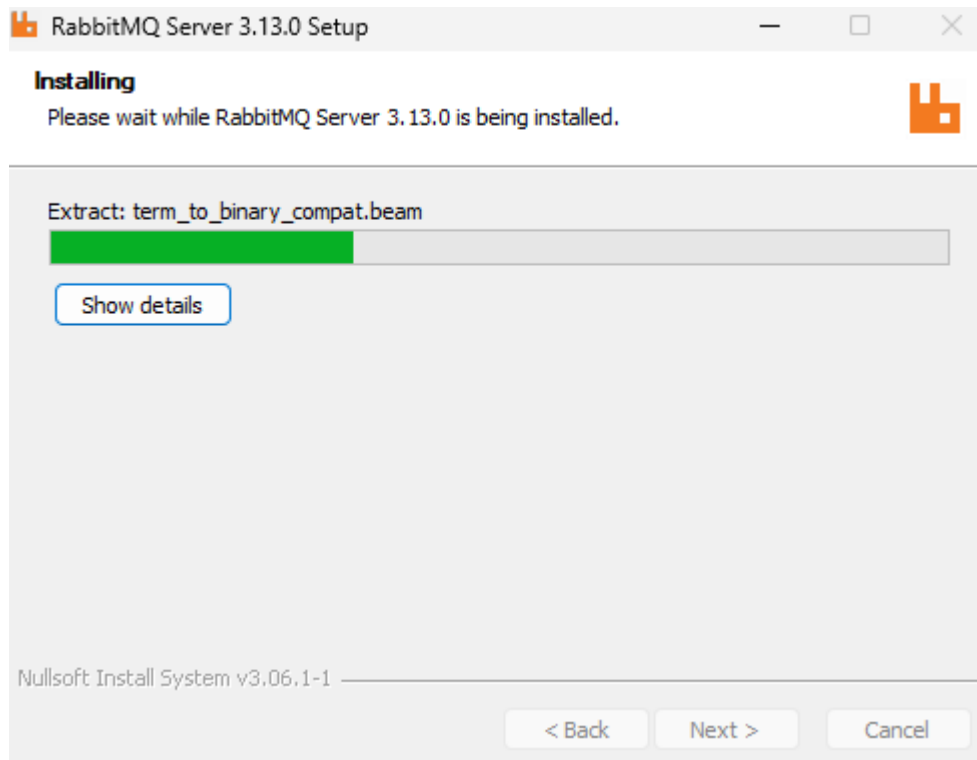


bu da Erlang'ın terminali

```
Erlang/OTP 26 [erts-14.2.3] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:1] [jit:ns]

Eshell V14.2.3 (press Ctrl+G to abort, type help(). for help)
1> 1+1.
2
2>
```

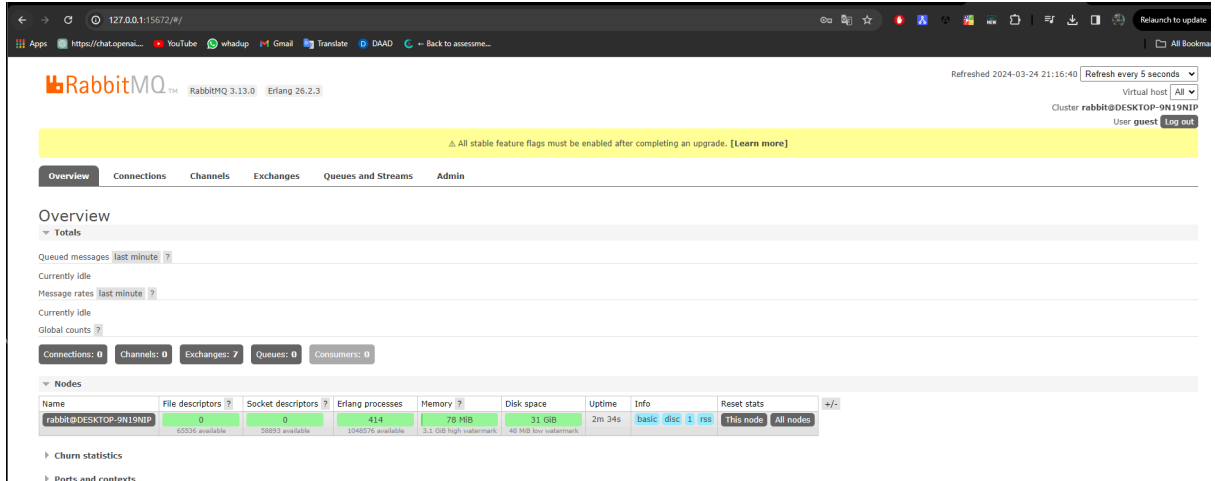
sonra RabbitMQ installerı indirip çalıştırdım.



