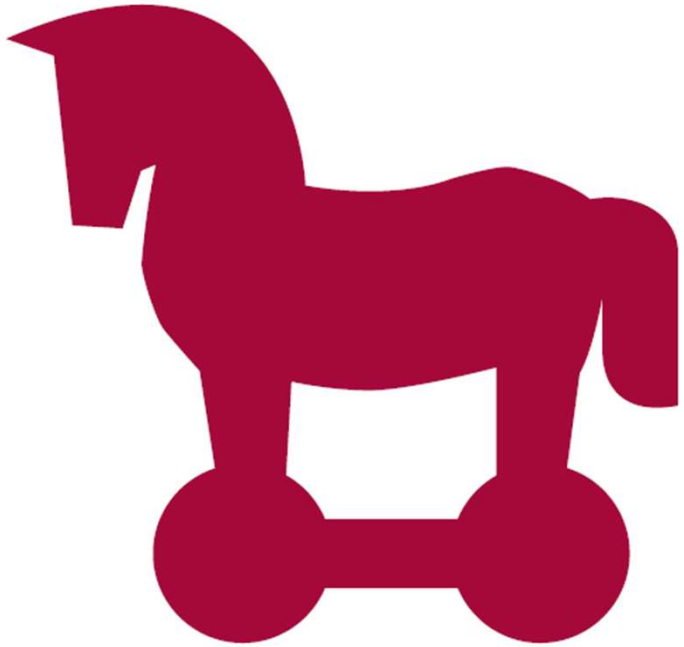
GANTEK ACADEMY

40+ YILLIK TECRÜBE İLE

GANTEK ACADEMY

40+ YILLIK TECRÜBE İLE

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE



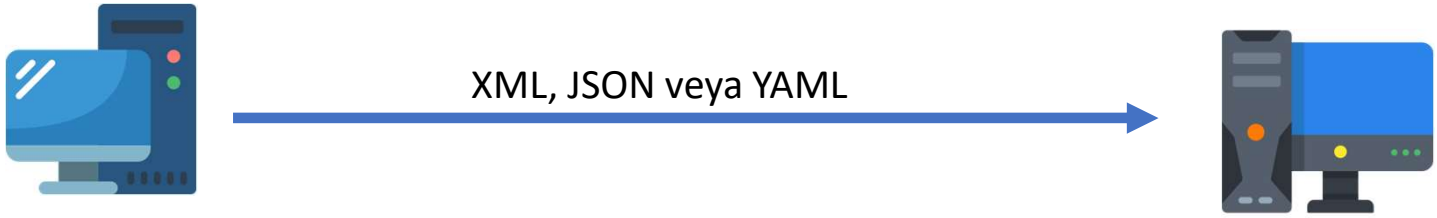
GANTEK

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

Serileştirme: XML, JSON, YAML…

Serileştirme (serialization) nedir?

- Serileştirme, bilgisayar belleğinde yer alan bir veri yapısını, ardışık baytlardan oluşan bir pakete dönüştürmeye verilen addır.
- Bir veri grubu, diğer bir bilgisayara gönderilirken, ardışık baytlardan oluşan bir yapıya dönüştürülmesi gereklidir. Bu yapı, karşı tarafa iletildiğinde, bütünleştirme (deserialization) adı verilen bir işlemle işlenerek, veri yapısının tekrar orijinal haliyle oluşturulması sağlanır.
- JSON, XML, YAML gibi dosya tiplerinin temel amacı da verileri taşınabilir şekilde serileştirmek ve hedef bilgisayarda kaynaktaki gibi aynı şekilde oluşturulmasını sağlamaktır.



https://www.flaticon.com/free-icons/tower-pc

Serileştirme dilleri

- Verilerin ve veri yapılarının oluşturulması için çeşitli diller geliştirilmiştir. Bunların arasında XML, JSON ve YAML en fazla tercih edilen dillerdir.
- XML «eXtensible Markup Language» kelimelerinin kısaltılmış halidir.
  - XML'i, HTML dilini de tasarlamış olan Tim Berners Lee geliştirmiştir. 1998'de «World Wide Web Consortium» (W3C) XML 1.0 standardını yayınlamıştır.
  - XML'in Unicode'u desteklemesi nedeniyle, farklı alfabeleri desteklemektedir.
  - Birçok endüstri standardında XML kullanılmaktadır.

XML

```
<Configurations>

   <Config>
     <Name>Ingress</Name>
     <Value>data/input</Value>
   </Config>

   <Config>
     <Name>Egress</Name>
     <Value>data/output</Value>
   </Config>

</Configurations>
```

https://community.cisco.com/t5/developer-general-knowledge-base/xml-vs-json-vs-yaml/ta-p/4729758

Serileştirme dilleri

- JSON «JavaScript Object Notation» kelimelerinin baş harflerinden oluşmaktadır.
- Douglas Crockford tarafından 2000'li yılların başında geliştirildi ve 2013 yılında standartlaştırılmaya başlandı ve standart dokümanı 2017'de yayınlandı.
- XML'in alternatifidir ve XML'e göre daha fazla ilgi çekmiştir. Çünkü XML'e göre daha hafif şekilde oluşturulabiliyordu.
- Ayrıca, JSON dokümanları JavaScript betikleriyle karıştırılmıyordu.
- Unicode karakterlerini desteklemektedir.
- Çok sayıda uygulama alanı bulunmaktadır.

```
{
    "configurations":[
        {
            "name": "Ingress",
            "value": "data/input"
        },
        {
            "name": "Egress",
            "value": "data/output"
        }
    ]
}
```

Serileştirme Dilleri

- YAML, öz yinelemeli bir akronim olan «**Y**AML **A**in't **M**arkup **L**anguage» kelimelerinin ilk harflerinden türetilmiştir.
- Clark Evans tarafından 2001'de önerilmiştir. O dönemde YAML «**Y**et **A**nother **M**arkup **L**anguage» kelimelerinden türetildiği düşünülmüştür.
- 2001'de ilk versiyonu, Ekim 2021'de de 1.2.2 sürümü çıkmıştır.
- XML ve JSON'a göre daha hafif bir dildir.

```
---
configurations:
 - name: Ingress
   value: data/input
 - name: Egress
   value: data/output
```

XML, JSON ve YAML

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- XML bir işaretleme (markup) dili iken JSON ve YAML veri formatlarıdır.
- XML bir ağaç yapısıyla etiketler kullanarak tanım yaparken, JSON anahtar/değer ikililerini kullanarak tanımlama yapar. YAML ise, veri tanımını listeler ve sıralı değerler şeklinde anahtar/değer ikilileri şeklinde ifade eder.
- JSON'da «comment» özelliği yoktur. XML ve YAML dosyalarında yorumlar bulunabilir.
- XML komplex veri yapılarını tanımlayabilir. JSON'da stringler, sayılar, diziler, doğru/yanlış ve nesneler bulunabilir. YAML, zaman damgası, sıralı diziler, iç içe ve öz yinelemeli değerler gibi karmaşık veri yapılarını tanımlayabilir.
- XML'de verileri okumak zordur. JSON'da biraz daha kolay şekilde veri yapıları okunabilirken, YAML'da verileri okumak daha kolaydır.
- XML veri alışverişi için kullanılır. JSON serileştirme formatı olarak ve API'lerde faydalıdır. YAML ise yapılandırma dosyaları için kullanılır.
- XML'de gereksiz bir çok ek vardır. JSON dosyaları XML'e göre daha küçüktür ama JSON'un süperseti olan YAML'da dosyalar daha da küçüktür.
- YAML dosyasını yazmak dikkat gerektirir ve çok hatayla karşılaşılabilir.

https://community.cisco.com/t5/developer-general-knowledge-base/xml-vs-json-vs-yaml/ta-p/4729758

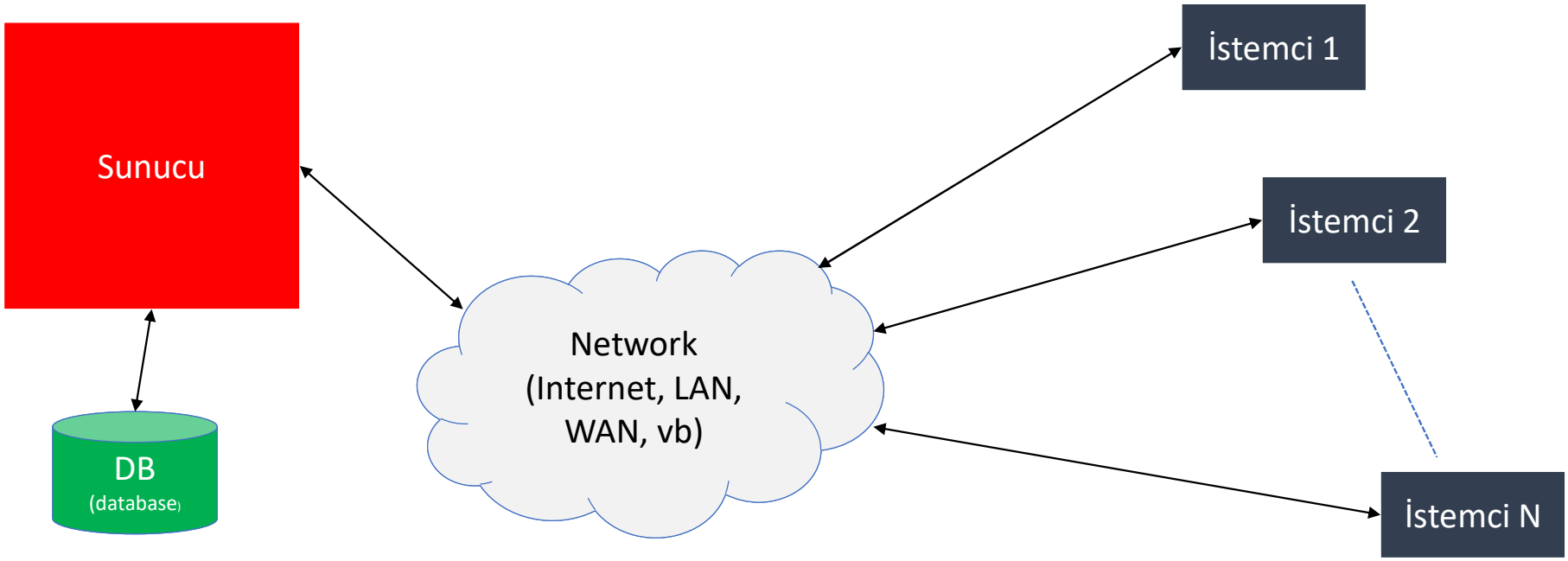
GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

İstemci Sunucu Mimarisi
Mikroservis Mimarileri

İstemci - Sunucu Mimarisi

- İstemci-sunucu mimarisi, bilgisayarlardaki veri kaynaklarının, ağ üzerindeki iş istasyonları ve kişisel bilgisayarlar tarafından kullanılabilmesini sağlayan bir mimaridir.
- 1960-70'lerde kuramsal olarak oluşturulan istemci-sunucu mimarisi, 1980'lerde yaygınlaşmış, hızlı ağ sistemlerinin gelişimiyle birlikte uygulama mimarilerinde baskın duruma gelmiştir
- Bu mimaride, sunucular 7x24 ağa bağlı durumdadır ve üzerlerinde birden fazla sunucu programı bulunmakta ve bu yazılımlar istemcilerden gelen sorgulara yanıt vermektedir.
- İlk istemci-sunucu uygulamaları arasında, web sunucuları, e-posta sunucuları bulunur. Daha sonraları ilişkisel veritabanlarının yaygınlaşmasıyla birlikte, sunucu-istemci mimarisi sayesinde veritabanındaki verilerin istemciler tarafından sorgulanmasına dayanan uygulamalar ortaya çıkmıştır.

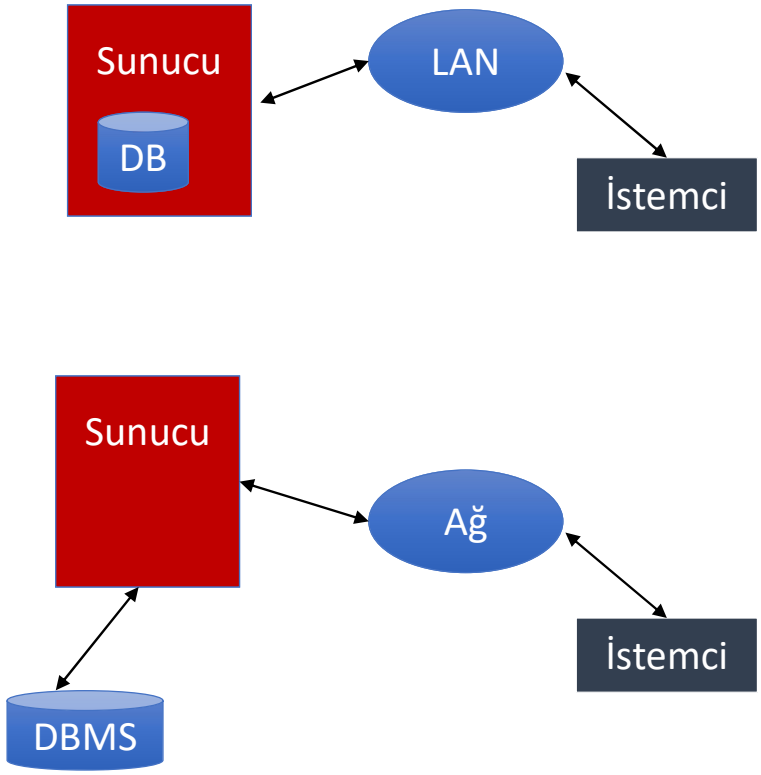
İstemci-Sunucu Mimarisi



İstemci, sunucudaki kaynaklara erişmek için sunucuya bağlanarak talepte bulunur ve sunucu istediği verileri ağ üzerinden gönderir.

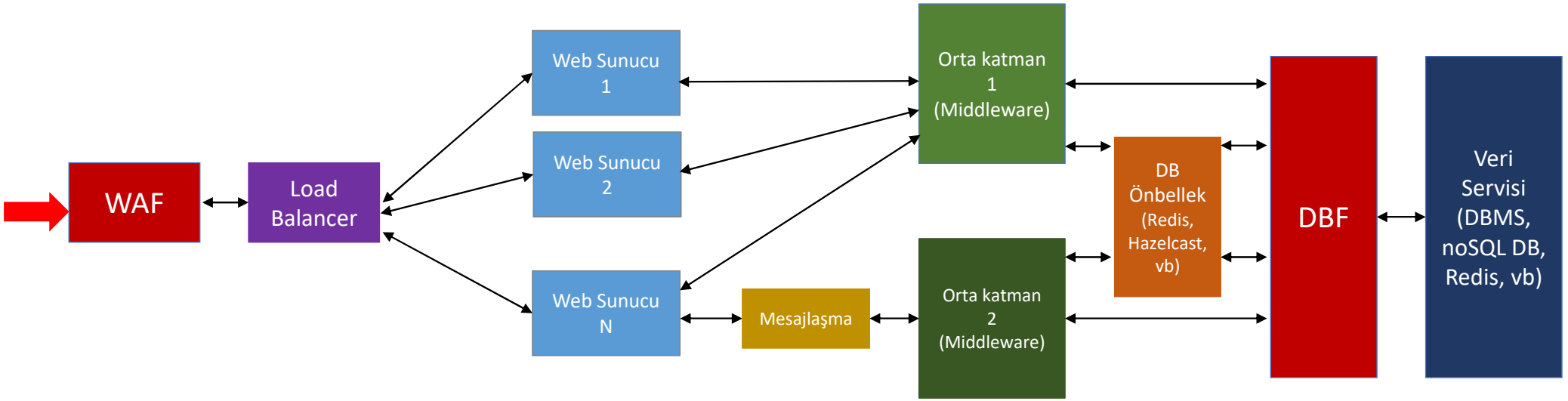
İstemci-Sunucu Mimarisi: bağlantı tipleri

- Geleneksel istemci-sunucu mimarisi
  - İstemcilerle sunucular aynı yerel ağ içindedir.
  - Sunucu üzerinde veritabanı (DB) bulunur
- İki aşamalı (two-tier) mimari
  - İstemciler sunucuya ağ (Internet, LAN, WAN, vb) üzerinden ulaşırlar
  - Sunucu, veritabanına arka planda bağlıdır.
  - İstemci sunucuya ağ üzerinden bağlıdır ve sunucu üzerindeki uygulamalarla veritabanı sunucusundaki verilere erişir.



İstemci-Sunucu Mimarisi: bağlantı tipleri

- N aşamalı (N-tier) mimari
  - Sunucuların arkasında farklı sayıda katman olabilir.

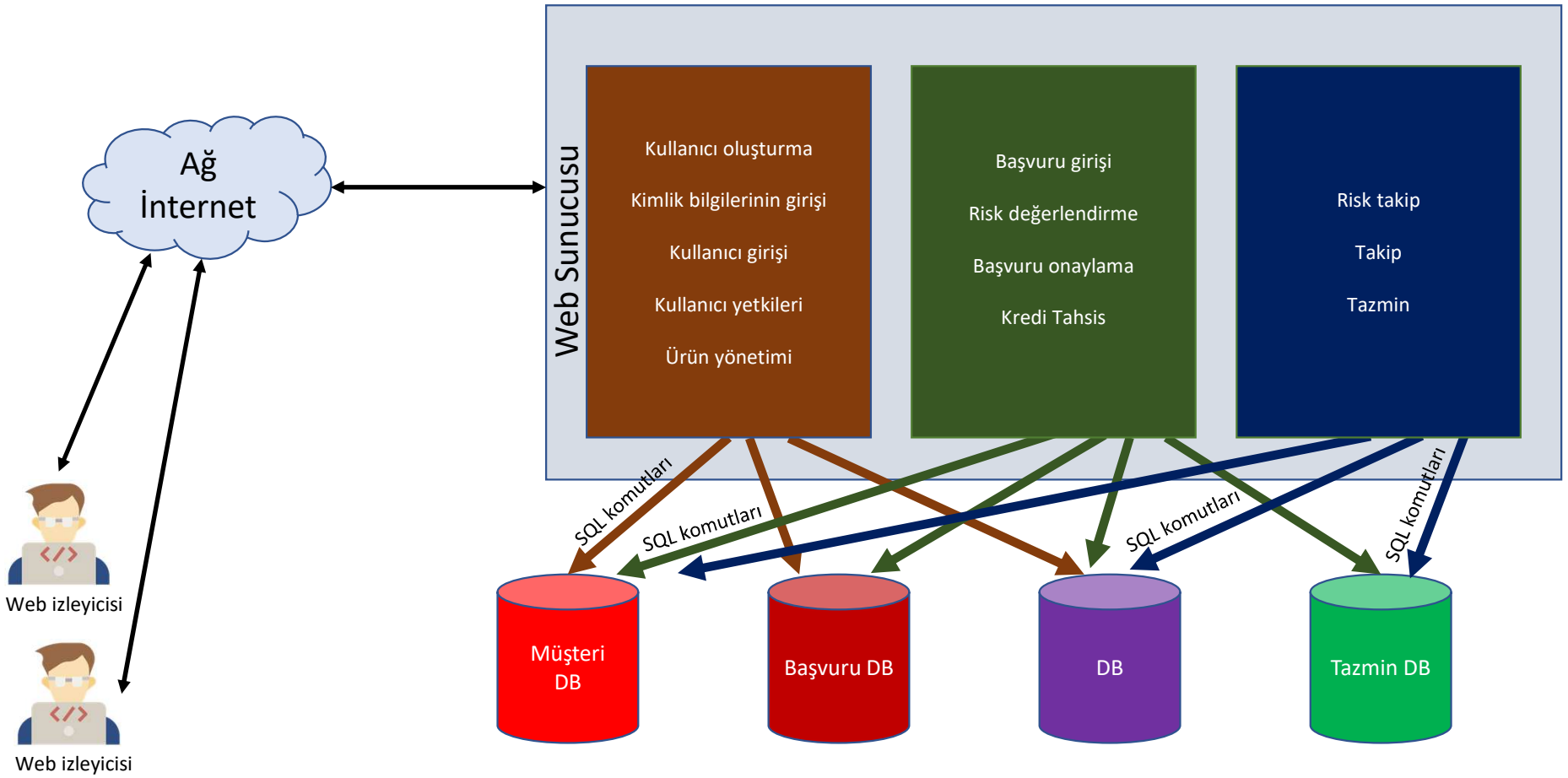


https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier

İstemci – Sunucu Mimarisi

- Geleneksel istemci-sunucu mimarisinde, sunucularda çalışan yazılımlar belirli bir istemciye göre yazılırlar.
  - Bir e-posta sunucusuna bağlanarak, hesabınızdaki e-postalara erişmek için veya e-posta göndermek için POP3 veya IMAP temelli bir e-posta okuyucusu kullanmanız gerekir.
  - Bir PostgreSQL veritabanı sunucusuna bağlanmak için, SQL komutlarını gönderebileceğiniz ve sonuçları alabileceğiniz bir ara yüz programı kullanmalısınız. Bu ara yüz üzerinden gönderdiğiniz SQL komutları sunucuda işlenerek sonuçlar aynı ara yüz üzerinden size iletilir.
  - Bir web sunucusunda bulunan web sayfalarına veya web nesnelerine ulaşmak için, HTTP/HTTPS protokolüyle sunucuya bağlanmak gerekmektedir. Çoğunlukla bu iş için bir web istemcisi kullanırsınız.
- Diğer bir deyişle, protokole bağımlılık, istemci-sunucu mimarisinin en büyük sorunlarından biridir.
- **SORU**: Protokole bağlı olmaksızın veri alabileceğiniz sorabileceğiniz bir yapı kurulabilir mi?

