

HW2

1.

Senkron: Senkron yani eşzamanlı iletişimde, iki aygıt başlangıçta kendilerini birbirleriyle senkronize eder ve ardından senkronize kalmak için sürekli olarak karakterleri gönderir. Veriler gerçekten gönderilmediğinde bile, sabit bir bit akışı, her bir cihazın, diğer herhangi bir zamanda nerede olduğunu bilmesini sağlar. Yani, Gönderilen her karakter ya gerçek veri ya da boş bir karakterdir.

Yazılım ekiplerinde kullanılan senkron iletişim şekillerine aşağıdaki örnekleri verebiliriz;

- Takımların her sabah yaptığı mevcut ilerlemenin, soruların ya da engellerin konuşulduğu, kısa ama etkili stand-up toplantıları
- Projenin geleceğinin tartışıldığı ve bu doğrultuda yapılan işlerin planlandığı, tüm takımın katılımcı olduğu toplantılar
- İlerlemenin değerlendirildiği, doğru ya da yanlış yapılan işlerin veya yöntemlerin tartışıldığı retro toplantıları
- Çalışanların bir iş ya da proje üzerinde birlikte yaptığı çalışmalar (örnek: pair programming)
- Takıma yeni katılanların uyum sürecinin sağlıklı olması için yapılan görüşmeler
- Değerlendirme toplantıları
- 1-1 görüşmeleri
- Önemli şirket gelişmelerinin paylaşıldığı toplantılar

Asenkron : “senkronizasyon yok” anlamına gelir ve bu nedenle boş karakterlerin gönderilmesini ve alınmasını gerektirmez. Bununla birlikte, verilerin her bir baytının başlangıcı ve bitişi, başlangıç ve durdurma bitleri tarafından tanımlanmalıdır. Başlangıç biti, veri baytının başlamak üzere olduğu ve durdurma bittiğinde bittiğini gösterir.

Yazılım ekiplerinde kullanılan asenkron iletişim şekillerine aşağıdaki örnekleri verebiliriz;

- Acil olmayan, kişiyi bölmenin gerekli olmadığı sorular
- Pull requestler
- Code review ve düzenlemeler
- Yeni çalışanlarla tanışma
- Şirket ya da proje hakkında büyük ölçekli haberlerin paylaşılması

2.

RabbitMQ ve Apache Kafka, üreticilerden tüketicilere veri taşıırken farklı yolları kullanır. RabbitMQ, uçtan uca mesaj iletimini önceliklendirirken genel amaçlı bir mesaj aracıdır. Kafka, sürekli büyük verinin gerçek zamanlı değişimini destekleyen dağıtık bir olay akışı platformudur.

RabbitMQ ve Kafka, farklı kullanım durumları için tasarlandıkları için mesajlaşmayı farklı şekillerde ele alırlar.