sonrasında rabbitmq_managementı kullanılabılır yaptım

```
[ ] rabbitmq_federation 3.13.0
[ ] rabbitmq_federation_management 3.13.0
[ ] rabbitmq_jms_topic_exchange 3.13.0
[E] rabbitmq_management 3.13.0
[e] rabbitmq_management_agent 3.13.0
[ ] rabbitmq_mqtt 3.13.0
[ ] rabbitmq_peer_discovery_aws 3.13.0
[ ] rabbitmq_peer_discovery_common 3.13.0
[ ] rabbitmq_peer_discovery_consul 3.13.0
[ ] rabbitmq_peer_discovery_etcd 3.13.0
[ ] rabbitmq_peer_discovery_k8s 3.13.0
[ ] rabbitmq_prometheus 3.13.0
[ ] rabbitmq_random_exchange 3.13.0
[ ] rabbitmq_recent_history_exchange 3.13.0
[ ] rabbitmq_sharding 3.13.0
[ ] rabbitmq_shovel 3.13.0
[ ] rabbitmq_shovel_management 3.13.0
[ ] rabbitmq_stomp 3.13.0
[ ] rabbitmq_stream 3.13.0
[ ] rabbitmq_stream_management 3.13.0
[ ] rabbitmq_top 3.13.0
[ ] rabbitmq_tracing 3.13.0
[ ] rabbitmq_trust_store 3.13.0
[e] rabbitmq_web_dispatch 3.13.0
[ ] rabbitmq_web_mqtt 3.13.0
```

burdada 127.0.0.1:15672 portunda RabbitMQ'yu başlattım.