Geleneksel İstemci - Sunucu Mimarisi



İstemci Sunucu mimarisinde sunucu yazılımı

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- 20 sene önce, sunucular için monolitik yapıda Java programları hazırlanırdı.
  - Struts veya JSF (Java Server Faces) web sayfalarının tasarımı ve gösterilmesi için kullanılırken,
  - Hibernate (ORM) veritabanına bağlanmak için faydalanılırdı.
  - Bütün program tek bir WAR dosyası haline getirmek için büyük çaba harcanırdı.
  - Yeni bir özellik eklenecekse veya bir «bug» çözülecekse, 2-3 satır bile olsa yeni kodlama yapıldıktan sonra yine büyük bir WAR dosyası oluşturulur ve sunucuya yüklenirdi.
  - Sunucuya yüklenen WAR dosyası çalıştırılmadan önce
    - tüm sistem kapatılır (tüm oturumlar kapatılır, veriler kaybolur….)
    - Güncellenen veritabanı tablo yapılarına uygun olarak eski tablolar dönüştürülür
    - Sisteme yüklenen WAR dosyasıyla birlikte sunucu ayağa kalkar.
- **SORU**: Ne var bunda canım? Biz hep böyle yapıyoruz zaten.
- **YANIT**: Buna pire için yorgan yakmak denir. Programın sadece ilgili bölümü güncellense ve sadece o bölümünü devreden çıkarıp yenisini devreye alsak iyi olmaz mı?

SOA: Servise Dayalı Mimari nedir?

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- *Geleneksel İstemci-Sunucu* mimarisinde en büyük sorun, yazılımın belirli protokollere göre (sorgularına dayanarak) yazılmış olmasıdır. Bu da çeşitli sıkıntılar ortaya çıkarır:
  - Programın güncellenmesinde sorunlar çıkarır. Belirli bir modüldeki, ufak bir güncelleme bile tüm sistemin durdurulmasına sebep olur. Yeni yazılım kurulur ve herkesin yeni yazılımı kullanması sağlanır.
  - Farklı bir veritabanı sunucusuna geçilmesi, tüm ara yüzlerin değiştirilmesine sebep olacağından sorun çıkarması muhtemeldir.
- Bütün bu sorunlar SOA mimarisine geçilmesiyle çözülebilir.
- Bu amaçla, geleneksel istemci-sunucu yazılımı bileşenlerine ayrılır ve her bileşen, web üzerinden çağrılabilen bir servis haline getirilerek farklı bir sunucuda bağımsız olarak çalışabilir hale getirilir.

https://www.ibm.com/cloud/learn/soa
https://www.redhat.com/en/topics/cloud-native-apps/what-is-service-oriented-architecture

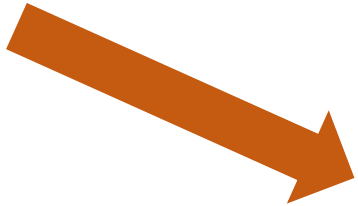
SOA: Servis nedir?

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Servislerin özellikleri şunlardır:
  - Tanımlı girdilerle istenen çıktıyı bir web ara yüzü üzerinden üretir.
  - Aynı web servis, eş zamanlı (concurrent) şekilde çalıştırılabilir.
  - Kendi kendine yeten bir yapıdadır ve genellikle durumsuz (stateless) yapıda olması istenir. Durumsuz olması, eş zamanlı çalışabilmesini de sağlar.
  - Kullananlar için servis kapalı bir kutudur. Arka planda nasıl çalıştığı ve iç ayrıntılar bu servisi çağıran yazılım tarafından bilinmez. Servis argümanlarla çağrılır ve servis de beklenen çıktıları döndürür.
  - Bir servis, arka planda, diğer servisleri çağırabilir ve döndüreceği verileri bunlardan aldığı verilere dayanarak oluşturabilir.

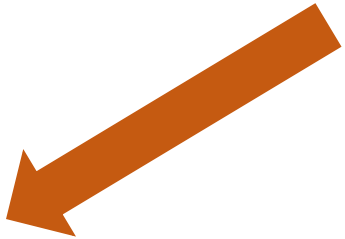
SOA: Servis nedir?

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```



SOA Servis

Web temellidir.
Kara kutu olarak görülebilir.
İstemciden bağımsızdır.



```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

SOAP nedir?

- SOAP, web temelli SOA servisleri için bir mesajlaşma protokolüdür.
- 1998'de Dave Winer, Don Box, Bob Atkinson ve Mohsen Al-Ghosein tarafından tanıtılmıştır.
- 2000'li yıllarda SOAP farklı sistemler arasında veri aktarımı konusunda popüler olmuştur. Farklı platformlarda çalışan sistemler ve farklı programlama dillerinde yazılmış programlar arasında veri alışverişinin gerçekleştirilmesini sağlama konusunda yaygın bir kullanım alanı bulmuştur.
- SOAP uygulamaları özellikle kurumsal uygulamalarda kullanılmış ve güvenilir ve güvenli bir iletişim aracı olarak ele alınmıştır.
- Ancak son yıllarda, SOAP'a göre daha esnek bir yöntem olan RESTful API'lerin kullanımıyla birlikte, SOAP'ın popülerliği azalmıştır.

https://dzone.com/articles/rest-vs-soap-a-comprehensive-guide-to-their-differ

SOAP : SOA Protokolü (request)

- Web sunucusunda bulunan web servislerine nasıl bağlanılacağı, nasıl çağrılacağı ve verilerin nasıl alınacağına dair tanımları yapar.

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

https://www.w3schools.com/xml/xml_soap.asp

SOAP : SOA Protokolü (response)

- SOAP, XML metin temelli veri tanımlama dili kullanır.
- SOAP talebi XML'de yapıldıktan sonra, cevap da XML türünde döner.

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

https://www.w3schools.com/xml/xml_soap.asp

REST nedir?

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Web URL üzerinden veri sorgulanmasını sağlayan bir SOA yöntemidir. REST mimarisinin amacı, verinin ve veri yapılarının web üzerinden transferini standartlaştırmaktır.
- REST API ve RESTful API kavramları, aslında, birbirinden farklı değildir. REST bir mimariyi tanımlarken RESTful API, REST mimari üzerinde çalışan web servislerini tanımlar.
- Twitter, Amazon, GCP, Azure, vb şirketler tarafından sistemlerinde yaygın şekilde kullanılmaktadır.
- İstemci, http/https üzerinden URL içinde belirttiği parametrelerle sorgulayarak, veriyi web sunucusundan JSON, XML, YAML, vb formatında metin biçiminde alabilir.
- Örnek olarak OpenWeatherMap REST API servisi verilebilir. İstenen bir koordinat için REST üzerinden hava durumu sorgulanabilir.
  - https://api.openweathermap.org/data/2.5/weather?lat=35&lon=139&appid={API key}

REST nedir?

TALEP

REST API URL + API yöntemi + parametreler

`https://api.openweathermap.org/data/2.5/weather?lat=39.76483&lon=30.52256`

protocol URL Yöntem Parametreler

CEVAP

REST XML JSON YAML Metin, vd

WEB SUNUCUSU

OpenWeatherMap REST API servisi

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Eskişehir Odunpazarı mevkiindeki Odunpazarı Modern Müze'nin Enlem boylamı **39.76483**, **30.52256**'dir. Bu koordinat için güncel hava durumu şu şekilde sorgulanabilir:
  - **curl "https://api.openweathermap.org/data/2.5/weather?lat=39.76483&lon=30.52256&appid=b9bdaf75a7b1e96362a172ec84cb9303"**
- Sonuç bir JSON dosyası olarak geri döndürülmektedir.
- Sıcaklık, hissedilen sıcaklık, rüzgar, hava durumu, yağış miktarı, vb hava verileri ve belirtilen koordinat için gün batımı, gün doğumu, güneşin tepe noktasında bulunduğu saat bilgileri bu JSON içinde bulunmaktadır.
- İstemci, REST'ten gelen JSON formatındaki dokümandan istenen bilgiyi JSON dokümanında arayarak bulabilir.

https://openweathermap.org/

```
{
  "coord": {
    "lon": 30.5226,     "lat": 39.7648
  },
  "weather": [
    {
      "id": 802, "main": "Clouds", "description": "scattered clouds",
      "icon": "03d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 297.06, "feels_like": 296.63, "temp_min": 297.06,
    "temp_max": 297.06, "pressure": 1019, "humidity": 43
  },
  "visibility": 10000,
  "wind": {
    "speed": 5.14, "deg": 320
  },
  "clouds": {
    "all": 40
  },
  "dt": 1653480485,
  "sys": {
    "type": 1, "id": 6988, "country": "TR",
    "sunrise": 1653446153, "sunset": 1653498846
  },
  "timezone": 10800, "id": 315202, "name": "Eskişehir",
  "cod": 200
}
```


```
$ curl https://api.openweathermap.org/data/2.5/weather?lat=39.76483&lon=30.52256&appid=b9bdaf75a7b1e96362a172ec83cb9303
```

REST API özellikleri

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- **İstemci-Sunucu mimarisi**: REST API kullanan sistemlerde, istemciler sunuculara bağlanarak HTTP talebi içinde bazı parametreler gönderirler ve cevaplarını XML/YAML/JSON gibi formatlarda alırlar.
- **Durumsuzluk**: REST API'ler durumsuzdur. Diğer bir deyişle sunucular istemcilerle ilgili herhangi bir bilgi tutmazlar ve açılan oturumla ilgili bilgi bulunmaz. REST API talebinin sunucuda karşılanması için bütün bilgiler talep içinde yer alır.
- **Ön bellekte tutulabilmesi**: REST API'ler ön bellekte tutulabilir ve tutulabilmelidir. Talebin ve daha önce verilen cevabın önbellekte tutulabilmesi performansı artıran bir özelliktedir.
- **Katmanlı sistem mimarisi**: REST API'ler katmanlı sistem yapısı kullanabilirler. Diğer bir deyişle, istemci kendi içinde, sunucu hakkında herhangi bir ayrıntı tutmak zorunda değildir. REST API hizmeti veren sistem, istemciden gelen bilgileri bir şekilde toplar ve istemciye sunar.
- **Basit ve standart arayüz**: REST API'ler HTTP talebi sayesinde standart bir arayüz sunarlar. Bir istemcinin diğer bir REST API servisine yönlendirilmesi basitleşir. Kullanılan dört standart yöntemle (GET, POST, PUT ve DELETE) bütün işler gerçekleştirilebilir.
- **Kaynak temelli oluşu**: REST API'ler kaynak temelli mimariye sahiptir. Diğer bir deyişle, bir kaynak (blog sayfası, bir kullanıcı, bir DB tablosu üzerinde yapılan işlemler eşsiz bir URL'e sahiptir.
- Bant genişliğinin optimum kullanımı: REST API'ler bant genişliğini etkin kullanırlar.

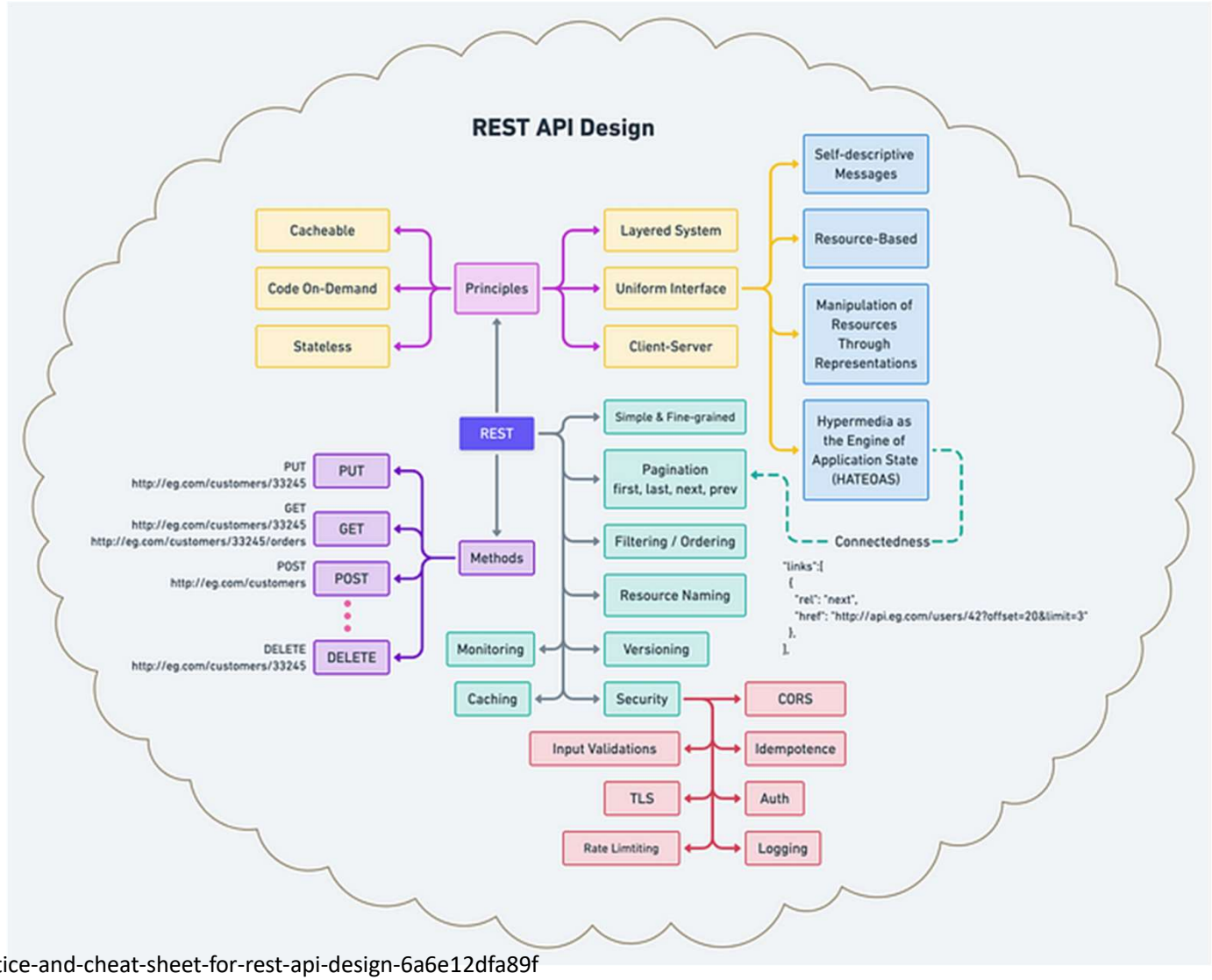
https://dzone.com/articles/rest-vs-soap-a-comprehensive-guide-to-their-differ

RESTful API

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- REST mimarisine göre çalışan web servislerine RESTful web API denir. Farklı programlama dillerinde yazılmış programlar, RESTful API üzerinden birbiriyle haberleşebilirler.
- RESTful API, verileri, HTTP protokolü üzerinden GET, PUT, POST ve DELETE gibi yöntemlerle okuyabilir, düzeltebilir, oluşturabilir ve silebilir (CRUD: Create, Read, Update, Delete).
- RESTful API yöntemleri, HTTP sunucu kodları üzerinden istemciyle haberleşir.
  - 200'lü kodlar talebin başarılı ve sonucun doğru olduğunu gösterir.
  - 400'lü kodlar istemci tarafında hata olduğunu belirtir.
  - 500'lü kodlar sunucu tarafından hata oluştuğunu bildirir.
- URI (Universal Resource Identifier) REST mimarisindeki her bir kaynağı belirtir.
  - URN: kaynağı belirlemek için kullanılan eşsiz isim
  - URL: Tipik web adresi

REST API tasarımı



https://blog.devgenius.io/best-practice-and-cheat-sheet-for-rest-api-design-6a6e12dfa89f

SOAP ve REST

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

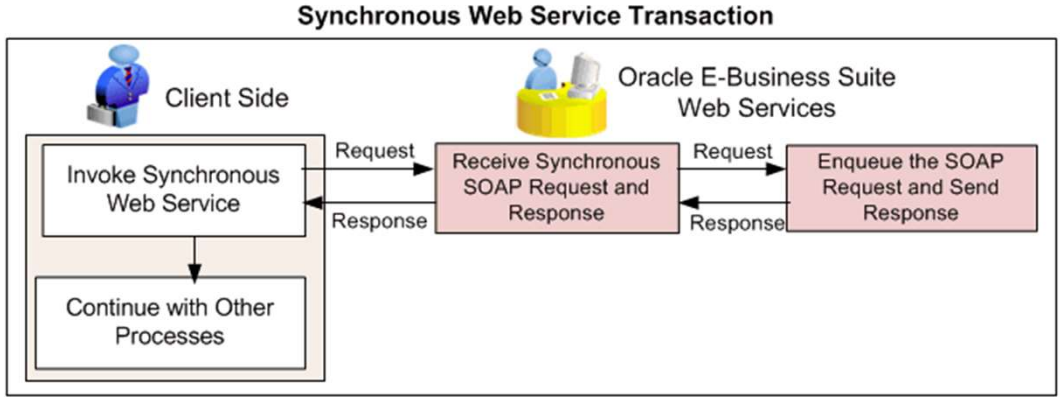
- SOAP ve REST yöntemleri SOA uygulamalarıdır.
- SOAP, web sunucuda bir web fonksiyonunun çağrılması ile ilgiliyken, REST, URL içine girilen parametrelerle sorgu yapılmasına dayanmaktadır.
- REST ve SOAP arasında önemli farklar bulunur.

REST avantajları	SOAP avantajları
Daha az bant genişliği	Daha çok bant genişliği gereksinimi (XML nedeniyle)
Ölçeklenebilir	Dağıtık uygulamalarda kullanılmaya yöneliktir.
HTTP GET, HTTP POST, HTTP PUT, HTTP DEL gibi çağrı türleriyle kullanılabilir.	SMTP, HTTP POST, MQ gibi yöntemler desteklenmektedir.
Senkron çalışır	Senkron ve asenkron çalışabilir.
HTTPS ne sağlarsa o kadar	WS güvenlikle daha güvenli
Servise gönderilen parametreler, HTTP çevre parametresi olarak gönderilir ve geri dönen veriler, herhangi bir formatta olabilir.	Belirli bir formatta parametre gönderilir ve cevap dönülür. Formatta hata yapıldığında, sorun çıkar.

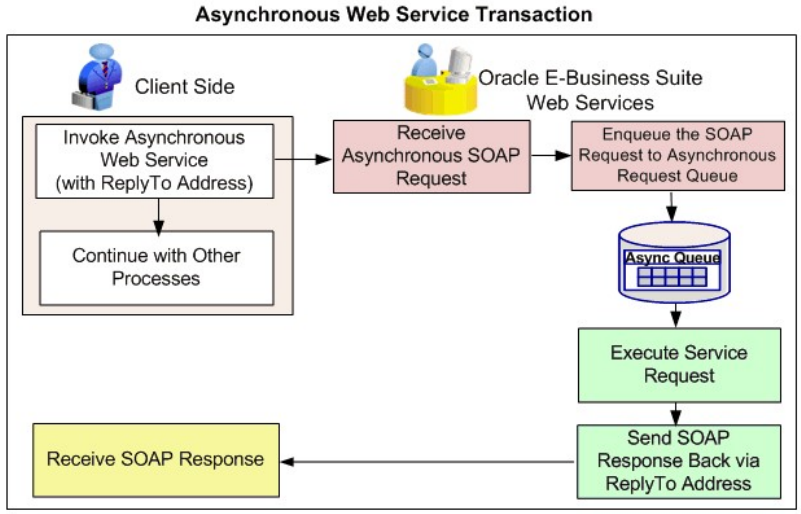
https://blog.devgenius.io/best-practice-and-cheat-sheet-for-rest-api-design-6a6e12dfa89f

SOAP

- Senkron web servis



- Asenkron web servis



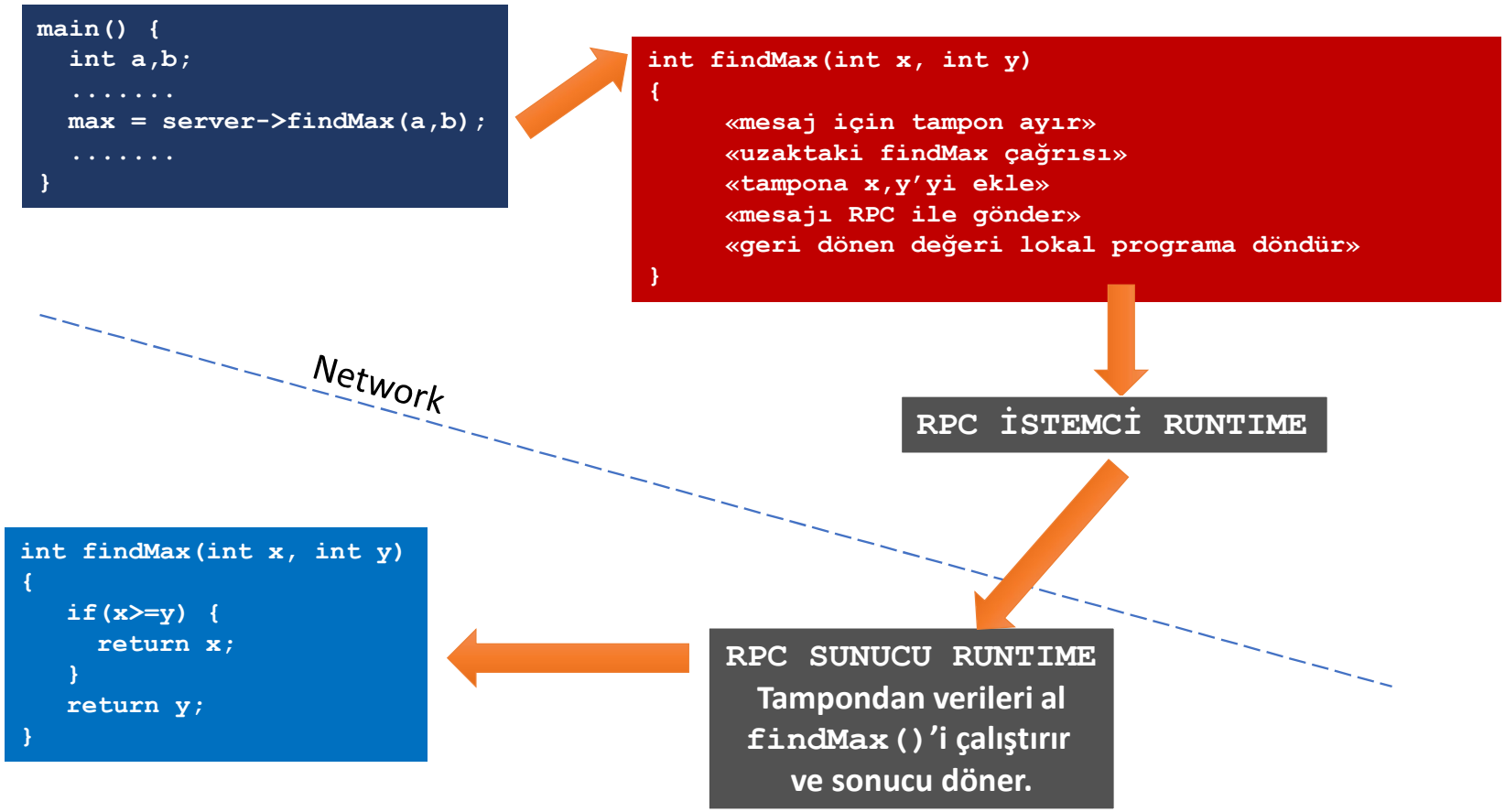
https://docs.oracle.com/cd/E26401_01/doc.122/e20925/T511175T626069.htm

RPC nedir?

- Dağıtık sistemlerde, RPC, bir fonksiyonun (yordamın) diğer bir bilgisayarda çalıştırılmasına denir.
- RPC bir çeşit IPC (Inter-process Communication) yapısıdır.
- Bir RPC bir istemci tarafından başlatıldığında, yordamın çalıştırılacağı uzaktaki sunucuya ağ üzerinden bir talepte bulunur argümanları gönderir.
- TCP ve UDP protokolleri, argümanların ve çıktıların iletilmesinde kullanılır. Her bir program, bir UDP/TCP portuna bağlanır. Argümanlar UDP ile gönderilir.
- Sunucu sistemde, verilen argümanlarla birlikte çalıştırılan yordamdan dönen çıktılar istemciye UDP üzerinden geri iletilir. Ancak yordamın çıktıları bir UDP paketine sığmazsa TCP üzerinden iletilir.
- RPC uygulamaları ilk olarak SunOS işletim sistemlerinde geliştirildi.
- RPC'ye dayanan ilk ve önemli uygulamalardan biri NIS (Name Information Server) ve NFS (Network File System) uygulamalarıdır.

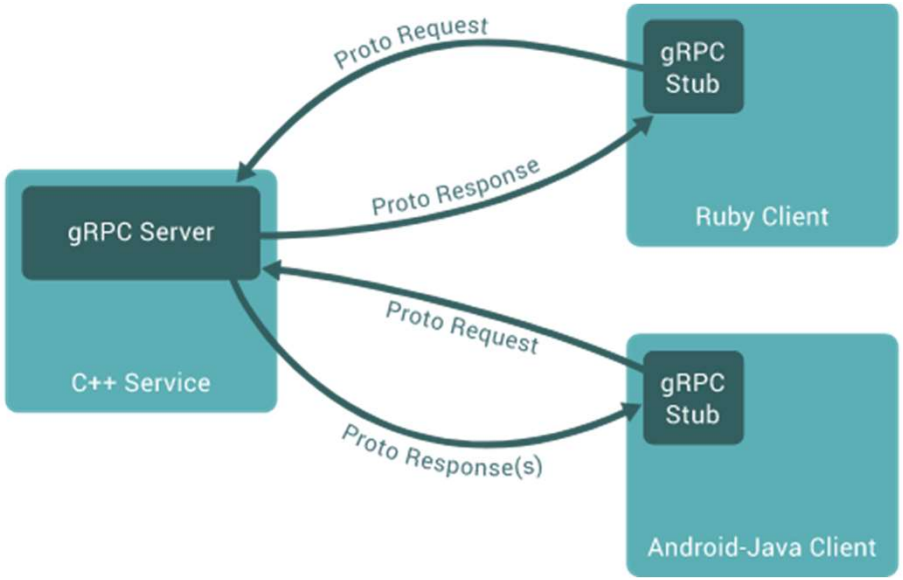
RPC

- Lokalde bir program içinde çalıştırılan bir fonksiyon, uzaktaki yordama bağlanarak oradaki kodu çalıştırır ve sonuçları sanki lokal fonksiyon döndürmüş gibi döndürür.



gRPC nedir?

- gRPC modeli API tasarımı için kullanılan ve eski RPC yöntemlerine göre daha çağdaş ilkelere sahip bir RPC yöntemidir.
- Diğer RPC'ler gibi, sanki lokaldeki bir fonksiyonu çalıştırır gibi sunucuda kod çalıştırmaya yönelik bir yaklaşımdır.
- HTTP/2 protokolünü model olarak kullanır ama API tasarımcısından bu ayrıntıları saklar.
- gRPC farklı ortamlardaki istemcilerin sunucularla olan iletişimini sağlamak için kullanılabilir. Bu istemciler farklı programlama dilleriyle yazılmış olabilir.
- gRPC, CNCF (Cloud Native Computing Foundation) projesidir ve açık kaynak kodlu bir proje olarak Google tarafından CCNF'in yönetimine verilmiştir. Google servislerinin API'sinde gRPC kullanılır.



https://grpc.io/
https://grpc.io/docs/what-is-grpc/introduction/

gRPC nedir?

- Diğer RPC sistemlerindeki gibi, gRPC veri yapılarını serileştirmek için «**Protocol Buffers**» özelliğini kullanır.
- Protokol tamponları, **.proto** ile biten metin dosyalarıdır ve uzaktaki RPC'ye gönderilecek verilerin ve mesajların tanımlanmasını sağlar.
- İstediğiniz programlama dilinde protokol tamponlarını derlemek için, «**protoc**» derleyicisi kullanılır.

person.proto