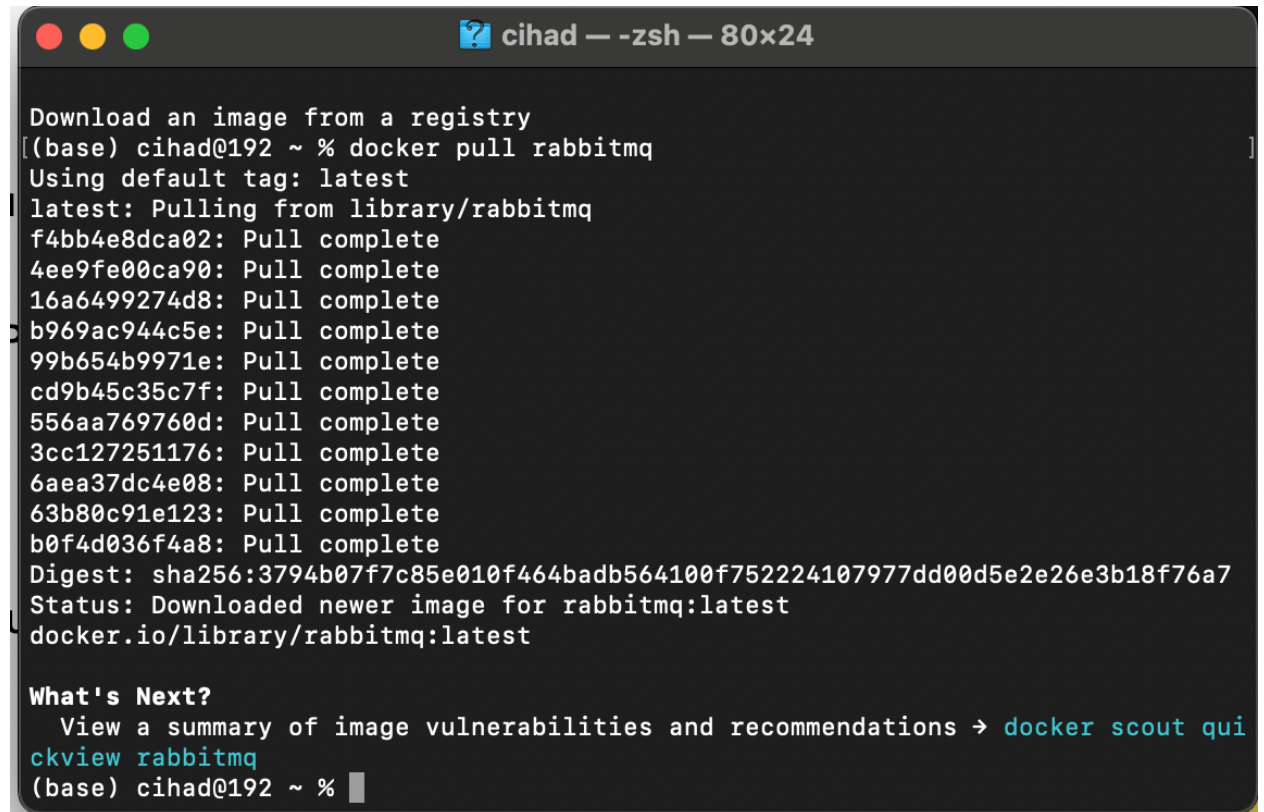
3.

Docker: Docker, uygulamalarınızı hızla derlemenize, test etmenize ve dağıtmanıza imkan tanıyan bir yazılım platformudur. Docker, yazılımları kitaplıklar, sistem araçları, kod ve çalışma zamanı dahil olmak üzere yazılımların çalışması için gerekli her şeyi içeren container adlı standartlaştırılmış birimler halinde paketler. Docker'ı

kullanarak her ortama hızla uygulama dağıtıp uygulamaları ölçeklendirebilir ve kodunuzun çalışacağından emin olabilirsiniz.

Sanal Makine: Genellikle yalnızca VM olarak kısaltan bir sanal makine, dizüstü bilgisayar, akıllı telefon veya sunucu gibi diğer fiziksel bilgisayardan farklı değildir. Dosyalarınızı depolamak için bir CPU, bellek ve diskleri sahiptir ve gerekirse internete bağlanabilir. Bilgisayarınızı oluşturan parçalar (donanım) fiziksel ve somut olduğundan, VM'ler genellikle fiziksel sunucular içinde yalnızca kod olarak bulunan sanal bilgisayarlar veya yazılım tanımlı bilgisayarlar olarak düşünülebilir

4.



```
cihad — -zsh — 80x24

Download an image from a registry
[(base) cihad@192 ~ % docker pull rabbitmq]
Using default tag: latest
latest: Pulling from library/rabbitmq
f4bb4e8dca02: Pull complete
4ee9fe00ca90: Pull complete
16a6499274d8: Pull complete
b969ac944c5e: Pull complete
99b654b9971e: Pull complete
cd9b45c35c7f: Pull complete
556aa769760d: Pull complete
3cc127251176: Pull complete
6aea37dc4e08: Pull complete
63b80c91e123: Pull complete
b0f4d036f4a8: Pull complete
Digest: sha256:3794b07f7c85e010f464badb564100f752224107977dd00d5e2e26e3b18f76a7
Status: Downloaded newer image for rabbitmq:latest
docker.io/library/rabbitmq:latest

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview rabbitmq
(base) cihad@192 ~ %
```

5.