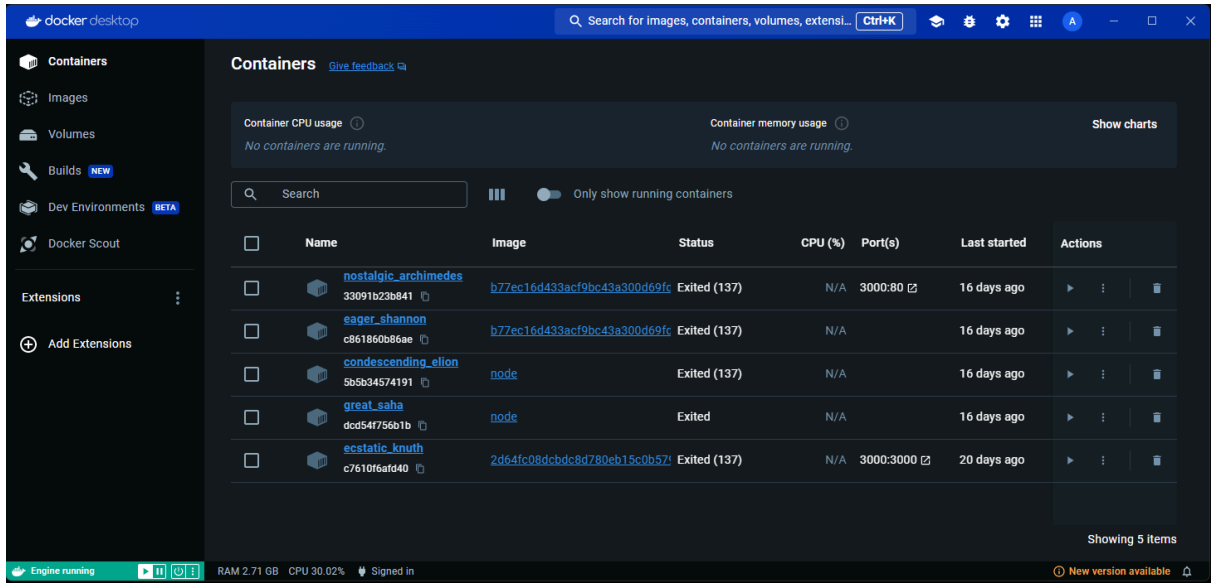


The screenshot shows the RabbitMQ Management UI at 127.0.0.1:15672. The interface is in English and shows the following details:

- Version: RabbitMQ 3.13.0, Erlang 26.2.3
- Refreshed: 2024-03-24 21:16:40
- Virtual host: All
- Cluster: rabbit@DESKTOP-9N19NIP
- User: guest
- Log out button
- Navigation tabs: Overview, Connections, Channels, Exchanges, Queues and Streams, Admin
- Overview section:
 - Totals:
 - Queued messages: last minute
 - Currently idle
 - Message rates: last minute
 - Currently idle
 - Global counts
 - Connections: 0, Channels: 0, Exchanges: 7, Queues: 0, Consumers: 0
 - Nodes table:

Name	File descriptors	Socket descriptors	Erlang processes	Memory	Disk space	Uptime	Info	Reset stats
rabbit@DESKTOP-9N19NIP	0	0	414	78 MiB	31 GiB	2m 34s	basic disc 1 vss	This node All nodes
 - Churn statistics
 - Ports and contexts

Docker'i önceden kullanmaktaydim hatta mevcutta node için oluşturulmuş bir containerım mevcut

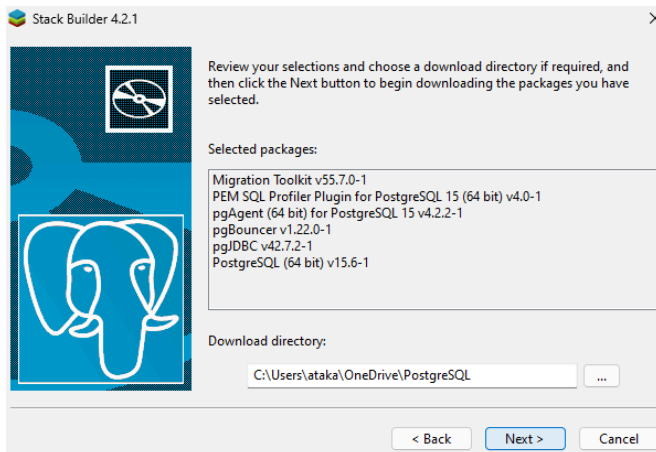


The screenshot shows the Docker Desktop interface. The left sidebar contains the following menu items: Containers, Images, Volumes, Builds, Dev Environments, Docker Scout, Extensions, and Add Extensions. The main area displays the 'Containers' section with a search bar and a toggle for 'Only show running containers'. A table lists the following containers:

Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
nostalgic_archimedes	b77ec16d433acf9bc43a300d69fc	Exited (137)	N/A	3000:80	16 days ago	
eager_shannon	b77ec16d433acf9bc43a300d69fc	Exited (137)	N/A		16 days ago	
condescending_elion	node	Exited (137)	N/A		16 days ago	
great_saha	node	Exited	N/A		16 days ago	
ecstatic_knuth	2d64fc08dc8dc8d780eb15c0b57	Exited (137)	N/A	3000:3000	20 days ago	

The bottom status bar shows: Engine running, RAM 2.71 GB, CPU 30.02%, Signed in, and a New version available notification.