```
message Person {
  string name = 1;
  int32 id = 2;
  bool has_ponycopter = 3;
}
```

```
// The greeter service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

https://grpc.io/docs/what-is-grpc/introduction/
https://grpc.io/docs/what-is-grpc/core-concepts/
https://protobuf.dev/overview/

gRPC nedir?

- «**protoc**» derleyicisi, «***.proto**» dosyasını kullanarak hem sunucu hem de istemci için kaynak kodu oluşturur.
- Bu kaynak kodları, programınıza eklenir ve sunucu ile istemci arasındaki iletişim bu kaynak kodları kullanılarak yapılır.
- Örneğin, C++ dilinde yapılan gRPC uygulamasında, «**protoc**» derleyicisi C++ sınıfları oluşturulur.

```
class GreeterServiceImpl final : public Greeter::Service {
  Status SayHello(ServerContext* context,
                  const HelloRequest* request,
                    HelloReply* reply) override {
     // ...
  }

  Status SayHelloAgain(ServerContext* context,
                       const HelloRequest* request,
                        HelloReply* reply) override {
    std::string prefix("Hello again ");
    reply->set_message(prefix + request->name());
    return Status::OK;
  }
};
```

https://grpc.io/docs/languages/cpp/quickstart/

gRPC veya REST?

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Yazılım geliştiriciler, gRPC ve REST kullanmak arasında çelişkiye düşebilirler.
- Her ikisi de aynı işe yarar ve bunlardan hangisinin kullanılması gerektiği konusu da tercih yapmayı gerektirir.
- gRPC:
  - gRPC, **`.proto`** dosyası gerektirir ve bunu oluşturabilmek için bir dizi kuralın bilinmesi gereklidir. gRPC derleyicisi, farklı programlama dilleri için kod oluşturur. Kod oluşturmak gRPC'de kolaydır.
  - HTTP/2 protokolüne ihtiyaç duyar ve bu şekilde aynı TCP bağlantısı üzerinden birden fazla talepte bulunulabilir. Sunucu istemcilere, kurulan ve kopmamış bağlantı üzerinden bildirim gönderebilir.
  - gRPC, Protokol Tamponu üzerinden aktarılacak verileri serileştir. Bu şekilde veriler sıkıştırılabilir ve daha hızlı şekilde karşı tarafa iletebilir.
- REST:
  - implementasyon açısından daha serbesttir ve kod oluşturma konusunda regülasyonları yoktur. Yazılım geliştiriciler, **`Swagger`** benzeri üçüncü parti araçlara ihtiyaç duyabilirler.
  - REST modeli HTTP:1.1'i kullanır ve el sıkışmanın yarattığı gecikme açısından gRPC'ye göre daha yavaştır.
  - REST API genellikle JSON ve XML formatlarıyla veri transfer eder. Bunların da transfer edilen verilere olan ek yükü söz konusudur.

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

Ne zaman SOAP kullanılmalı?

- **Kurumsal uygulamalar**: SOAP API genellikle güvenlik ve dayanıklılık gerektiren kurumsal uygulamalarda tercih edilen yöntemdir. SOAP API'sinin içinde şifreleme ve dijital imza gibi güvenlik tedbirleri yer alır ve karmaşık uygulamalarda güvenlik açıklarını engeller.
- **Yüksek güvenlik seviyesi gerektiren uygulamalar**: SOAP API'nin standartları içinde yer alan güvenlik özellikleri, yüksek güvenlik seviyesi gerektiren uygulamalar için faydalıdır. Örneğin, SOAP API'ler hassas verinin iletiminde şifreleme ve dijital imza gibi özellikleri kullanarak verinin doğruluğunu ispatlayabilir.
- **Farklı sistemler arasında entegrasyon**: SOAP API'ler farklı sistemler arasında standart bir biçim sunarlar ve bu da farklı sistemler arasında veri alışverişini kolaylaştırır. Bu şekilde verilerin, ulaştıkları sistemlere, gönderildikleri sistemdeki halinde gitmeleri sağlanır.
- **Büyük miktarda verinin transfer edilmesi gerekli olan uygulamalar**: SOAP API'ler RESTful servislere göre daha çok verinin iletilmesini sağlarlar.

Ne zaman gRPC kullanılmalı?

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Hafif mikroservis bağlantıları ve düşük gecikme ile yüksek iletim hızı gerekiyorsa gRPC kullanılabilir. gRPC'nin RESTful API'ye göre daha hızlı olduğu kabul edilir.
- Gerçek zamanlı akımlarda daha faydalıdır. Zira, HTTP/2 kullanması sebebiyle tek bağlantı üzerinden birden fazla dosya talebi yapılabilir ve sunucu-istemci arasında bildirimler gönderilebilir.
- İstemci cihazlarda (örneğin gömülü IoT sistemlerde) düşük güç tüketimi gerekiyorsa gRPC'nin daha başarılı olduğu kabul edilir..
- Düşük bant genişliği varsa gRPC kullanılmalıdır çünkü gönderilecek parametreler ve yanıtların boyutları, sıkıştırma yöntemleriyle daha da azaltılabilir (İstemci ve sunucu taraflarındaki kaynak kodlara ek özellikler eklenebilir)
- Güvenlik gerektiren uygulamalarda (finansal, banka, vb) gRPC ek güvenlik tedbirler sağlar.

https://hevodata.com/learn/grpc-vs-rest-apis/

Ne zaman REST kullanılmalı?

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- REST API'leri mikroservis temelli uygulamalar için en fazla tercih edilen yöntemlerden biridir.
- Yüksek hız gerektiren uygulamalarda, REST API ideal çözümlerden biri olarak ele alınabilir. Birçok üçüncü parti REST API aracı mevcuttur ve bunlar kullanılarak HTTP protokolünün hızlı iteratif yapısından faydalanılabilir.
- Bulut uygulamalarında, örneğin Kubernetes ve OpenShift uygulamalarında, çalıştırılan podlar geçici (ephemeral) nitelikte ve durum bilgisi olmayan (stateless) yapılar olması ölçeklenebilmeyi sağladığından, RESTful API'ler tercih edilmektedir.
- Birden fazla istemci türünün aynı RESTful API'sini kullanabilmesi mümkündür. Sadece web projeleri, iOS uygulamaları, IoT cihazları, vb sistemler üzerinden ek bir yazılıma ve çabaya gerek duyulmadan erişilmesi mümkündür. gRPC'de olduğu gibi, istemci ve sunucu tarafında kütüphanelere, derleyicilere, ek yazılımlara gerek duyulmaz.

https://hevodata.com/learn/grpc-vs-rest-apis/

Ne zaman REST kullanmalı?

- **Basit ve hafif veri aktarımlarında**: Karmaşık işlem gerektirmeyen işlerde REST API'ler çok başarılıdır. Örneğin, IoT sisteminden hava durumu sunucularına doğru olan iletişimde, IoT sistemi hızlı bir şekilde mevcut hava durumunu sorgulayabilir.
- **Mobil uygulama geliştirme**: akıllı telefonlarda ve tabletlerde çalışan mobil uygulamalarda yaygın bir şekilde kullanılırlar. Mobil iletişim kanalı üzerinden çok az veri aktarıldığından tercih edilirler. Bu şekilde veri paketinin optimum kullanımını sağlarlar.
- **Bulut temelli temelli servislerde kullanım**: SOAP API'lere göre daha hızlı bir iletişim yöntemi sunarak, verilerin bulut temelli sistemlere aktarılmasını kolaylaştırırlar.
- **Ölçekleme konusunun önemli olduğu uygulamalar**: REST API sistemleri, ölçeklenebilir sistemlerdir. Sunucunun sorguyu cevaplaması için durum bilgisi gerekmediğinden

GraphQL : API için sorgulama dili

- RESTful API'lerin sınırlamalarından biri, tek bir URI üzerinden tek bir kaynağın sorgulanmasıdır.
- GraphQL, tek bir API sorgusuyla birden fazla kaynağın sorgulanmasını sağlayan açık kaynak kodlu bir sorgulama dilidir.
- GraphQL, istemcilerin birden fazla kaynağı tek bir seferde sorgulayabilmesini ve tek bir JSON dosyası olarak sonuçları alabilmesini sağlayabilir.
- GraphQL, birçok uygulamada RESTful API'ye göre daha fazla tercih edilmeye başlandı.
- Ancak, işlemleriniz temel CRUD işlemleriyle tek bir seferde yapılabiliyorsa, RESTful API'nin kullanılması daha basit olabilir.