Docker login komutu ile Docker kayıt defterine giriş yapabilirsiniz, burada her türlü **Docker** imajını çekebiliriz

```
docker login
```

Docker pull komutu ile Docker registr'dan bir imaj çekebilirsiniz.

```
docker pull name-of-the-image
```

Belirli bir image arıyorsanız ancak tam adından emin değilseniz **Docker Search** komutunu kullanabilirsiniz.

```
docker search name
```

Birkaç image çektikten sonra, Docker Images komutuyla mevcut olanların bir listesini alabiliriz.

```
docker images
```

Docker RMI komutu ile kullanmadığınız **Docker** imajelerini silebilirsiniz. Buda Docker içerisinde size bir alan açacaktır.

```
docker rmi IMAGE-ID
```

İndirdiğiniz image'leri çalıştırmak için **Docker Run** komutu kullanılır.

```
docker run IMAGE-ID
```

Çalışan containerları listelemek için **Docker PS** komutunu çalıştırabilirsiniz:

```
docker ps
```

Bir konteyneri durdurmak için **Docker Stop** komutunu çalıştırabilirsiniz.

```
docker stop CONTAINER-ID
```

Konteyneri yeniden başlatmak için,

```
docker start CONTAINER-ID
```

Belirli bir konteyneri silmek için önce kabı durdurmanız ve ardından çalıştırmanız gerekir.

```
docker rm CONTAINER_ID
```

6.

Monolitik ve mikroservis mimarileri arasındaki farkları anlamak önemlidir. İşte kısaltılmış bir versiyon:

Monolitik Mimari:

- Avantajlar:

- Basit geliştirme ve hızlı piyasaya sürme.
- Kolay dağıtım ve bakım.
- Test ve hata ayıklama süreçleri basit.

- Dezavantajlar:

- Ölçeklenme zorluğu.
- Yeni teknolojilere uyum güçlüğü.
- Yüksek bağımlılık ve sistem çökme riski.

Mikroservis Mimari:

- Avantajlar:
 - Bağımsız hizmetler ve esnek geliştirme.
 - Farklı teknolojilerin kullanılabilmesi.
 - Ölçeklenebilirlik ve güvenilirlik.
- Dezavantajlar:
 - Zaman ve kaynak tüketimi.
 - Karmaşık dağıtım ve test süreçleri.
 - Yüksek teknik uzmanlık gerektirme.

7.

Load Balancer: API istek trafiğini birden fazla sunucu arasında dağıtan bir bileşendir. Bu, sistem yanıt verimliliğini artırır ve bireysel kaynakların aşırı yüklenmesini önleyerek başarısızlıkları azaltır.

Bir hizmete istek yapıldığında (örneğin, bir web sitesi veya video), yük dengeleyici isteği alır ve hangi sunucunun işleyeceğini ve yanıt vereceğini belirler. Bu, aşağıdaki görüntüde gösterildiği gibi, yük dengeleyicinin çeşitli istemciler ve istekleri sunacak olan arka uç sunucu çifti arasında bulunduğu durumda gerçekleşir.

API Gateway:, API tüketicileri ile arka uç API'leri arasında bir aracı olarak işlev görür. API geçitleri, gelen istekleri yönlendirir, denetler, filtreler, dönüştürür ve güvenlik önlemleri uygular. Bu, API'lerin daha

verimli, güvenli ve ölçeklenebilir bir şekilde kullanılmasını sağlar. API geçitleri ayrıca analiz ve izleme gibi ek işlevsellikler sağlayabilir, böylece API trafiği üzerinde daha iyi bir görünürlük elde edilir.

Service Discovery: Mikro hizmet mimarisinde kullanılan bir kavramdır. Bu, bir ağdaki uygulamalar arasındaki hizmetlerin dinamik olarak bulunması ve erişilmesi işlemidir. Hizmetlerin IP adresleri ve port numaraları genellikle sabit değildir, bu nedenle bir uygulamanın diğer hizmetlere dinamik olarak erişmesini sağlayan bir mekanizmaya ihtiyaç duyulur. Servis keşfi, hizmetlerin adresini ve durumunu belirlemeyi ve diğer hizmetlere erişmek için bu bilgileri kullanmayı içerir. Bu genellikle bir "Service Registry" adı verilen bir bileşen tarafından yönetilir.

8.

Hibernate

- Java tabanlı bir ORM çerçevesidir.
- Nesne yönelimli programlama dilleri ile ilişkisel veritabanı sistemleri arasında veri dönüşümünü sağlar.

JPA (Java Persistence API):

- Java EE platformunun bir parçasıdır.
- İlişkisel veritabanlarıyla etkileşimde bulunmak için kullanılan bir API'dir.

Spring Data:

- Spring Framework'ün bir parçasıdır.
- Veritabanı işlemlerini kolaylaştırmak için tasarlanmıştır.
- Çeşitli veritabanı teknolojileri ile etkileşimde bulunmak için genel bir arayüz sağlar.