PostgreSQL'in installerını sitesinden indirdim

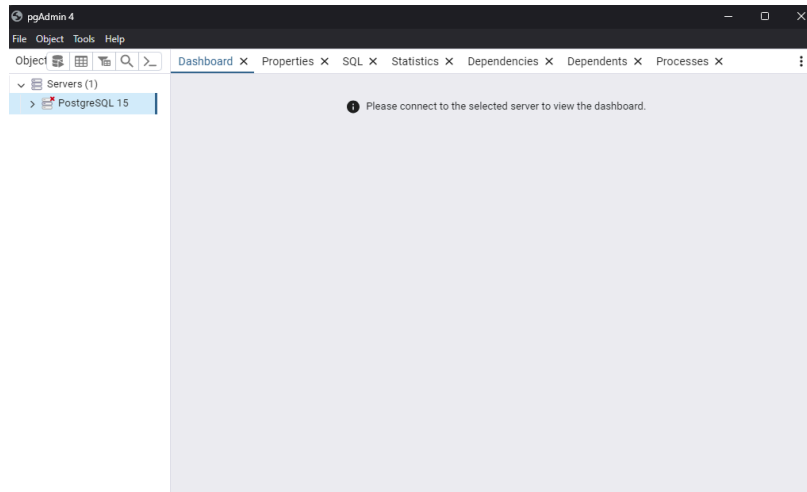


The screenshot shows the Stack Builder 4.2.1 window. It displays a list of selected packages for PostgreSQL installation:

- Migration Toolkit v55.7.0-1
- PEM SQL Profiler Plugin for PostgreSQL 15 (64 bit) v4.0-1
- pgAgent (64 bit) for PostgreSQL 15 v4.2.2-1
- pgBouncer v1.22.0-1
- pgJDBC v42.7.2-1
- PostgreSQL (64 bit) v15.6-1

The download directory is set to C:\Users\ataka\OneDrive\PostgreSQL. The window includes a 'Next >' button and a 'Cancel' button.

Sonrasında pgAdmin appini açtım



Soru5-Docker komutlarını örneklerle açıklayınız

1-docker --version:

Docker'ın şu anda yüklü olan sürümünü gösterir.

\$ docker --version

```
C:\Users\ataka>docker --version
Docker version 25.0.3, build 4deb41
```

2-docker pull:

docker pull <image name>

Docker reposundan imajları çekmek için kullanılır websitesini ziyaret edip ne tür imajlar var görüntüleyebiliriz.Daha önce node'u pullamıştım



\$ docker pull node

3-docker run:

docker run -it -d <image name>

Docker imajından yeni bir container oluşturur ve çalıştırır

\$ docker run -it -d ubuntu

4-docker ps:

Mevcutta çalışan container ları gösterir -a eklenirse de hepsini gösterir

docker ps -a

5-docker exec:

docker exec -it <container id> bash

Çalışmakta olan bir containerda bir komut çalıştırır

Örneğin

\$ docker exec -it own_container bash

6-docker stop:

Çalışan bir containerı durdurur.Sonuna id veririz: docker stop <container id>

7-docker kill:

docker kill <container id>

Id si verilen containerı siler (yokulur).

```
$ docker kill owned_container
```

8-docker commit:

docker commit <container id> <username/imagename>

Değiştirilmiş bir containerden yeni bir image oluşturur.

```
$ docker commit my_container my_username/my_image
```

9-docker login:

```
docker login
```

Docker Hub registerına giriş yapar.

10-docker push:

Docker Hub reposuna verilen bir image'ı yollar.

```
$ docker push atkan/atkn_image
```

11-docker images:

Localdeki tüm Docker imageler'ı listeler.

```
$ docker images
```

12-docker rm:

Durmuş bir containerı siler(sonuna id vermeliyiz)

```
$ docker rm atkn_container
```

13-docker rmi:

Localden idsi verilen bir image'ı siler

```
$ docker rmi my_image_id
```

14-docker build:

Belirlenmiş bir Dockerfile'dan bir Docker image'ı build eder.

```
$ docker build .
```

15-docker copy:

Docker image oluşturma sırasında dosyaları veya dizinleri ana makinenin dosya sisteminden container'ın dosya sistemine kopyalar.