https://www.airplane.dev/blog/graphql-vs-rest-apis-a-complete-guide

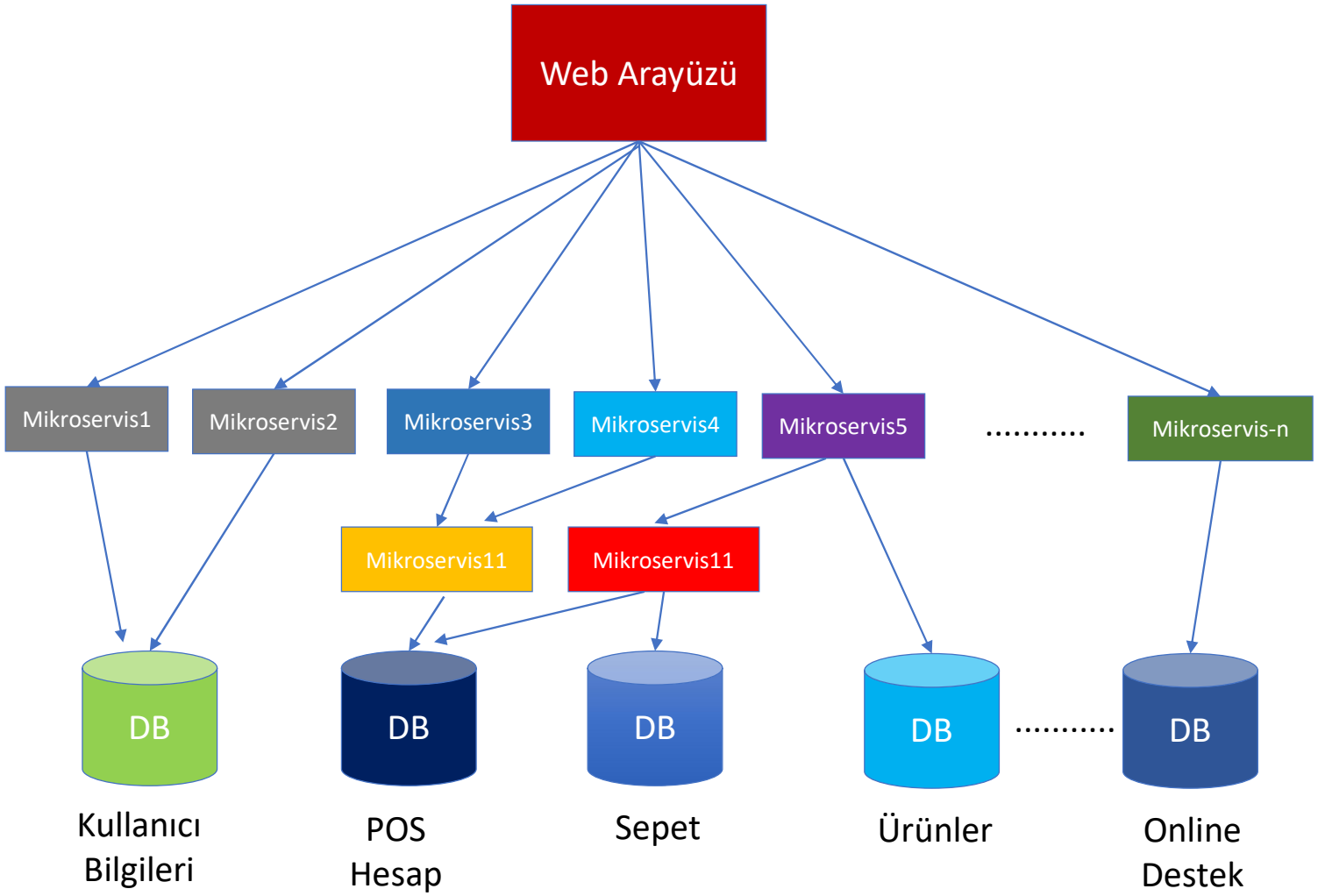
Mikroservis Mimarisi

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- SOAP ve REST'te, servisler içinde çok sayıda işlem yapılabilir ve çok fazla sonuç dönülebilir.
- Mikroservis mimarisi, bir uygulamanın çok sayıda küçük servisten oluşacak şekilde inşa edilmesine dayanmaktadır.
- Servisler
  - Hızla güncellenebilir ve test edilebilirler
  - Bağımsız çalışabilirler (çalışması iyi olur)
  - Diğer servislerden bağımsız şekilde çalıştırılabilirler
  - Küçük takımlarca hazırlanır
- Uygulamaların, birbirine fazla bağlı olmadan çalışan servislere bölünmesi, özellikle karmaşık uygulamaların devreye alınmasını hızlandırır ve kolaylaştırır.

https://microservices.io/

Mikroservis Mimarisi



Mikroservis Mimarisinin Avantajları

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Mikroservisler birbirilerinden soyutlanmış (genellikle) bağımsız yapılar olarak hazırlanır.
- Genellikle durum bilgisine veya önceki işlemlere dayanmadan çıktı üretirler (stateless).
- Girdileri ve çıktıları sabit kaldığında, mikroservisler üzerinde yapılan güncellemeler, sistemin çalışmasını etkilememesi gerekir.
- Kolaylıkla ölçeklenebilir.
- Hızlı bir yazılım geliştirme imkanı sağlar ve CI/CD döngüsünü hızlandırır.
- Mikroservis sanal bir makinede veya konteynerde veya birçok konteynerde izole şekilde çalıştırılabilir.
- Bulut sistemlerinde kullanılabilirler.

Dezavantajları

- Var olan bir yazılımın mikroservis mimarisine döndürülmesi, tüm sistemin yeniden yazılmasını gerektirebilir.
- Yazılım firmaları alışık oldukları mimariye bağlılıkları ve bu mimariye dayanan kurumsal kültürleri ve örgütlenmeleri nedeniyle, mikroservis yapısına geçerken zorlanabilirler.
- Mikroservis mimarileri, mikroservisler ve sunucular arasındaki ağ üzerindeki iletişime bağlıdır ve bu nedenle mesajlaşma yoğunlaşabilir.
- Test edilen bir mikroservisin girdisine ve çıktısına bakılarak birim testi yapılabilir ama tüm sistemdeki davranışı tahmin edilemeyebilir. Bu nedenle mikroservis kullanan sistemler için hata ayıklama süreci zor olabilir.
- Özellikle dağıtık yapılarda, mikroservisler arasındaki iletişimin güvenli hale getirilmesi ek çaba gerekir.

SOAP, Webservis, REST, Mikroservis: tarihçe

- SOA kavramı, 1990'ların sonunda ortaya çıktı. Amaç, bir uygulamayı daha küçük parçalara ayırmak ve bunları farklı sistemlerde tutmaktı. Ancak 1990'ların sonundan önce bu mantıklı fikrin nasıl gerçekleştirileceği bilinmiyordu.
- 1997'de IBM Enterprise Java Beans ürününü çıkardı. Küçük bir servisin web üzerinden nasıl servis edilebileceği fikrinin ilk örneklerinden biriydi.
- 1999'da Microsoft, SOAP ile HTTP protokolü üzerinden bilgi aktarımını ortaya koydu. Veriler dupleks şekilde XML formatında ve HTTP protokolü üzerinden aktarılıyordu.
- 2000'li yılların başında kurumsal piyasada her şey HTTP üzerinden yapılıyordu ve metin temelli İnternet haberleşmesi önemliydi. Bu nedenle SOAP uygulamaları çok başarılı oldu.
- 2005-2007 arasında insanlar SOAP'un gitgide karmaşıklaşan yapısını reddetmeye başladılar. 2008'den 2010'a kadar olan süreçte çok daha basit olduğu için REST (Representational State Transfer) gündeme geldi ve popüler oldu. RESTful temelli web servisleri yaygın şekilde kullanılmaya başlandı.
- Mayıs 2011'de Venedik'te bir çalıştay sırasında ilk olarak mikroservis kelimesi kullanıldı. 2012'den itibaren artık herkes bu kelimeyi kullanmaya başladı. O dönemde yaygınlaşan DevOps felsefesiyle de uyumu sebebiyle benimsendi ve bir çeşit mimari olarak ortaya çıktı.
- SOAP, 2000'li yılların ortalarında web servis kavramıyla birleşti ve mikroservisle eş anlamlı hale gelmeye başladı.

https://www.dataversity.net/a-brief-history-of-microservices/

Güvenlik

- API tasarımında en önemli sorunlardan biri güvenliğin nasıl sağlanacağıdır.
- «API Security: Latest Insights & Key Trends: 2022 Research Report» başlıklı rapora göre:
  - API'ler hazırlanırken, kodlamaya kıyasla güvenlik konularına zaman harcanmıyor.
  - Firmaların %50'si son 12 ayda API güvenlik sorunuyla karşılaşmış.
  - API'lerdeki yapılandırma yanlışları ve alınmayan güvenlik tedbirleri büyük sorun oluşturur. Karşılaşılan sorunların %40'ını yapılandırma yanlışları, %35'ini güncellenmeyen API'ler ve %34'ünü de botlar/spam'lerle yapılan saldırılar oluşturuyor.
  - Firmaların %53'ü karşılaşılan güvenlik sorunları nedeniyle sürüm çıkartmayı geciktirdikleri bildirilmiştir.
  - API'lerdeki sorunların %62'si prod ortamında fark edilmiştir.
  - Firmaların %59'unda gelişmiş API güvenlik tedbirleri bulunmuyor ve gelişmiş tedbirlere ihtiyaç duyduklarını belirtiyorlar.

https://services.google.com/fh/files/misc/google_cloud_api_security_research_report.pdf?hl=ko

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

Sanallaştırma, İşletim Sistemi Sanallaştırma, Konteyner teknolojileri, Docker, vd….

Fiziksel Sunucu (Bare metal)

- Fiziksel sunucu üzerinde çalıştırılan sistemler, donanımın tüm imkanlarını kullanırlar.
- Ancak, kurulu olan yazılım donanımın sağladığı tüm kapasiteyi kullanamadığında atıl kapasite ortaya çıkar.
- Her bir sunucu sistem için ayrı bir sistem tahsis edilmesi, güvenlik açısından olumlu olabilir.
- Ekonomik açından bakıldığında, birçok servisi gruplandırarak aynı makineye kurmak iyi bir çözüm olabilir.
- Fizik sunucular üzerinde konulan farklı servislerin (web, ftp, e-posta, vb) kapasite kullanım oranları zamanla değişebilir. Bunların yönetilmesi kolay değildir. Örneğin, eposta sistemini başka bir sisteme aktarmak için yeni bir makinenin ve e-posta yazılımının kurulması ve ardından verinin aktarılması gerekebilir.
- Sanallaştırma, bu tür sorunların çözümü için IT yöneticilerine büyük imkan sağlar.

Sanallaştırma nedir?

- Sanallaştırma, donanıma doğrudan bağlı çalışan birden çok bilgisayar sisteminin, tek bir donanım içinde birbirinden **izole** şekilde çalıştırılabilmesini sağlar.
- Bu şekilde, tek bir donanım üzerinde çalışan uygulamaların/işletim sisteminin atıl kapasitesi, diğer sanal uygulamaları çalıştırmak kullanılabilmekte ve **kaynaklardan daha verimli faydalanılmaktadır**.
- Ticari veya kamusal bulut mimarileri üzerinde uygulamaların çalıştırılabilmesini ve yönetilmesini sağlayan **bulut sistemlerinde sanallaştırma önemli bir yapı taşıdır**.
- Günümüzde sanallaştırma sistemleri, işletim sistemine bağlı olmadan sanal makineleri çalıştırabilme özelliğine ulaşmıştır.
- Sanal makineler, üzerindeki durum bilgileriyle birlikte kaydedilebilir ve diğer makinede kaldığı yerden devam edecek şekilde çalıştırılabilirler.

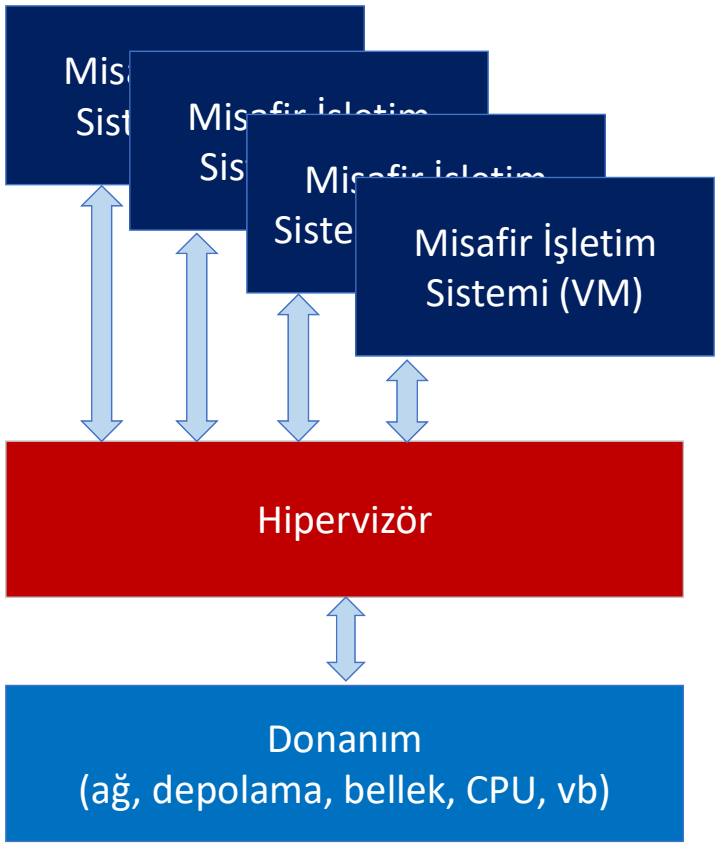
https://www.redhat.com/en/topics/virtualization/what-is-virtualization

Sanal Makine Nedir?

- Bir sanal makine (VM) izole edilmiş bir bilgisayar ortamıdır.
- Sanal makine, donanımla arasında duran hipervizör yazılımı tarafından kendisi için oluşturulmuş sanal CPU'ları, bellek ve depolama birimlerini kullanarak işlev görür.
- Hipervizör, sanal makineyi fiziksel altyapıdan ayırır ve bu fiziksel kaynakları sanallaştırarak, çalıştırdığı sanal makineler arasında paylaştırır.
- Hipervizör, sanal makinelerden gelen talepleri sıraya sokarak, fiziksel kaynakları ortak şekilde kullandırır.
- Sunucu sistemin kaynakları sanallaştırma sayesinde daha verimli çalışır.

Hipervizör – Tip 1

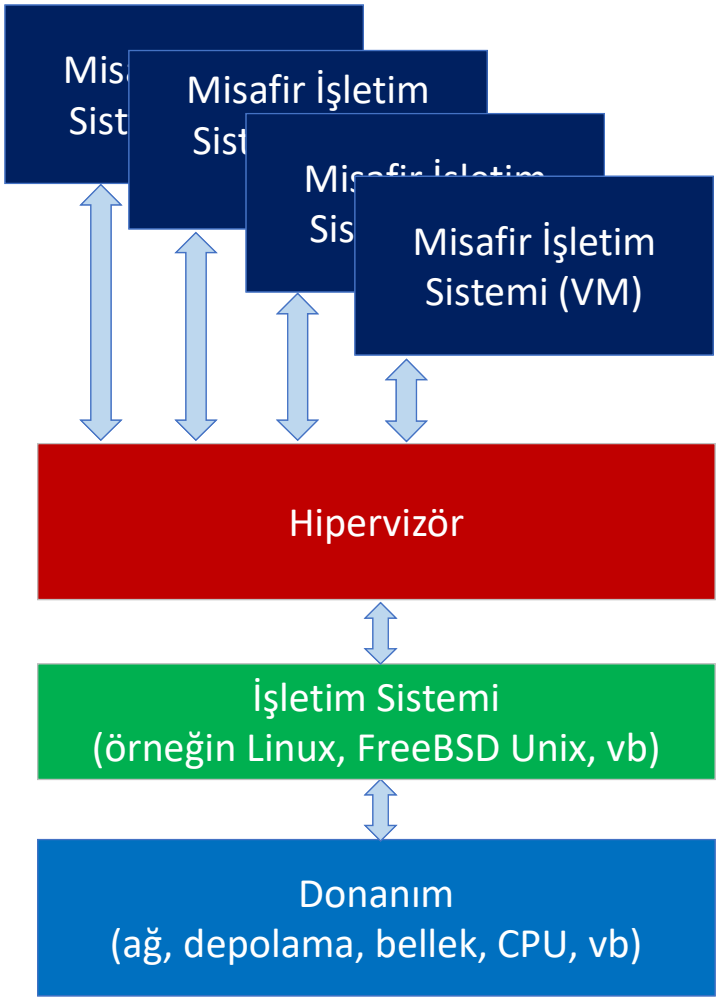
- Tip-1 hipervizör, sanal makineleri oluşturan ve çalıştıran yazılımdır.
- Hipervizör, donanım birimlerini sanallaştırarak, her bir misafir makinenin, konuk (host) makinenin kaynaklarını paylaşmasını sağlar.
- Donanım kaynaklarını sanallaştırarak, misafir makinelere paylaştıran Tip-1 hipervizör kavramı yeni bir kavram değildir. IBM, Sun gibi şirketlerin çok uzun süreden beri sundukları ürünler bulunmaktadır.
- Bu tür hipervizörler, kurumsal veri merkezlerinde kullanılır. Linux KVM, Microsoft Hyper-V, VMware vSphere, Tip1 hipervizörlere örnek verilebilir.



https://www.redhat.com/en/topics/virtualization/what-is-a-hypervisor

Hipervizör – Tip 2

- Tip-2 hipervizör, bir işletim sistemi üzerinde çalışan hipervizör yazılımıdır.
- Hipervizör, işletim sisteminin kaynaklarını kullanarak, misafir makineleri yani sanal makineleri çalıştıran bir yazılımdır.
- Tip-2 hipervizör, üzerinde çalıştığı işletim sisteminin olanaklarını kullanarak, misafir donanım birimlerini sanallaştırarak, her bir misafir makinenin, konak (host) makinenin kaynaklarını paylaşmasını sağlar.
- VMware Workstation, Oracle Virtual Box, tip2 hipervizörlere örnek verilebilir.

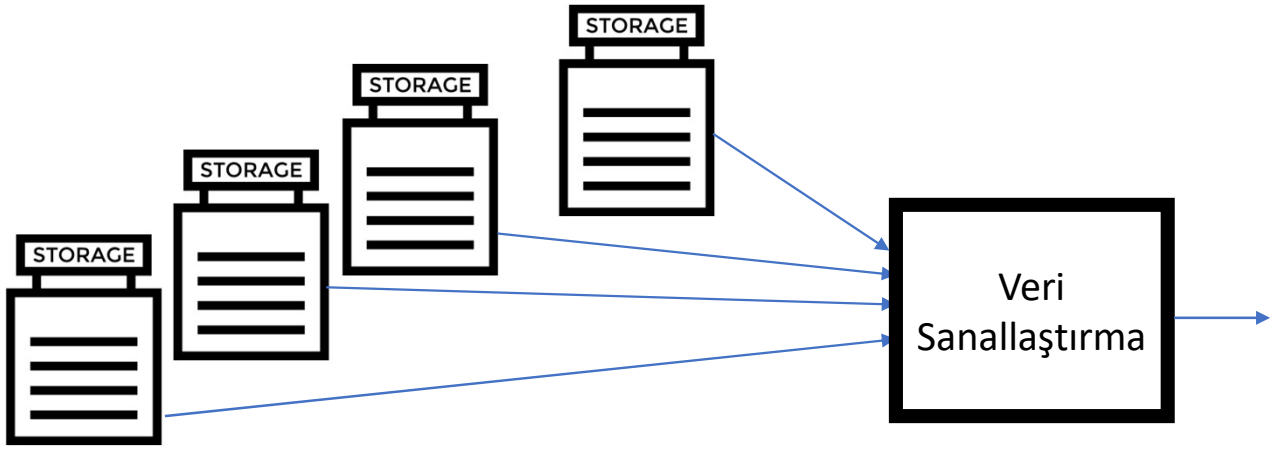


Sanallaştırma Türleri

- Sanallaştırma, fiziksel bir ortamın fiziksel bağlamından koparılması, bütünleştirilmesi ve paylaştırılması anlamına gelmektedir. Sadece sunucuların sanallaştırılması olarak düşünülmemelidir.
- Bu ilkenin uygulandığı farklı sanallaştırma türleri bulunmaktadır:
  - Veri sanallaştırma
  - Masaüstü sanallaştırma
  - Sunucu sanallaştırma
  - İşletim sistemi sanallaştırma
  - Ağ sanallaştırma

Veri sanallaştırma

- Veri sanallaştırma, farklı fiziksel veri depolama ünitelerinde yer alan kapasitenin, bir hipervizör sayesinde konsolide edilerek tek bir kaynak olarak gösterilmesi anlamına gelir.
- Hipervizör çalıştıran bilgisayar sistemi, bütün depolama ünitelerine bağlanır.
- Dışarıdan kendisine bağlanan bütün istemcilere, kendisini tek bir depolama ünitesi olarak gösterirken, gelen talepleri farklı ünitelere paylaştırılır.
- Bu şekilde farklı marka-model ve kapasitedeki sistemlerde, veri sanallaştırma sistemi sayesinde tek bir depolama sistemi olarak gösterilir ve kullandırılır.