Komut: COPY <source_file> <destination_directory>

16-docker history:

Geçmişteki logları gösterir ve image ismi eklenirse yanına o image ait olanları verir.

Komut: docker history <image_name>

```
$ docker history
```

```
$ docker history my_image
```

17-docker logout:

Docker registry'e kimlik doğrulamak için kullanılan kimlik bilgilerini siler veya oturumu kapatır.

```
$ docker logout
```

18-docker logs:

Bir container için logları gösterir.

Komut: docker logs <container_id>

19-docker network:

Containerla iletişimde olan Docker networklerini yönetir

Komut: docker network create <network_name>

```
$ docker network create my_network
```

20-docker restart:

Bir ya da birden fazla docker containerını yeniden başlatır

Komut: docker restart [OPTIONS] CONTAINER [CONTAINER...]

```
$ docker restart atakan_container
```

21-docker volume:

Bir volume yaratır Docker'daki volumeler container lardan bağımsız olarak oluşturulur.

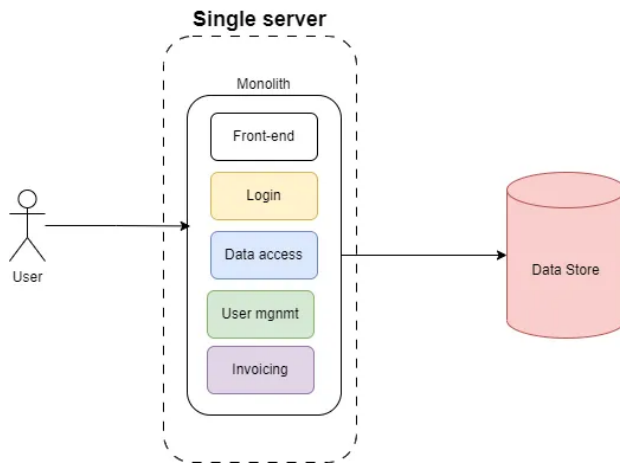
Komut: `docker volume create <volume_name>`

`$ docker volume create my_volume`

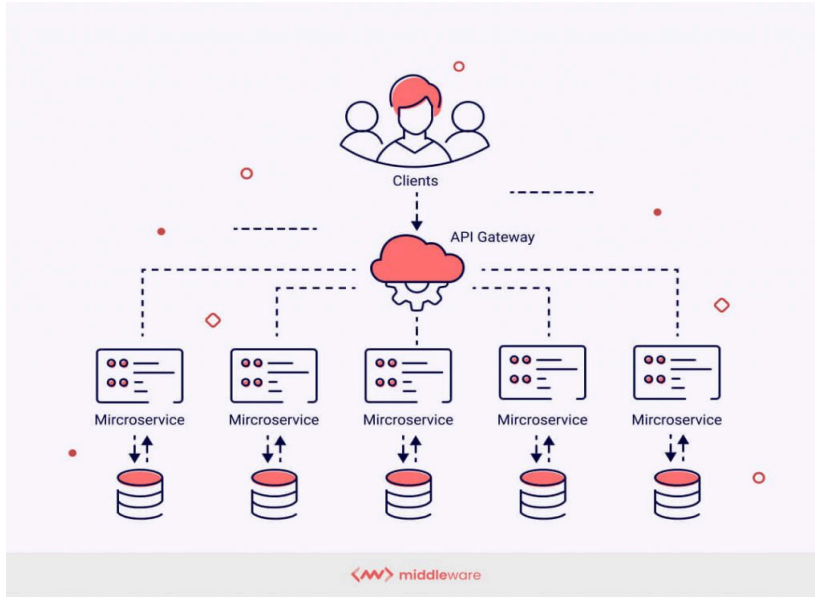
Soru6-Microservice ve Monolith mimarileri açıklayınız

İlk olarak, Monolitik Mimariyi tartışalım. Her şeyi içine alan büyük, sağlam bir kod bloğu oluşturduğunuz hayal edin - kullanıcı arayüzü, iş mantığı ve database. Bunu yapmak, bütün meyveleri aynı sepete koymak gibidir. Başlangıçta, Monolitik uygulamalar her şeyin tek bir yerde oluşturulup çalıştırılması edilmesi nedeniyle geliştirmesi kolaydır. Ancak, genişledikçe karmaşıklık da artar. Bunlar, temel web siteleri veya bloglar gibi daha küçük projeler için iyi çalışır. WordPress veya Django gibi platformlar Monolitik mimarinin klasik örnekleridir.

Mikroservis Mimarisinde ise, uygulamanızı daha küçük, bağımsız bileşenlere olan hizmetlere parçalamanızı içerir. Her bir hizmet belirli bir görevi yönetir ve diğerleri ile belirli bağlantılar aracılığıyla iletişim kurar. Bu, istediğiniz gibi çoğaltıp budaklandırabileceğimiz yeniden düzenleyebileceğiniz bir yapboz gibidir. Mikroservisler, ölçeklenebilirlik, esneklik ve hızlı değişikliklere uyum sağlama konusunda mükemmel performans gösterir. Netflix, Amazon ve Uber gibi önemli şirketler bu sistemi kullanır.



Temel ayrımları ise bir Monolitik kurulumda, tek bir büyük kod tabanınız vardır, oysa Mikroservisler size daha küçük, bireysel kod tabanları sağlar. Ölçeklendirme söz konusu olduğunda, Monolitik uygulamalar bir bütün olarak ölçeklenirken, Mikroservisler her bir hizmeti bağımsız olarak ölçeklendirmenize olanak tanır. Geliştirme açısından, Monolitler hızlı başlangıç sağlarken, Mikroservisler projeniz evrildikçe daha fazla esneklik ve uzmanlaşma sunar. Ayrıca, dağıtma zamanı geldiğinde, Mikroservisler daha hassas kontrol sağlar - bir hizmeti diğerlerini etkilemeden güncelleyebilirsiniz, Monolitlerde ise tüm sistem aynı anda dağıtılır.

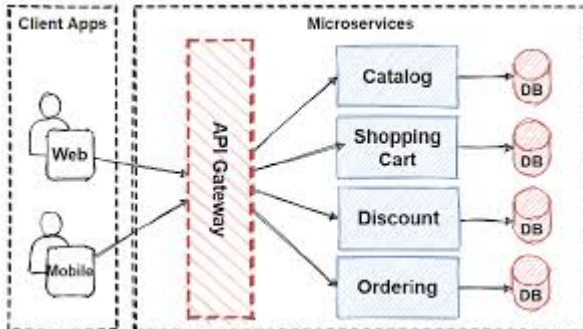


(Ornekte görüldüğü gibi sistem büyüdükçe büyüyen ve yönetimi ve bağımlılıklar kontrolü zorlaşan bir microservis mimarisi)

Bu mimariler arasında karar verirken, proje büyüklüğü, ekip yapısı ve teknolojik gereksinimler gibi faktörler göz önünde bulundurulmalıdır. Her ikisinin de kendi zorlukları vardır, ancak uygun planlama ve en iyi uygulamalara uyum sağlanmasıyla bunların üstesinden kolaylıkla gelebilirsiniz. Biri diğerinden daha iyi denilmesi doğru değildir ihtiyaçlar göz önünde bulundurulmalıdır ve temelde bir takım avantajlar devamında dezavantajları da getirmektedir burada da microservislerde her bir servis için bir db bulunması ve birbirleriyle iletişim kurması durumlarının yönetimi oldukça effort isteyen süreçlerdir.