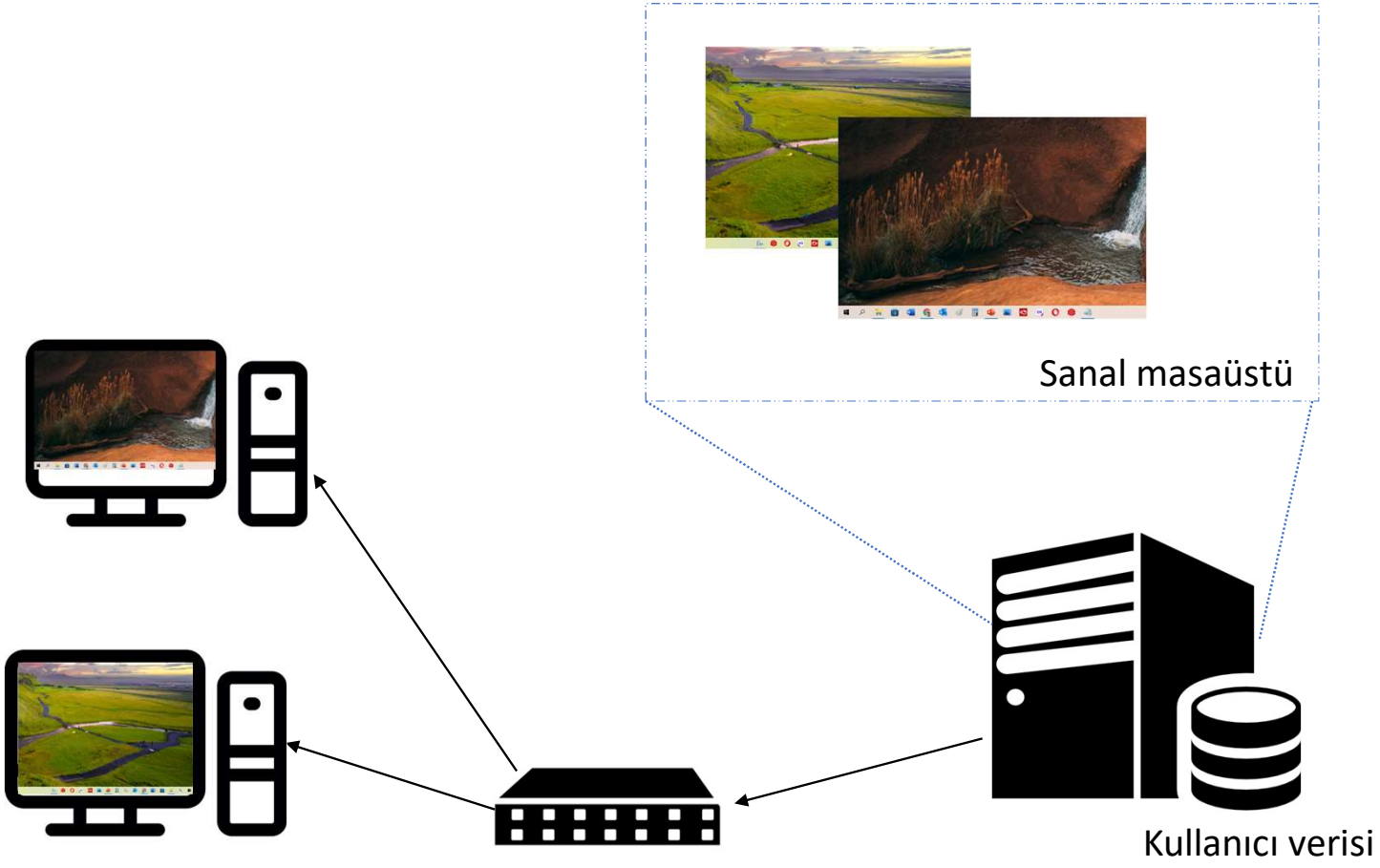
https://www.flaticon.com/free-icon/storage_151298

Masaüstü sanallaştırma

- İşletim sistemi sanallaştırmasıyla karıştırılan bir türdür.
- Kullanıcının kullandığı bilgisayar, uzaktaki sunucularda oluşan sanal masaüstü ortamını göstermek için kullanılan araçlardır.
- Uzaktaki sunucularda oluşan sanal masaüstü ortamı, kullanıcı tarafından istendiği gibi değiştirilebilir ve sanki yerel makinede çalışıyormuş izlenimi verilir.
- Sistem yöneticileri, sanal masaüstü ortamlarını hızla kurabilirler ve yapılandırabilirler.
- Güvenlik ve yönetim açısından kolaylık sağlar.

https://www.redhat.com/en/topics/virtualization/what-is-virtualization

Masaüstü Sanallaştırma



https://www.flaticon.com/free-icon/server_30454 http://cliparts.co/clipart/2346240 https://www.flaticon.com/free-icon/computer_73443

Sunucu Sanallaştırma

- Sanallaştırmanın en önemli bileşeni, hipervizör adı verilen yazılımdır.
- Hipervizör, fiziksel kaynaklarla (CPU'lar, CPU zamanı, sabit diskler, USB bağlantılar, grafik kartları, vb) sanal makineler arasında bir köprü oluşturmakta ve sanal makinelerin fiziksel kaynaklara erişimlerini kendi üzerinden sağlamaktadır.
- Hipervizörler, işletim sisteminin üzerinde bir katman olarak çalışırlar ve donanım üzerine bir sunucu yazılımı gibi yüklenirler.
- Hipervizörler, fiziksel kaynakların tamamını yönetir, sanal makinelerin doğrudan donanım yerine sanal donanım birimlerini kullanmasını sağlar.

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

Sanallaştırma nasıl yapılır?

- Sanallaştırma yapılırken, bir sanal makinenin donanımsal olarak ihtiyaç duyduğu bütün çağrılar da sanallaştırılmalıdır:
  - Disk ve ağ cihazlarının sanallaştırılması
  - Kesmeler (interrupts) ve zamanlayıcılar (timers)
  - Ana kart, veri yolları (PCI, SCSI, PCI-e, vb), BIOS
  - Öncelikli işlemler ve sayfa tabloları (bellek erişimi)
- Bu bileşenlerin hepsi sanallaştırılmalı veya para-sanallaştırılmalıdır.

Sanallaştırma

- Mikroişlemcilerin tasarımından gelen donanımsal destek olmaksızın sanallaştırma yapılması oldukça zor bir işlemdir. Ayrıca İşletim sistemlerinin kaynak kodlarında da değişikliklere gerek duyulur.
- İşletim sistemlerindeki çekirdekler, donanım üzerinde en yetkili işlemleri doğrudan yapabilecekleri düşüncesiyle hazırlanırlar. Örneğin, işletim sistemleri sayfa tablolarında yapılan korumalı işlemleri doğrudan gerçekleştirirler.
- Hipervizör donanımla işletim sistemi arasında gireceğinden, sanallaştırma hem donanım hem de işletim sistemlerinde değişiklikleri gerektirir.
- Sanallaştırma gelir kaybı anlamına geldiğinden işlemcilerde sanallaştırmaya imkan sağlayacak tasarım değişikliklerini donanım üreticileri yapmak konusunda isteksiz davranabilirler.
- Bu durum özellikle x86 mimarisinde görülmüştür. Donanım desteği olmamasına rağmen X86 mimarisine dayanan platformların sanallaştırılmasını, «**binary translation**» yöntemiyle ilk hayata geçiren VMware firması oldu.

Sanallaştırma

- Sanallaştırmaya donanım desteğinin olmaması ve işletim sisteminin Ring 0'da çalışması, x86 mimarisinde iki farklı yaklaşımın benimsenmesine sebep oldu.
  - VMware'in geliştirdiği «binary translation» yöntemiyle, donanım desteği olmaksızın yetkili işlemlerin daha düşük yetkilerle gerçekleştirilmesini sağlayan sanallaştırma katmanı, diğer bir deyişle hipervizör sayesinde gerçekleştirilen tam sanallaştırma
  - XenSource şirketinin işletim sistemindeki sürücülerin sanallaştırmaya uygun hale getirilmesiyle gerçekleştirilen para-sanallaştırma
- 2005 yılında Intel firması sanallaştırmaya donanım desteği sağlamak üzere VT-x ve 2006 yılında da AMD firması AMD-V komut setlerini içeren işlemcileri piyasaya sürdüler.

Tam-Sanallaştırma (Full Virtualization)

- Tam sanallaştırma, sanal makinenin doğrudan donanıma erişmesini sağlarken hipervizörün de devrede olmasını gerektirmektedir.
- Hipervizör, sanal makinelerin birbirinden izole olmasını ve bu ortamda çalışmasını sağlamaktadır.
- Ancak, işletim sistemleri genelde tasarım itibariyle, en yetkili (priviledged) işlemleri (örneğin bellek yönetimi kapsamında sayfa tabloları (page table) işlemlerini gerçekleştirebilecek şekilde yazılmaktadır.
- Araya girilmesi için donanım desteği gerekmiş ve AMD/Intel 2005/2006 yıllarında sanallaştırmayı destekleyen mikroişlemci ürünlerini piyasaya sürdüler.

Para Sanallaştırma

- Önemli bir kavram olan «para-sanallaştırma», işletim sistemlerini donanım üzerinde sanal şekilde çalıştırmak için, işletim sistemi üzerinde değişiklik yapılmasına dayanan bir çözümdür.
- Xen projesinde, sanal makineyi çalıştırmak için konuk işletimindeki G/Ç sürücüleri değiştirilerek hipervizör üzerinden donanıma erişerek istenen işlemleri yapması sağlanmıştır.
- Değişikliklerin yapılması için XenSource, Intel, AMD ve Microsoft gibi firmalarla birlikte çalışmıştır.
- Para-sanallaştırma, işletim sisteminde değişiklik gerektirdiğinden ve donanım üzerinde yapılan işlemlerin hipervizör üzerinden gerçekleştirilmesi nedeniyle, tam sanallaştırmaya göre daha yavaş olduğu öne sürülmektedir.

https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf

Donanım Destekli Sanallaştırma

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Mikroişlemci üreticileri, kendi cihazlarının sanallaştırmayla etkin şekilde kullanılmasının gelir kaybına sebep olacağını biliyorlardı.
- İşletim sistemi satan geliştiriciler de sanallaştırmayı birden fazla işletim sistemi lisansı satma fırsatı olarak gördüler. Microsoft gibi firmaların para-sanallaştırmaya verdikleri destek bu sebeple oldu.
- Piyasadaki kullanıcıların talepleri ağır bastı ve özellikle yüksek işlem kapasitesi sunan işlemcilerde değişikliklere gidilerek sanallaştırma desteği sağlandı.
- Bu tür donanımsal özelliklerin devreye girmesiyle Xen ve VMware başta olmak üzere çözüm sunan bir çok şirket, 2006-2007'den sonra, donanım desteğini kullanacak şekilde hipervizör yazılımlarını güncellediler.

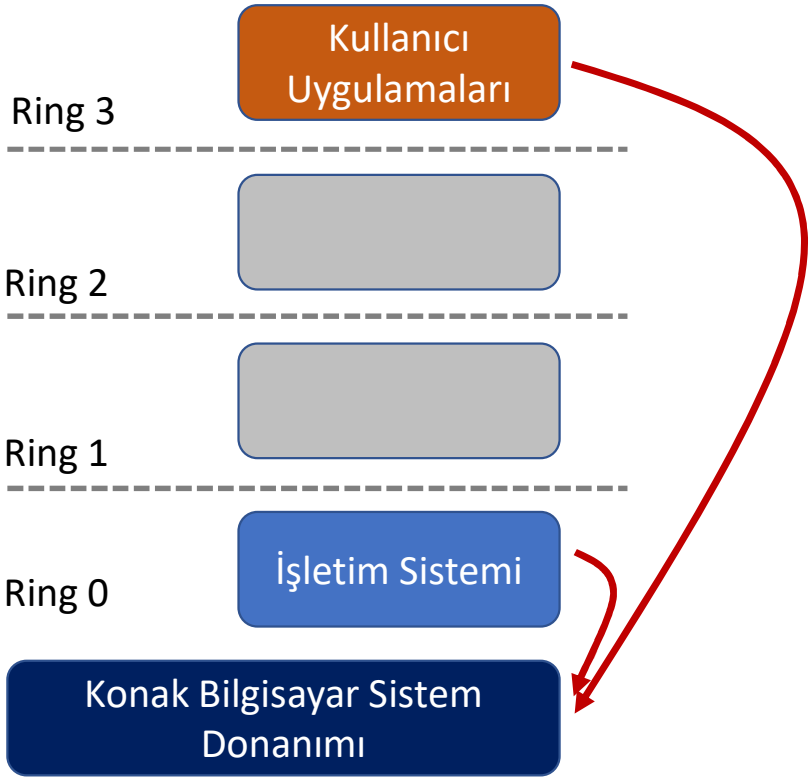
VMware 1999'da sanallaştırmayı nasıl başardı?

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- 2005 senesinden önce x86 mimarisinde tam sanallaştırma mümkün değildi.
- VMware, x86'da ring 0'da çalışan işletim sistemini, daha düşük seviyede çalıştırabilmek için, «**`binary translation`**» adı verdikleri bir yöntem kullandı.
- «**`binary translation`**» işletim sisteminin kodu içinde yetkili komutları (priviledged instruction) yakalayarak, onları ring 0 yerine ring 1'de çalıştırmak için kodu çalışma esnasında değiştirmek (code patching) anlamına gelir.
- VMware'in bu ilk sanallaştırma çözümünde «kod değiştiren kod» tehlikeli bir durum yaratmaktadır.
- x86'da çalışan bu yöntem için fikri mülkiyetlerini korumak için patent aldılar.

Geleneksel x86 mimarisi (2005 öncesi)

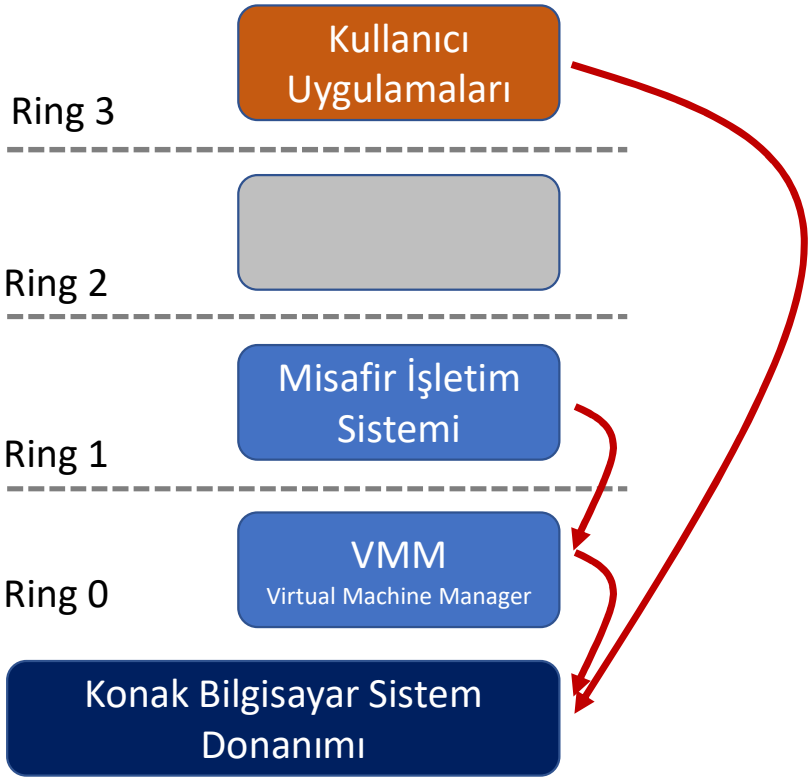
- 2005 öncesi geleneksel x86 mimarisinde, donanım üzerinde çalışan işletim sistemi, Ring 0'dan 3'e kadar olan makine komutlarını çalıştırabilmektedir.
- Yüksek öncelik gerektiren komutları çalıştırabilmesi, işletim sisteminin doğrudan donanıma erişmesini sağlamaktadır.
- Ancak, olası bir hipervizörün de Ring 0'daki yetkili komutları çalıştıracak olması, hipervizörün işletim sistemiyle donanım arasına girmesini engellemektedir.
- Burada olması gereken, Misafir işletim sisteminin Ring 1'deki komutları çalıştırması ve Ring 0'da çalışan hipervizörün, işletim sisteminin yetkili komutları çalıştırdığı zaman devreye girmesini (trap'ler yoluyla) sağlamaktır.
- Ancak x86 mimarisi buna izin vermemektedir.



https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf

VMware'in tam sanallaştırma çözümü

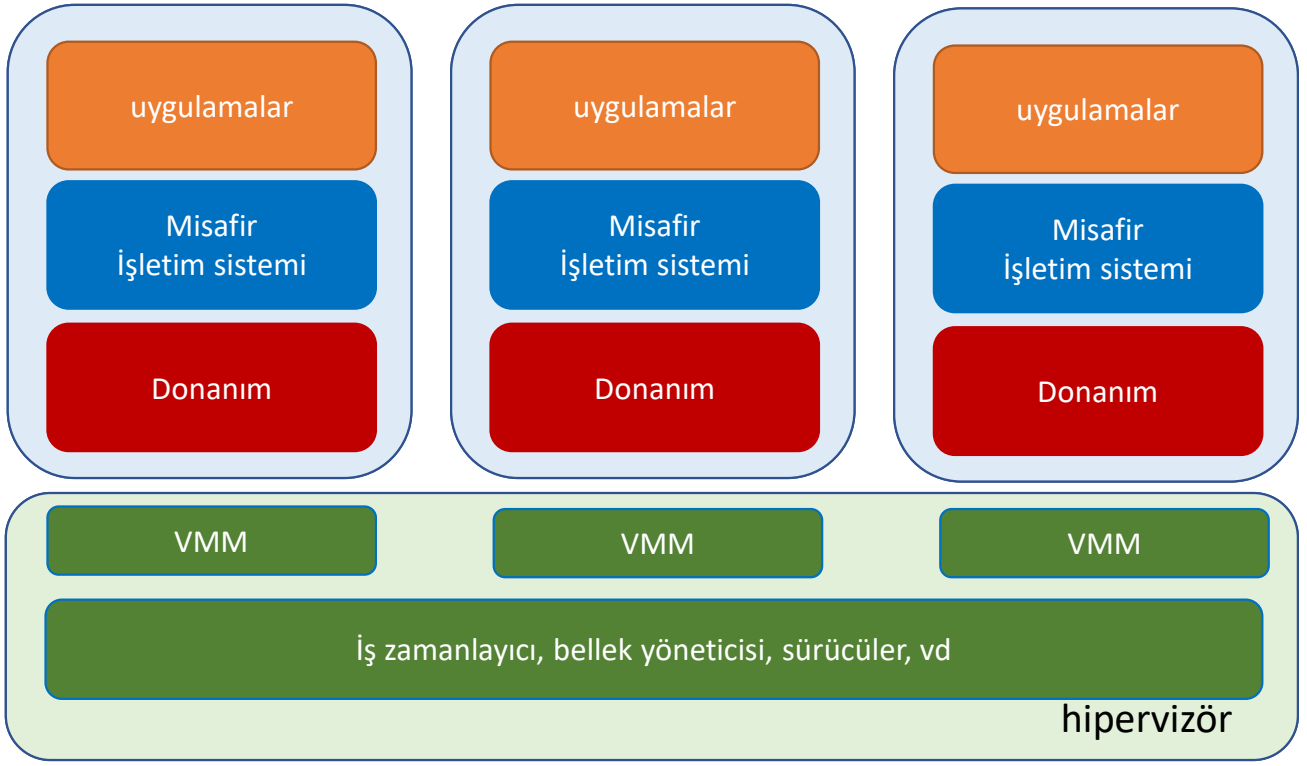
- İşletim sisteminin x86 mimarisinde Ring 0'da çalışması, hipervizörün araya girememesine ve sanallaştırmanın yapılamamasına sebep olmaktadır.
- VMware bunu çok farklı şekilde çözümlemiştir.
  - İşletim sistemi yine Ring 0'da çalışmaktadır. Ama VMware hipervizörü, çalışırken işletim sisteminin koduna müdahalede bulunarak, Ring 0'daki komutların yerine hipervizörü çağıran kodların konulmasını sağlamaktadır.
  - Kısacası, VMware hipervizörü, işletim sistemi kodunu yamamakta ve kodu canlı şekilde değiştirmektedir.
  - KOD DEĞİŞTİREN KOD!!!!!



https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf

VMware tam sanallaştırma çözümü

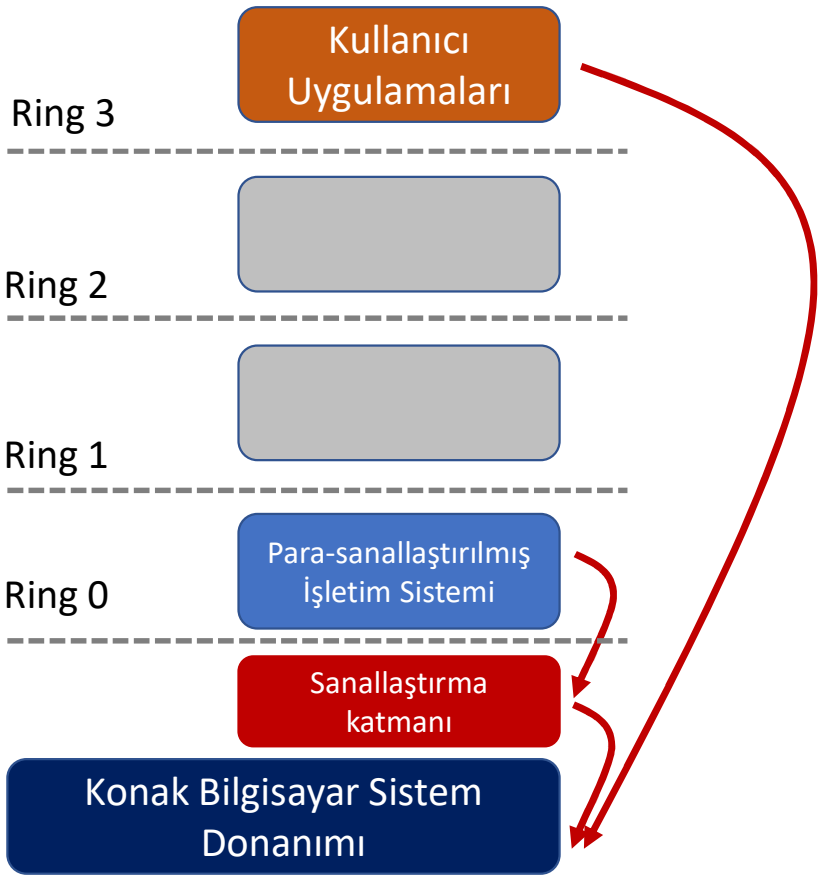
- Her bir misafir işletim sistemine VMM eklenmekte ve çalıştırılacak her bir işletim sistemi VMM üzerinde çalışacak şekilde çalışırken yamanmaktadır.



https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf

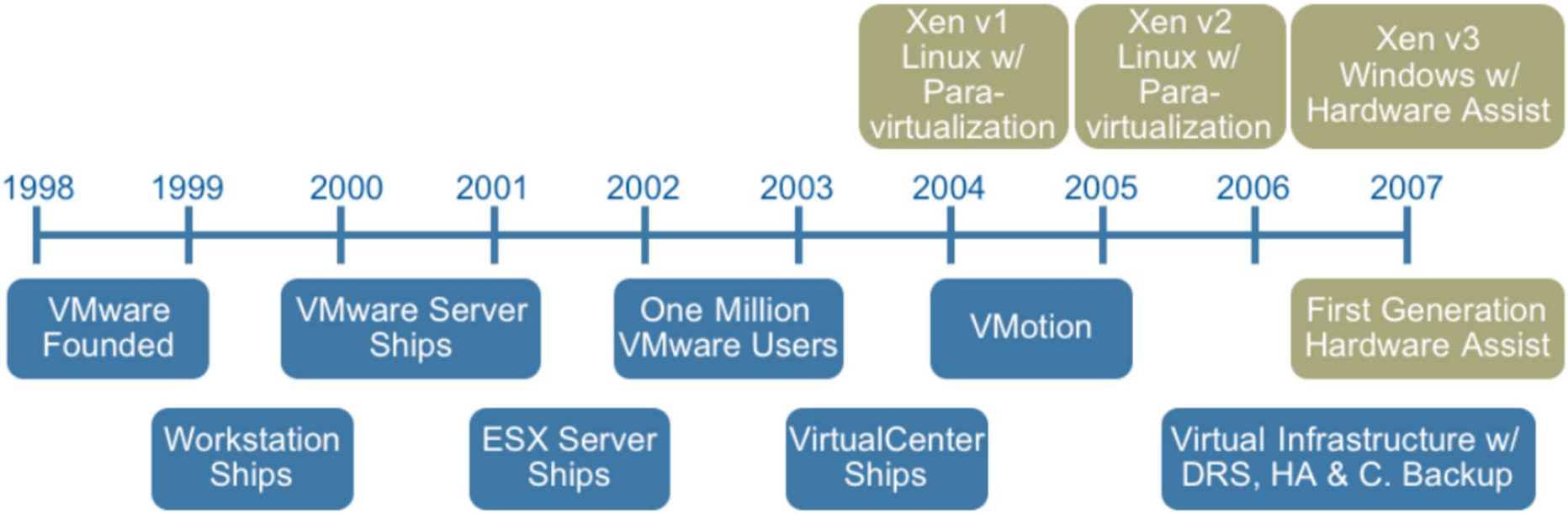
Para-Sanallaştırma

- Para sanallaştırma çözümünde, işletim sisteminde sanallaştırmaya uygun şekilde tasarlanmış sürücüler kullanılmaktadır.
- Diğer bir deyişle, işletim sisteminin donanıma doğrudan değil sürücüler üzerinden erişimi sağlanmalıdır. .
- Donanıma, sürücüler üzerinden ulaşılmakta ve bu özel sürücüler yapılacak işleri «sanallaştırma katmanına» iletmektedir.
- «sanallaştırma katmanı», donanımsal komutları çalıştırıp sonucu, sürücüler üzerinden işletim sistemine döndürmektedir.



https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf

Sanallaştırma Tarihçesi



https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf

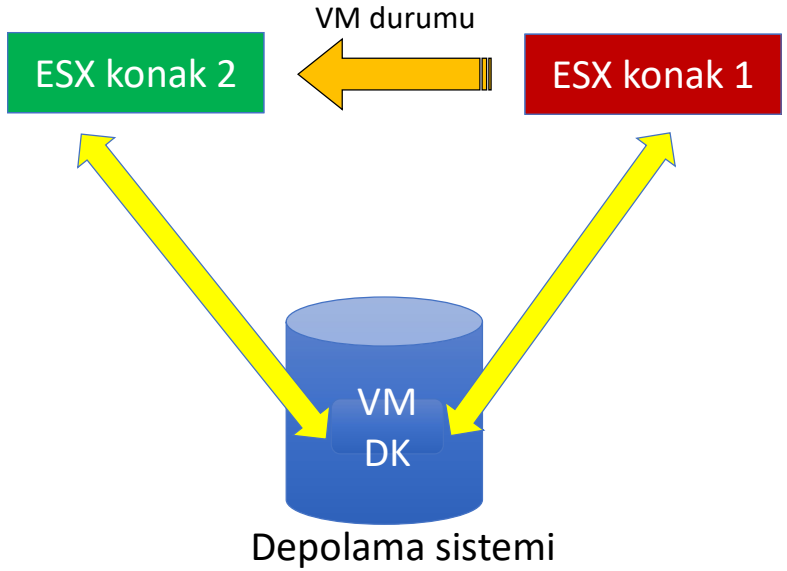
Xen

- Xen, tip-1 hipervizörü olan açık kaynak kodlu bir sanallaştırma platformudur.
- Kurumsal ortamlardaki veri merkezlerinde kullanılan bir çözümdür.
- Xen, Cambridge Üniversitesinde, Ian Pratt ve doktora öğrencisi Keir Fraser tarafından yürütülen bir araştırma projesi sonucunda geliştirildi. Projede çalışanlar daha sonra XenSource isminde bir şirket kurdular.
- Xen yazılımının ilk sürümü Ekim 2003'te sunuldu ve ikincisi bir sene sonra çıkarıldı.
- 2005'te sunulan v3.0'la birlikte, kurumsal olarak kullanılan birçok işlemci (Intel Xeon, PowerPC, IA-64, vb) ve işlemci komut tablosuna eklenen Intel VTx, AMD SVM gibi sanallaştırma uzantıları desteklendi.
- 2007'de XenSource'u Citrix firması satın aldı.
- 2008'de çıkarılan v3.1 ile birlikte «Live Migration» özelliği kazandı ve veri merkezlerinde kullanılmasının önü açıldı.
- 2013'te ağ sanallaştırma kapsamında sanal anahtar desteği sağladı.
- Xen'in hipervizörü, açık kaynak kodlu olarak sunulan tek hipervizördür.
- Xen'in hipervizörü, mikroçekirdek olarak tasarlandığı için 1MB gibi küçük bir bellek alanını kaplamaktadır.

https://wiki.xenproject.org/wiki/Xen_Project_Software_Overview

Vmware vMotion

- vMotion, çalışan sanal bir makinenin fiziksel bir sunucudan diğerine aktarılarak hizmette kesintiye uğramadan çalışmasının sağlanmasıdır.



- vMotion'ın bir çok faydası bulunur:
  - Fiziksel sunucudaki bir arıza sonrasında, üzerinde çalışan sanal makinelerin diğer fiziksel makinelere kesinti olmaksızın aktarılması
  - Fiziksel makine üzerindeki işlem yükü arttığında, bazı sanal makinelerin diğer sunuculara aktarılarak yükün dengelenmesi

https://www.vmware.com/products/vsphere/vmotion.html

vMotion nasıl yapılır?

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Sanal makineler, çalışma esnasındaki durum bilgisi (BIOS, aygıtlar, CPU'daki yazmaçlar, Ethernet kartları için MAC adresleri, IP adresleri, vb) periyodik olarak VMDK uzantılı imaj dosyalarına kaydedilir.
- Sanal makinenin durumu diğer fiziksel sunucuya aktarıldığında, eğer bu sunucu da sanal makinenin VMDK dosyasına erişebiliyorsa, sanal makine durum bilgisini okuyarak, sanal makineyi diğer sunucuda kesintiye uğramadan çalıştırılabilir.
- vMotion olanağını kullanabilmek için, sanal makineleri oluşturan VMDK dosyaları, VMware sunucularının erişebildiği ortak bir depolama alanına konulmalıdır.
- Bütün fiziksel sunucular imaj dosyalarının bulunduğu yere erişirler.

«Live Migration»

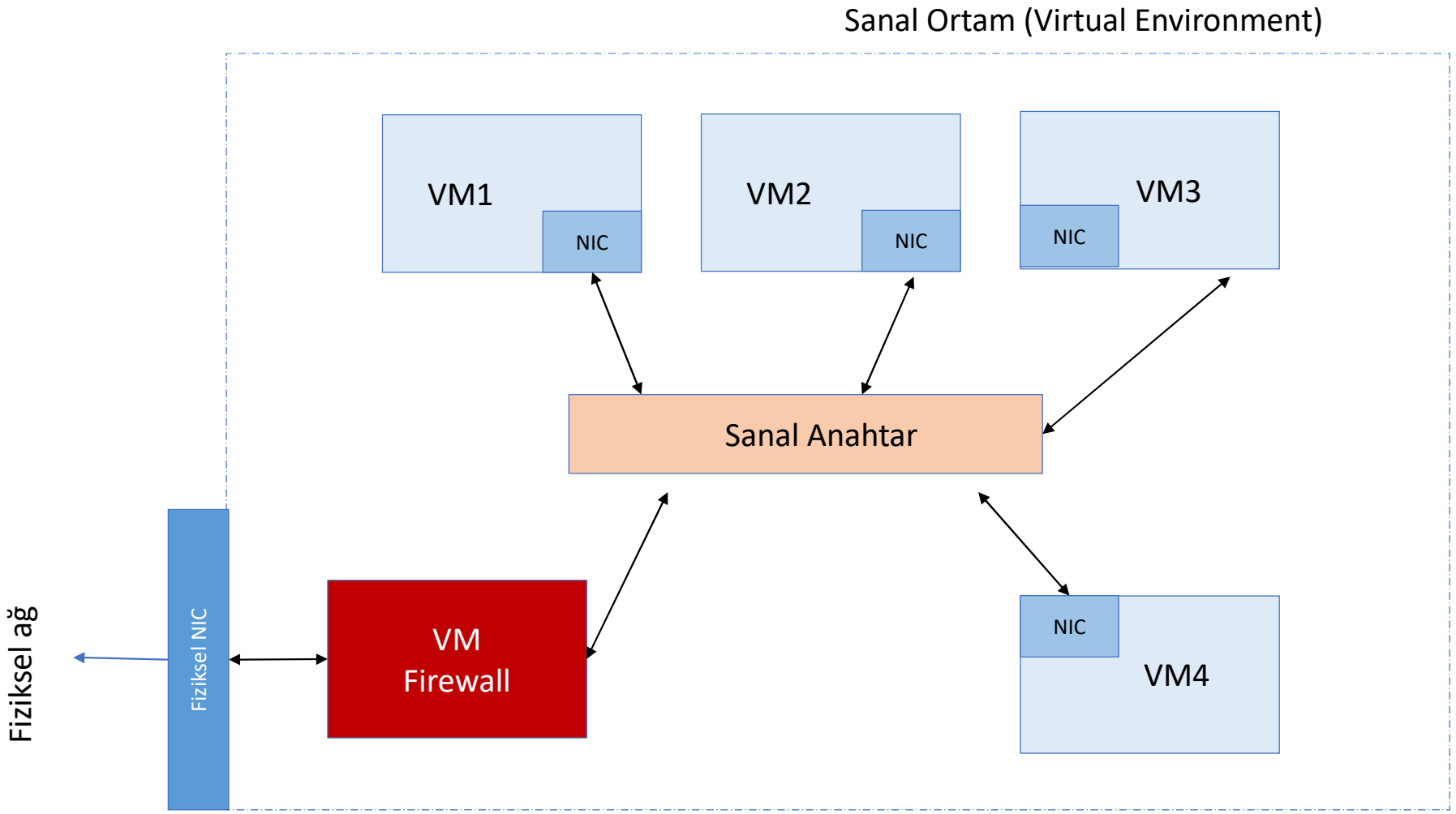
- «Canlı Göç» bir sunucuda çalışan sanal bir makinenin çökmesi durumunda (örneğin sunucu çökebilir), anlık şekilde diğer bir sunucuda kaldığı yerden çalıştırılmasıdır
- «VMware vMotion» bir canlı göç uygulamasıdır.
- Canlı göç için, iki sunucunun ortak bir depolama alanına erişimi gereklidir.
- Depolama sisteminde tutulan sanal diskte, sistemin son durumu yer almaktadır. Eğer bir sunucu çökerse, bu dosyaya erişen diğer bir sunucu, sanal makineyi kaldığı yerden çalıştırabilir.
- Bu şekilde, yüksek bulunurluk (high availability) özelliği sağlanmış olur.

Ağ sanallaştırma

- Ağ işlevlerinin sanallaştırılması, fiziksel ağ cihazları üzerindeki trafiğin sanal ortamlar üzerinden aktarılması anlamına gelir.
- Sanal sunucuların birbirleri arasında gerçekleştirdikleri ağ trafiğinin fiziksel ağ cihazları (anahtarlar, omurga anahtarları, firewall, ağ depolama cihazları, vb) üzerinde akması zordur.
- Bu nedenle, sanal makineler arasındaki iletişim sanal ağ sistemleri kullanılır ve iletişim bunlar üzerinden belirli kurallarla aktarılır.
- Fiziksel ağ bağlantılarından bağımsız hale gelen ve sanallaştırılan ağ üzerinden, sanal makineler arasında daha hızlı ve güvenilir şekilde verilerin aktarılması mümkündür.
- Ayrıca, Firewall, IPS gibi ayrı kutular halinde bulunan güvenlik cihazları da sanal ağ içinde sanal makineler olarak çalıştırılabilmektedir.
- Sanal ağ cihazları da sunuculardaki fiziksel ağ ara yüzlerine bağlanarak dış ağa bağlantı da sağlanabilmektedir.

https://www.vmware.com/content/dam/learn/en/amer/fy21/pdf/688008_Network_Virtualization_Dummies_Guide.pdf

Sanal Ağ


Sanal Ortam (Virtual Environment)
VM1
NIC
VM2
NIC
VM3
NIC
Sanal Anahtar
Fiziksel ağ
Fiziksel NIC
VM
Firewall
NIC
VM4

KVM Nedir?

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

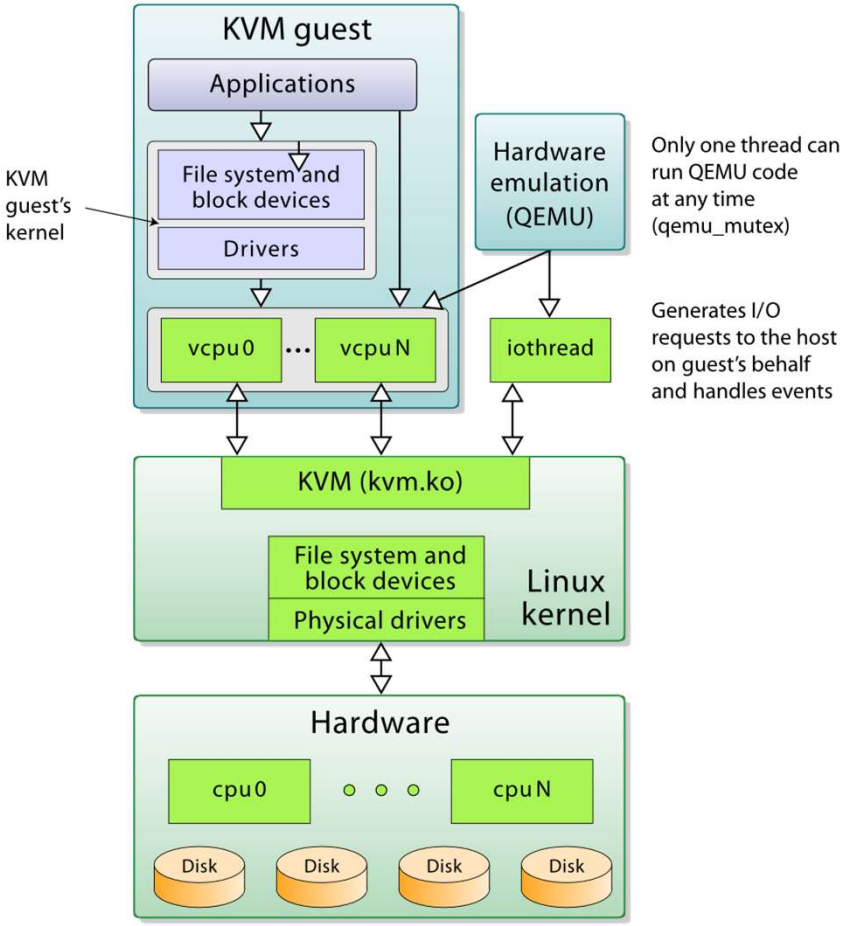
- KVM, Linux işletim sistemini tip-1 hipervizöre dönüştürür.
- KVM Linux'un bir parçasıdır. Linux v2.6'dan beri Linux'te yer almaktadır
- KVM, işletim sisteminin sağladığı kaynak ve kaynak paylaşımı olanaklarından faydalanır.
- KVM için her bir konuk sanal makine sıradan bir Linux prosestir ve Linux çekirdeği bu proseslerin çalıştırılmasını sağlar.
- Güvenlik açısından SE Linux eklerini ve güvenli sanallaştırma (sVirt) özelliklerini kullanır. SELinux, sanal makineler arasında sınır çizer ve sVirt ise SELinux özelliklerini genişleterek MAC (Mandatory Access Control) sayesinde sanal makinelerin bazı nesnelere erişimini sınırlandırır.

https://www.redhat.com/en/topics/virtualization/what-is-KVM

KVM Nedir?

- KVM, sanal makineleri fiziksel konaklar arasında hizmet kesintisi olmaksızın taşıyabilir. Böyle bir durumda ağ bağlantıları çalışır durumda kalır ve VM diğer konağa aktarılınca durumu KVM tarafından saklandığı için kaldığı yerden çalışmaya devam eder.
- KVM için sanal makine bir Linux prosesidir ve dolayısıyla sanal makineler istendiği gibi kontrol edilebilir.
- Daha üst seviyeli bir komut (sabit diske doğrudan erişim, vb) verdiğinde, Linux çekirdeği bu talebi, **`kvm.ko`** isimli bir LKM'ye yönlendirir. Bu LKM, talebi proses olarak çalışan sanal makine için karşılar ve bir hipervizör gibi sonuçların döndürülmesini sağlar.