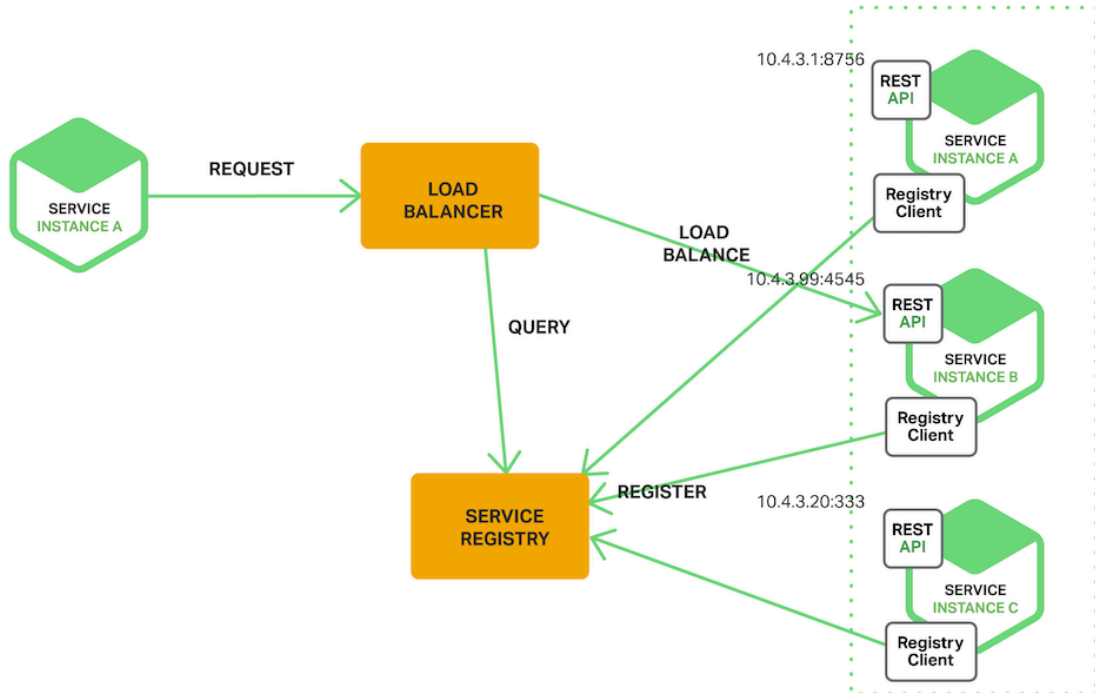
Soru7-API Gateway,Service Discovery,Load Balancer nedir?

Bir **API Gateway**, mikroservis mimarisinde istemciler ile sunucular arasında köprü görevi görerek, isteklerin akışını kolaylaştırır. Trafik kontrolörü gibi hareket eder, isteklerin uygun bir şekilde yönetilmesini ve doğru hedefe yönlendirilmesini sağlar. Bununla birlikte, API Gateway, kimlik doğrulama, yetkilendirme ve hız sınırlama gibi önemli görevleri de yerine getirir. Özetle, derinlemesine teknik bilgi gerektirmeksizin çeşitli hizmetlere erişim için merkezi bir nokta sunarken, aynı zamanda sistemdeki güvenlik ve işletim verimliliğini izler ve yönetir. Aşağıdaki görselde de gösterildiği gibi farklı clientlar ve servisler arasında bir protokol görevi görür.



Service Discovery, mikroservis dünyasında otomatik olarak mevcut hizmetleri tanımlayarak kritik bir rol oynar. Bir kalabalık içerisinde arkadaşları bulmak gibi hızla görünebilen değişen veya kaybolan hizmetlerin varlığını izler. Hizmetlerin mevcudiyetini duyurmasını ve ayrılmasını sağlayarak, farklı hizmetler arasında sorunsuz iletişimi kolaylaştırır. Diğer hizmetlerin birbirlerini bulmasına yardımcı olarak, Service Discovery, sürekli değişen bir ortamda güvenilir ve verimli iletişimi sağlar.