KVM



https://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine

SORULAR

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- SORU: Linux makineyi, Linux üzerinde bir proses gibi çalıştırabilir miyiz?
- SORU: Bunu yaparken, kaynakları tüketen hipervizör yerine doğrudan Linux çekirdeğini kullanabilir miyiz?
- SORU: Bir sanal makineyi proses gibi çalıştırdığımızda, diğer sanal makinelerden nasıl izole edeceğiz?
- SORU: Linux çekirdeğini nasıl koruyacağız?
- Bu sorular 1990'lı yıllardan itibaren sorulmaya başlanmıştır.
- Soruların cevapları için zaman içinde farklı çözümler geliştirilmiştir. Bazıları Linux çekirdeğinde değişiklik gerektirmiş, bazılarının kurulumu zor olmuş, bazılarında güvenlik sorunları yaşanmış, ……
- Bu bölümde, işletim sistemi üzerinde izole şekilde VM'leri proses gibi çalıştırmak için geliştirilen teknolojileri anlatacağız ve tarihsel gelişimlerine bakacağız. Hiçbir şeyin tesadüf olmadığını, bir çok mühendisin fikirsel olarak katkıda bulunduğunu göreceksiniz.

İşletim Sistemi Seviyesinde Sanallaştırma

- İşletim sistemi seviyesinde sanallaştırma, Linux'un çekirdek kodundaki özellikleri kullanarak, farklı prosesleri aynı işletim sistemi üzerinde ama izole şekilde çalıştırma anlamına gelir..
- Linux veya diğer işletim sistemlerinin üzerine kurulur ve uygulamaların/proseslerin, konteyner denilen izole kullanıcı alanlarında ve aynı işletim sistemi üzerinde çalışması sağlanır.
- Konteyner içinde çalışan programlar, çalıştırma ortamının kendilerine izin verdiği kadar, konteyner içinde kendilerine atanmış içerikleri (özel bir kök dizin, uygulamalar, vb) ve cihazları görebilirler.
- Docker, LXC, Solaris Containers, Podman, chroot (jails) gibi sistemler işletim sistemi sanallaştırma imkanı sağlayan yazılımlardır.

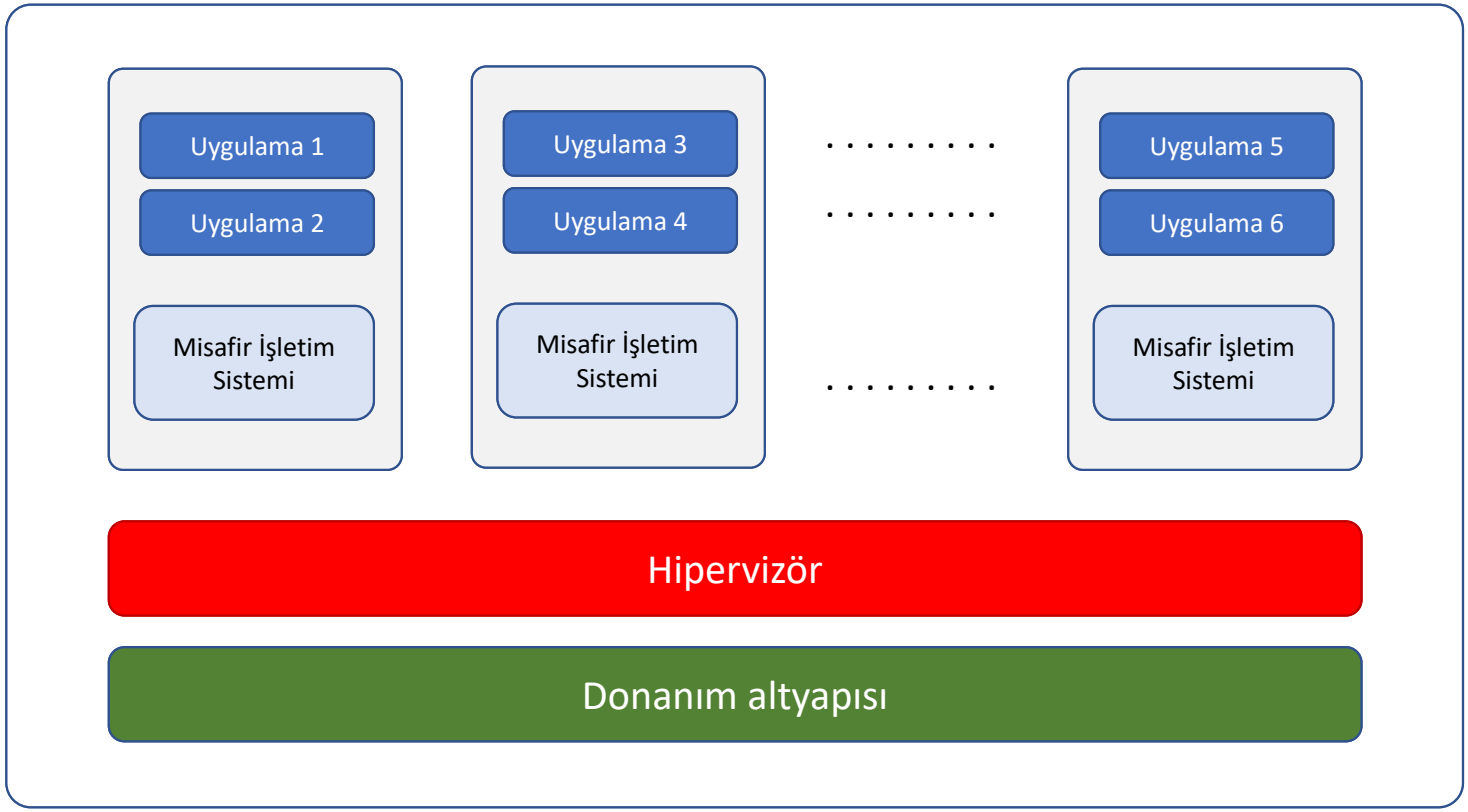
https://computer.howstuffworks.com/server-virtualization.htm

Konteyner nedir?

- Konteynerler, yazılımların izole şekilde çalışması için geliştirilen ortamlardır.
- Konteynerler, az kaynak harcayan ve alttaki işletim sistemi üzerinde çalışan modüler nitelikte sanal makineler olarak düşünülebilir.
- Konteynerlerde, diğer sanallaştırma yöntemlerinde olduğu gibi her sanal makineye işletim sistemi yüklenmesine gerek duyulmaz.
- Bir konteyner, alttaki işletim sisteminde sadece kendisinin çalıştığını görmektedir. Bu nedenle konteynerleri «hafifletilmiş» sanal makineler olarak görebiliriz.
- Her konteyner, bir imaj dosyası olarak gelir. İmaj dosyasında, konteynerdeki bir uygulamayı çalıştırmak için gerekli her türlü program, kütüphane, yapılandırma dosyası, vb bulunur. Bu nedenle diğer bir sisteme kopyalanabilir. Diğer bir sisteme aktarılabildiğinden ve istendiği şekilde çalıştırılabilir olduğundan taşınabilir niteliktedir.

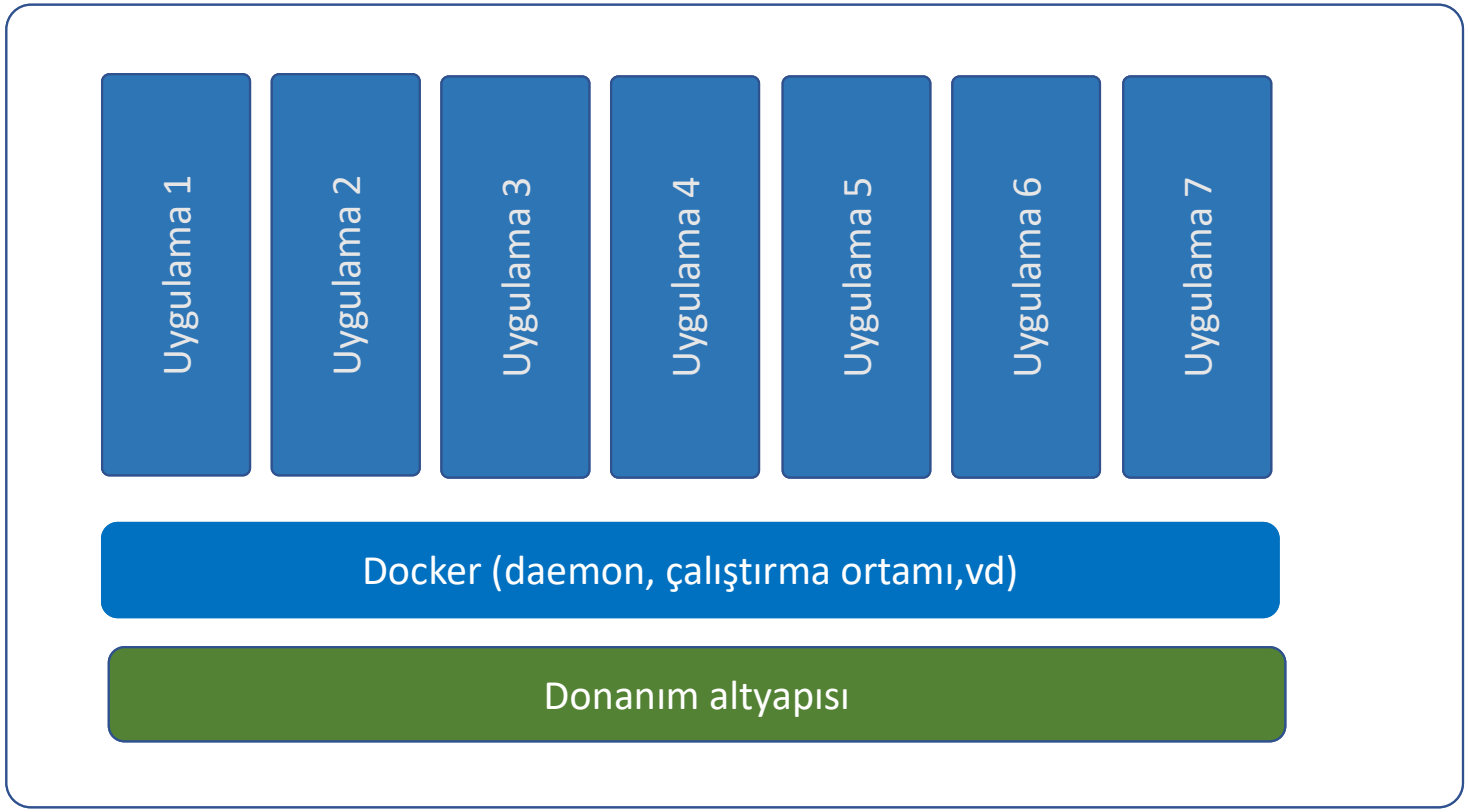
https://kubernetes.io/docs/concepts/containers/

Hipervizör Üzerinde Sanal Makineler



https://www.docker.com/resources/what-container

Konteyner



https://www.docker.com/resources/what-container

Faydaları

- Konteyner'lar, sanal makinelere göre daha hızlı çalışır çünkü klasik sanal makine konfigürasyonda, hipervizör de sistem kaynakları kullanır. Konteyner çalıştırma ortamı doğrudan işletim sistemini kullanmakta ve hipervizör bulunmamaktadır.
- Konteyner içindeki yazılım için gerekli bütün kütüphaneler ve gerekli yazılımlar, imaj içinde paketlenmektedir.
- Konteynerler, sanal makinelere göre daha basit yapılardır. Bir konteyner paketlendiğinde, gerekli bütün yazılımlar aynı paketin içindedir ve bütün makinelerde aynı şekilde çalıştırılır.
- Her işletim sisteminde bulunan konteyner çalıştırma ortamları, bu imajları aynı şekilde konteyner olarak çalıştırır.

Özetle konteyner teknolojisi

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Konteyner teknolojisinin ardında «**işletim sistemi sanallaştırma**» kavramı bulunmaktadır.
- Bir sistem üzerinde çalışan konteynerler aynı işletim sistemi üzerinde çalışırlar ama her biri izole durumdadır.
- Kütüphaneler ve uygulamalar diğer konteynerlerden ve işletim sistemindeki diğer proseslerde yalıtılmış durumdadır.
- Konteyner içinde, o uygulamanın çalıştırılması için gerekli kütüphaneler ve programların hepsi bulunur. Bu durum taşınabilirliği sağlar.
- Standart bir imajdan başlatılan bir konteyner bir Linux makinede nasıl çalışıyorsa, farklı bir distro/çekirdek sürümü/vb olan bir Linux makinede de aynı şekilde çalışır.
- Daha hızlı çalıştırılırlar.
- Çalışma ortamı üzerinde çalışan konteynerler sayesinde elastik mimariler oluşturulabilir ve sanal ağ yapıları kurulabilir.

Faydaları

- Yazılım ekiplerinin geliştirdiği ve kendi geliştirme ortamlarında çalışan yazılımın, «**`prod`**» ortamına alındığında çalışmaması sık karşılaşılan bir durumdur.
- Konteyner olarak geliştirme ortamında oluşturulan paket, «**`prod`**» ortamında olduğu gibi kullanılabilir. Çünkü bütün gerekli kütüphaneler ve ayarlar «**`prod`**» için kullanılacak konteyner içinde yer almaktadır. Yazılım ekibi tarafından oluşturulan konteyner, **`prod`** ortamına kurularak devreye alınabilir.
- Bir konteyner, diğer bir sisteme hızla aktarılabilir. Hatta bir sunucu sistemden bulut sistemine taşınabilir. Kurulum yapmaya veya konteyner imajını değiştirmeye gerek duyulmaz.
- Mikroservis mimarisiyle uyumludur. Bir programı oluşturan farklı servisler konteyner haline getirilebilir ve birbirinden bağımsız şekilde çalıştırılabilirler.
- Diğer değişle, bütün yumurtaları aynı sepete koymaya gerek yoktur. Bir sistem üzerinde çalıştırılan konteynerlerden biri bozulsa da diğerleri etkilenmez.

Faydaları

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Aynı tipte konteynerler, bir yönetim sistemi üzerinden izlenebilir ve gerekli olduğunda daha fazla konteyner devreye alınabilir veya gerektiğinde konteynerler devreden çıkarılabilirler.
- Sorunlu, hatalı çalışan veya cevap vermeyen konteynerler, yönetim sistemi sayesinde devreden çıkarılabilir ve yeniden çalıştırılabilir.
- Geliştirme ekibi yazılımın yeni sürümünü hazırlayıp farklı bir konteyner imajı oluşturduğunda, eski konteynerler devreden çıkarılıp, yenisi devreye alınabilir. Bu şekilde etkin bir geliştirme ve devre alma döngüsü oluşturulabilir.

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

Konteyner Teknolojisinin Tarihsel Gelişimi

`chroot` (Change Root)

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

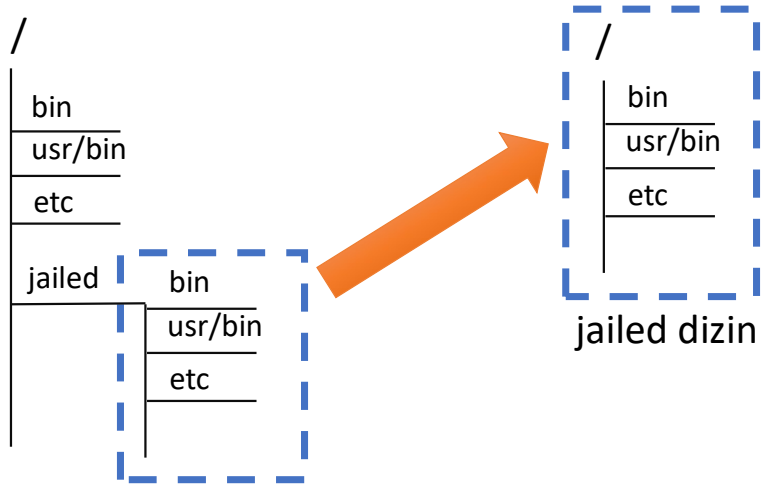
- **`chroot`**, Linux sanallaştırmada kullanılan bir kavramdır.
- **`chroot`** bir prosesin, ana kök dizine erişimini sınırlandırmak ve farklı bir dizin yapısını kendi kök dizini olarak görmesini sağlamak için kullanılır.
- **`chroot`** 1979'da Unix v7'de bulunan bir özellikti ve 1982'de BSD Unix'e de eklendi. **`chroot`** özelliği konteyner teknolojisinin öncülü olarak görülmektedir.
- Bir proses ve ondan türeyen prosesler için bir kök dizini oluşturulur ve bu prosesin kullanacağı uygulamalar buraya aktarılır. Bu şekilde, proseslerin işletim sisteminde yer alan diğer komutlara erişimi engellenir ve sadece içinde belirli programlar bulunan bir dizin yapısını görmesi ve kök dizin olarak kabul etmesi sağlanır.

https://www.geeksforgeeks.org/linux-virtualization-using-chroot-jail/?ref=lbp

`chroot` (Change Root)

- **`chroot`**, işletim sistemi seviyesinde bir nevi sanallaştırma imkanı sağlar. Aynı işletim sistemi kodu kullanılırken, proseslere sadece gerek duydukları uygulamalara ve kütüphanelere erişim sağlanır.
- **`chroot`** komutunun kullanımı oldukça kolaydır. Bir dizin oluşturulur ve gerekli uygulamalar birlikte kütüphaneler buraya kopyalanır. **`chroot`** ile çalıştırılacak programa bu dizin kök dizin olacak hale getirilir.

```
chroot /yeni/kok/dizini komut
```



https://www.geeksforgeeks.org/chroot-command-in-linux-with-examples/?ref=rp

jail (FreeBSD Jail Sistemi)

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Ancak zaman içinde, **chroot** ile belirlenmiş dizinin dışına çıkmanın değişik yöntemleri geliştirilince, güvenliği sağlamak için ideal bir çözüm olmaktan çıkmıştır. Ayrıca, **chroot**, prosesin sadece dosya sistemi erişimini sınırlandırmakta ve diğer sistem kaynakları (kullanıcı isimleri, çalışan diğer prosesler, ağ alt sistemi, vb) konusunda herhangi bir önlem almamaktadır.
- 2000'li yıllarda FreeBSD v4.x'de **jails** sistemi, **chroot**'un sınırlılıklarını gidermek amacıyla kullanıcılara sunulmuştur.
- **chroot** ile sağlanan güvensiz ortamı, güvenli kılmak için çeşitli özelliklerin **jails** bünyesine eklendiği görülmektedir.

https://docs.freebsd.org/en/books/handbook/jails/

`jail` (FreeBSD Jail Sistemi)

- «`jail`» olarak tabir edilen sınırlandırma özellikleri dört unsurun dikkate alınmasıyla gerçekleştirilmektedir:
  - Sınırlı dizin yapısı: `chroot`'un sağladığı gibi bir sınırlandırma burada da yapılmaktadır.
  - Host ismi: «`jail`» kapsamında kullanılacak konak ismi belirlenmektedir ve bu da makine isminden farklıdır.
  - IP adresi: Ana konak adresinden farklı bir adrestir ve ana makineyle aynı ağ arayüzü üzerinden verilmektedir.
  - Komut: Bütün komutlar, «`jail`» içindeki ana kök dizini referans alınarak belirtilmektedir.
- Ayrıca, herhangi bir proses için oluşturulmuş «`jail`» içinde, bu «`jail`»e özel kullanıcılar bulunmaktadır.
- «`jail`» içinde görülen «`root`» hesabı sadece bu «`jail`» içinde geçerlidir ve dışarıda bir işlem yapamamaktadır.