Mikroservisler bağlamında, Bir Yük Dengeleyici, bir konserdeki yardımcı etkinlik personeline benzer şekilde, katılımcıların çeşitli girişlerden sorunsuz bir şekilde giriş yapmalarını sağlar. Yük Dengeleyicileri, gelen internet trafiğini tek bir sunucuda aşırı yüklenmeyi önlemek için birden çok sunucuya dağıtır. Trafik yöneticisi olarak görev yaparak, Yük Dengeleyicileri iş yükünü sunucular arasında adil bir şekilde dağıtarak, performansı optimize eder ve güvenilirliği artırır. İş yükünü dengelerken, Yük Dengeleyicileri uygulamaların ve web sitelerinin hızını ve güvenilirliğini korur, hatta yoğun aktivite dönemlerinde bile kesintisiz hizmet sunar. Verilen görselden de anlaşıldığı gibi çeşitli algoritmalarla yük dengeleyicisi servislerdeki uygunluğa göre request için uygun yolu bulur.



Soru8-Hibernate,JPA, Spring Data frameworklerini açıklayınız

JPA ORM'ın javadaki standartıdır ama başka ORMLerde vardır. Bu standartta uygun kod yazılırsa bunu destekleyen bir ORM aracıyla çalışılabilir. Öte yandan hibernate ise bir üründür. İlişkiyi şu şekilde açıkladığımızda daha kalıcı oluyor: Hibernate bir araba markası Fiat Volkswagen vs. JPA ise bir araba modeli (standartıdır) sedan gibi hatchback gibi ORM ise taşıt gibidir ve başka dillerde de vardır C# gibi. Detaylı anlatacak olursak:

Spring Data:

Spring Data, Spring Framework'ün bir parçasıdır ve veri erişim katmanı işlemlerini kolaylaştırmak için geliştirilmiştir. Spring Data, çeşitli veritabanları ve veri depolama teknolojileri (SQL tabanlı, NoSQL tabanlı) için genel yapılar ve yardımcı sınıflar sağlar. Bu,

geliştiricilerin tekrar eden veri erişim kodlarını yazmaktan kaçınmalarına olanak tanır. Spring Data, JPA, MongoDB, Redis, Cassandra gibi farklı veri depolama teknolojileri için modüller sağlar ve bu teknolojilerle entegrasyonu kolaylaştırır. Özellikle büyük ve karmaşık uygulamalarda veritabanı işlemlerini yönetmek için oldukça kullanışlıdır.

Hibernate:

Hibernate, Java tabanlı bir ORM (Object-Relational Mapping) framework'üdür. ORM, ilişkisel veritabanı tablolarını nesne yönelimli programlama (OOP) dilindeki nesnelerle eşlemek için kullanılır. Hibernate, veritabanı işlemlerini gerçekleştirmek için SQL sorguları yazmaktan kaçınarak, geliştiricilere veritabanı işlemlerini daha basit ve nesne odaklı bir şekilde gerçekleştirme imkanı sunar. Hibernate, ilişkisel veritabanılarına bağlanmak, verileri sorgulamak, ekleme, güncelleme, silme gibi işlemleri gerçekleştirmek için kullanılır.

JPA (Java Persistence API):

JPA, Java EE (Enterprise Edition) standartlarının bir parçasıdır ve ORM (Object-Relational Mapping) için bir Java API'si sunar. JPA, veritabanı işlemlerini gerçekleştirmek için kullanılan bir arayüz sağlar ve bu arayüzü uygulayan çeşitli ORM framework'leriyle (örneğin, Hibernate, EclipseLink gibi) entegre çalışabilir. JPA, Java sınıflarını veritabanı tablolarıyla eşleştirmek için JavaBean kurallarına dayanan bir dizi anotasyon sağlar. Bu sayede geliştiriciler, veritabanı işlemlerini gerçekleştirmek için daha az kod yazabilir ve daha yüksek bir seviyede soyutlama sağlar.