https://docs.freebsd.org/en/books/handbook/jails/

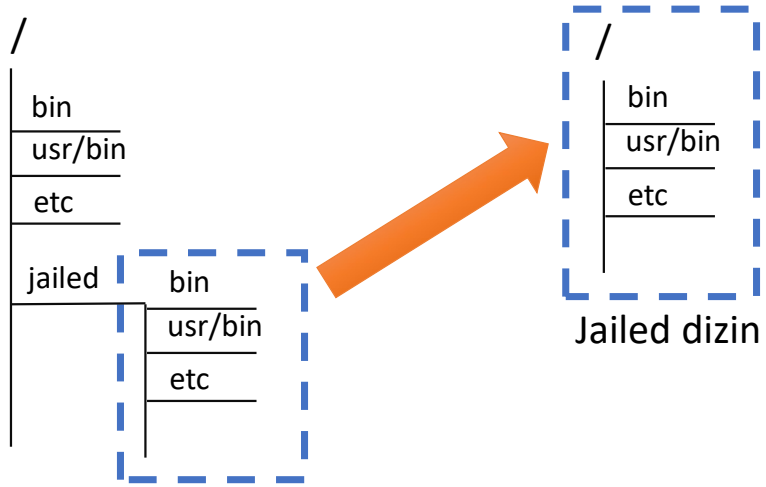
`jail` (FreeBSD Jail Sistemi)

- Ancak, «`jail`» sisteminin kullanılması için çok sayıda yapılandırma dosyasında elle değişiklik yapılması gerekliydi.
- Yapılan işlemler uzun sürüyordu ve hatasız yapılabilmesi zordu.
- «`jail`» işlemini daha hızlı yapabilmek için «`ezjail`» isimli bir program paketi geliştirildi.
- «`ezjail`» «`jail`» yaratmak için faydalı bir program olsa da yapılandırma pek çok parametrenin girilmesi sebebiyle yine de uzun sürüyordu.
- «`jail`» kavramı, konteyner teknolojileri için öncül niteliği taşıdığından oldukça önemlidir. Yaşanan sorunlar ve yapılandırmanın zorluğu, konteyner teknolojilerinin geliştirilmesi için faydalı bir deneyim olmuştur.

https://docs.freebsd.org/en/books/handbook/jails/

`chroot/jail` ve konteynerler

- Konteyner çalışma ortamının yapması gereken ilk şey, konteynerdeki sistemin göreceği kök dizini değiştirmektir. Bunun için uzun zamandan beri bilinen **`chroot`** yardımcı programı kullanılmaktadır.
- Oluşturulan sanal kök dizin (jailed root) aslında konteyner içindeki uygulamaların görebileceği konfigürasyon dosyalarını, uygulama **`exe`**'lerini ve kütüphane dosyalarını içermektedir.
- **`chroot`**, Unix veya Linux işletim sistemlerinde, bir uygulama için, ana işletim sisteminin kök dizinden farklı bir kök dizini oluşturma (jailed root) anlamına gelmektedir.



https://www.geeksforgeeks.org/chroot-command-in-linux-with-examples/?ref=rp

Linux VServer

- «jail» sistemi daha da geliştirilerek ve işletim seviyesi sanallaştırma özellikleri çekirdeğe eklenerek Linux VServer projesi geliştirilmiştir.
- Proje, sistem kaynaklarını çekirdek seviyesinde paylaştırma düşüncesine dayanmaktadır. Örneğin, dosya sistemi (jail), CPU zamanı, ağ adresleri ve bellek, prosesler arasında bölünmektedir.
- Güvenlik bağlamı (security context) adı verilen bu sınırlı çalışma ortamı dahilinde, işletim sistemi seviyesinde proseslerin sanal makine gibi görülmesi sağlanmaktadır. Debian ve Fedora gibi işletim sistemleri, VServer üzerinde herhangi bir değişiklik yapılmadan çalışabilmektedir.
- Linux VServer projesini ilk geliştiren kişi Jacques Gelinass'tır. Daha sonra 2003 yılında Herbert Pötzl projeyi devralmış ve Enrico Scholz'un ardından 2005'te Benedikt Böhm projeye önemli katkılarda bulunmuştur. Proje daha sonraki dönemde yoğun bir geliştirme sürecine girse de son stabil sürümü, Linux çekirdeği v2.4 için 2008'de çıkmıştır.

http://linux-vserver.org/Overview http://linux-vserver.org/Paper

Linux VServer

- Sistem, Linux çekirdeğine yapılan yamalarla kurulmaktadır. En son sürüm,2019'da yapılan Linux çekirdeğinin v4.9.159 sürümüne olan yamadır. Ancak stabil değildir.
- Linux VServer'ın en önemli faydası, sanal makinelerin konaktaki sistem çağrılarına doğrudan erişmesi ve arada, hipervizör veya emülatör gibi zaman kaybı yaratacak sistemlerin bulunmamasıdır.
- Linux VServer'ın en büyük dezavantajı, ana Linux çekirdek geliştirme sürecinden (mainstream) farklı bir geliştirme süreci olması ve ana Linux çekirdeğine yama yapılarak uygulanmasıdır. Ayrıca, ağ sistemi sadece izole edilmekte ve ağ sanallaştırması bulunmamaktadır.

http://linux-vserver.org/Overview http://linux-vserver.org/Paper

Diğer Sanallaştırma Projeleri

- **Plex86**: Linux üzerinde hafif sanal makine çalıştırmayı amaçlayan bir projedir. 2003'te plex86.sourceforge.net'te geliştirilmeye başlandı. VMware mantığıyla Linux'un çalıştırılması amaçlanmıştır. Ancak, konak üzerinde çalışan sanal makinede Linux çekirdeğe yama yapılması gerekmektedir.
- **Bochs**: Emülatör programıdır. C++'ta yazılmıştır ve 32 bitlik x86 makineler için makine kodlarını emüle etmek için geliştirilmiştir. Kaynak kodları ve proje sayfası bosch.sourceforge.io sitesinde bulunmaktadır. 2021'de bir sürümü çıkmıştır.
- **User-Mode Linux (UML)**: Linux'un kullanıcı modunda çalıştırılmasını hedefleyen para-sanallaştırma sistemidir. UML, v2.6'dan itibaren Linux çekirdeğinde yer almaktadır. UML'de çekirdek sanallaştırılmamış sadece konuk makineler sanallaştırılmıştır.

https://www.kernel.org/doc/html/latest/virt/uml/user_mode_linux_howto_v2.html

OpenVZ

- Rusya'da Moskova Fizik ve Teknoloji Enstitüsünde(MIPT) geliştirilen işletim sistemi sanallaştırma temelli bir sistemdir.
- Enstitüde, 1999 yılında, Linux'un 2.2 sürümünde herhangi bir proses için sanal bir ortamın nasıl kurulabileceği konusunda çalışmalar yapılmaktaydı.
- 2000 yılında, MIPT'de ticari bir sistem olarak hazırlanan Virtuozzo projesine başlandı. Geliştirilen yazılım 2002 yılında Amerika'da satışa sunuldu ve uluslararası piyasada satılan bir ürün oldu.
- 2005 yılında, Virtuozzo sisteminin kaynak kodu, OpenVZ adıyla GPL ile kamuya açıldı.
- 2006 yılında, Fedora, Debian, RHEL, vb sistemlere de port edildi.
- 2009 yılında firma isim değişikliğine uğradı ve Parallels adını aldı. Şirket, bu dönemde, Linux çekirdeğine en fazla katkıda bulunan ilk 10 firma arasına girdi.

https://openvz.org/

`cpusets` : proseslerin izolasyonu

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- İşletim sistemi temelli sanallaştırma teknolojisinin temelinde, bir prosesin izole edilmesi gereksinimi bulunur. Sanal makine olarak çalıştırılacak proses, bir şekilde sınırlandırılmalı ve konak sistemden izole edilmelidir.
- Bu talep işletim sistemi çekirdeklerinin bütün prosesleri mümkün olduğunca daha fazla çalıştırma hedefine uymamaktadır. Bu nedenle proseslerin CPU ve bellek kullanımını sınırlandırma yöntemleri daha sonraları çekirdeklere eklenmiştir.
- Konteyner kavramının çıkış noktasının «**`cpusets`**» adı verilen Linux v2.6'a eklenen bir özellik olduğu öne sürülmektedir. «**`cpusets`**», bir grup CPU'nun ve bellek bölümünün bir veya daha fazla prosese atanabilmesini sağlayan bir özelliktir.
- 2004 yılında arasında Silicon Graphics şirketinden bir grup mühendis tarafından önerilmiş ve daha sonra bu özellik Linux çekirdeğine eklenmiştir.
- «**`cpusets`**» sayesinde, herhangi bir prosesin hangi CPU'yu kullanacağı ve ne kadar belleği kullanabileceği sınırlandırılabilmektedir.

https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v1/cpusets.html

cgroups (kontrol grupları)

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Konteyner mimarisindeki en önemli kavramlardan biri «**cgroups**»tur. Özellikle «**jail**» kavramından sonraki en önemli aşamadır.
- 2006 yılında, Paul Menage ve Rohit Seth (Google) Linux çekirdeğinde olan «**cpusets**» kavramını kullanarak, çekirdeğe yük getirmeyecek «**jenerik proses konteyneri**» sistemini geliştirmiştir. Daha sonra sistemin adı, daha geniş bir anlamı olabileceği düşünülerek «**kontrol grupları**» veya kısaca «**cgroups**» olarak değiştirilmiştir.
- Kısacası, konteyner terimini ilk ortaya koyanlar, Google mühendisleridir (2006). 2008'de Linux v2.6.24'da yayınlanmıştır.
  - https://lwn.net/Articles/199643/
- «**cgroups**», proseslerin gruplanabilmesini ve her grubun, belirlenmiş bir miktar bellek, CPU ve disk G/Ç kapasitesini sistemden alabilmesini sağlamaktadır. Kısacası, tek bir konteynerin, bütün sistemin kaynaklarını kullanabilmesi engellenmektedir.
- Örnek: Linux bir makine eğer Üniversite'de kullanılıyorsa, öğretim üyelerine kaynakların %60'ı, öğrencilere %40'ı verilebilir.

https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v1/cgroups.html

`cgroups` (kontrol grupları)

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- 2008'de bellek sistemiyle ve aygıtlarla ilgili kontrol grupları oluşturuldu. Bu özellik 2014'te **`cgroups`** v2 sürümü Linux v4'de yer almaktadır.
- **`cgroups`**, çekirdek içine konulan sayaçlarla (bellek, CPU, vb) proseslere sağlanan kaynakları sınırlandırabilmektedir.
  - bir grup proses tarafından kullanılan bellek miktarının belirlenen limiti geçmesi engellenebilmektedir.
  - Diğer yandan, önceliklendirmeyle bir grup prosesin CPU kullanımında daha fazla pay alması sağlanabilmektedir.
  - Bir kontrol grubuna tabi olan proseslerin kaynakları ne kadar kullandıkları istatistik olarak kaydedilmekte ve belirlenen sınırları geçme durumu takip edilmektedir.
  - Proses grupları (içindeki proseslerle beraber) istendiği zaman duraklatılabilmekte ve gerekirse bütün grup yeniden başlatılabilmektedir.
- Sistem çağrıları, bir çok kontrol grubu işleminin yapılabilmesi için çekirdek içinde tanımlanmıştır.

https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v1/cgroups.html

İsimalanı (**`namespace`**) kavramı

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Konteyner mimarisindeki diğer önemli kavram «**`namespace`**»tir ve bir prosesin veya bir grup prosesin izolasyonunu sağlamak için kullanılır.
- «**`cgroups`**» kavramı, kaynakların sınırlandırılması anlamında önemli bir fayda sağlamakla birlikte prosesi izole edememektedir. Normal şartlarda, kaynakları sınırlandırılsa da proses, diğer prosesleri ve tüm sistem kaynaklarını görebilir.
- **SORU**: «Eğer sistem kaynakları «**`namespace`**»ler altında bölünürse ve bu isim-alanlarındaki prosesler sadece ait oldukları isim-alanındaki kaynakları görebilirse, proseslerin izolasyonunu bir adım daha ileriye götürebilir miyiz?"
- Diğer bir deyişle, X isim-alanındaki bir proses, X isim-alanındaki kaynakları görebilir ve erişebilirken, Y isim-alanındaki kaynaklara ulaşamaz ve göremez.

İsimalanı (**`namespace`**) kavramı

- Bu düşünce Konteyner kavramının ardındaki mantığı da göstermektedir. «**`cgroups`**» konteyner içindeki prosesleri gruplayarak kaynakları sınırlandırabilmekte ve «**`namespace`**» kavramı da sadece kendi isim alanındaki kaynakların görülmesi sağlanmaktadır.
- «**`namespace`**» 2007 sonunda yayınlanan Linux v2.6.23'te yer aldı ve çekirdekteki bazı fonksiyonların çalışması (**`clone`** ve **`unshare`**) değiştirildi ve kullanıcı isim alanı kavramı kullanılmaya başlandı. Belirli kullanıcıların, konteyner içinde **`root`** yetkisine sahip olması sağlandı ve bu kullanıcıların konteyner dışında **`root`** olmaları engellendi.
- İsim alanı konusu, işletim sisteminin bir çok parçasını ilgilendirdiği için çalışmalar aşamalarla ve uzun bir zaman çerçevesinde gerçekleştirildi.
- 2007'de ağ sistemlerinde isim alanı oluşturuldu. Artık prosesler, sanallaştırılmış bir ağ yığıtına sahip olmakta ve kendi yönlendirme tabloları, firewall kuralları ve ağ ara yüzleri konak makineden farklı olabilmektedir.

İsimalanı (**`namespace`**) kavramı

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- 2007'de proseslerin PID'leri isim alanına eklendi. Prosesler, aynı isim alanındaki prosesleri **sanal** PID değerleriyle görebilmeye başladı.
  - Örneğin, konteynerde çalışan **`init`** veya **`systemd`**'nin PID değerleri, bu proseslere 1 olarak gösterilebildi.
- Ancak isim alanı kavramının eksiksiz hale gelmesi 2012'de çıkarılan Linux'un v3.8 sürümünde gerçekleşmiştir.

Solaris Zones/Containers

- 2004 başında Solaris 10'da sunulan, X86 ve SPARC mikro işlemcili sistemler için işletim sistemi seviyesinde sanallaştırma uygulamasıdır.
- Solaris'in bu çözümü, bir işletim sistemi üzerinde birbirinden izole edilmiş sanal sunucular çalıştırarak maliyeti düşürmeyi hedeflemektedir.
- «Solaris Containers», proseslerin kaynak kontrolünü ve izolasyonu sağlayan «zones» adı verilen bir sisteme dayanmaktadır. Solaris Containers'in, aslında, zones uygulamasına yönetim ara yüzünün eklenmesiyle oluşturulduğu bilinmektedir.
- «Zone» bir işletim sistemi üzerinde yer alan birbirinden izole şekilde çalıştırılan sanal sunuculardır. Her zone'un bir adı, sanal veya fiziksel ağ arayüzü ve depolama alanı bulunur. Her birinin etrafında, bir zone'daki proseslerin diğer zone'dakilere ulaşmasını engelleyen bir güvenlik bariyeri vardır.
- Solaris zones/containers, 2010 yılında Solaris ürünün artık devam edilmemesi nedeniyle kullanılamamaktadır.

https://docs.oracle.com/cd/E36784_01/html/E36848/zones.intro-2.html#scrolltoc
https://asiyeyigit.com/solaris-containers/

LXC (LinuX Containers)

- LXC gerçek anlamda konteyner kavramının uygulamasını sağlamıştır.
- LXC'de her bir konteyner **`/var/lib/lxc`** dizini altında bir dizin içinde bulunur.
- «Küçük» bir yapılandırma dosyası ve kök dosya sistemi bulunur.
- Konfigürasyonun yapılması ve konteynerin çalıştırılması, uygulama geliştiricileri için zorken, sistem yöneticileri için kolay olabilir.
- Buradaki en büyük problem, konteynerlerin taşınabilirliği konusudur. Konfigürasyon dosyasının üzerinde çalışan sisteme özel bileşenleri vardır ve yeni sisteme özgü ayrıntılarla ilgili yapılandırma dosyalarında değişiklik yapılması gereklidir. Parametreler karmaşıktır.
- Örneğin, Debian'da LXC'de oluşturulan bir konteyneri, Fedora dağıtımında çalıştırmak oldukça zor olabilir.

https://linuxcontainers.org/lxc/manpages/man5/lxc.container.conf.5.html

LXC

- LXC’nin kullandığı özellikler şunlardır:
  - Linux **`namespace`**
  - Linux **`cgroups`**
  - SELinux (Secure Linux) profilleri
  - Linux çekirdeğinde v2.6’dan beri sunulan «**`seccomp`**» politikaları (bir prosesin sadece **`exit()`**, **`sigreturn()`**, daha önce açılmış dosyalar için **`read()`** ve **`write()`** komutları dışında, sistem çağrısı yapamaması özelliği)
  - **`chroot`** («**`jailed`**» kök dizini oluşturmak)
  - Çeşitli Linux çekirdek özellikleri

https://kubernetes.io/docs/tutorials/security/seccomp/
https://linuxcontainers.org/lxc/introduction/

`systemd-nspawn`

- **`chroot`**'a benzer bir çalışma sistemi vardır.
- Ancak, daha fazla opsiyon sağlar ve tüm özellikleri ile Linux temelli bir işletim sistemini konteyner içinde çalıştırabilir.
- **`/sys/`**, **`/proc/sys/`** ve **`/sys/fs/selinux`** gibi dizinlere erişimleri «**`read-only`**» hale getirebilir.
- Genellikle zayıf bir konteyner çalışma ortamı olarak değerlendirilir.

https://wiki.archlinux.org/title/systemd-nspawn

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

Konteyner Teknolojileri ve Docker

Konteyner teknolojileri ve Docker

- Linux üzerinde izole şekilde çalışan prosesler, 1990'lı yılların ikinci yarısından itibaren üzerinde çalışılan ve farklı çözüm önerileri geliştirilen bir kavramdır.
- 1998'de VMware'in x86 mimarisinde sanallaştırmayı uygulanabilir hale getirilmesi, Linux'un sanallaştırılması çabalarını hızlandırdı. Bir çok yazılım geliştirici, grup ve şirket bu konuda çalışmalara başladı.
- ★Bir prosesin veya bir grup prosesin, Linux üzerinde kullanabildiği kaynakları çekirdek seviyesinde sınırlandırmak, Linux'un kullanılabilirliğini artıracak önemli bir adımdı ve bu hedefle yola çıkıldı.
- 2006'ya kadar, Linux çekirdeğine ek olarak yapılan yamalarla çalışmalar gerçekleştirildi ve «upstream» Linux çekirdeğine ekleme yapılmadı.
- 2004'te «**`cpusets`**», 2006'da «**`cgroups`**» ve ardından 2008'de «**`namespace`**» özellikleri «**`upstream`**» Linux çekirdeğine eklendikçe, işletim sistemi seviyesinde sanallaştırma fikri olgunlaştı.

Konteyner teknolojileri ve Docker

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Bu özellikleri kullanan LXC'nin 2008'de sunulması, Linux üzerinde çekirdek yaması gerektirmeden çalışan diğer bir önemli aşamayı temsil eder. O dönemde, «tek» bir prosesin sistemden yalıtılması sadece LXC tarafından sağlanmaktaydı. O dönemde, LXC en başarılı konteyner uygulaması olarak kabul edilmekteydi.
- LXC stabil ve güvenilir bir uygulama olmasına rağmen, kurulumunun zorluğu, taşınabilirlik sorunları kullanımını sınırlandırmıştır.
- LXC'nin kullanımının kolaylaştırılması ve bulut yapılarında da kullanımı için, **`cgroups`** ve **`namespace`** özelliklerini kullanan Warden yazılımı 2011'de Cloud Foundry tarafından piyasaya sunulmuştur. Ancak, Warden yazılımının önemli sınırlılıkları yaygınlaşmasını engelledi.
- 2013'te ise Docker, konteyner uygulamalarında katmanlı imaj kavramını getirmiş ve Docker hub üzerinden sunduğu imajlarla, kullanıcıların kendi konteynerlerini daha kolay oluşturmasını sağlamıştır. Ağ arayüzü konusunda sunduğu geniş özellikler de eklenince Docker en önemli konteyner teknolojisi olmuştur.

https://www.altoros.com/blog/cloud-foundry-containers-warden-docker-and-garden/

Docker

- İşletim sistemi üzerinde uygulamaların sanallaştırılmasını ve konteynerlerin çalıştırılmasını sağlayan yazılımdır. Docker, Linux işletim sistemlerinin sanallaştırma ve izolasyon özelliklerini kullanır.
- 2013'ten beri Docker şirketi tarafından geliştirilen yazılım artık bir endüstri standardı haline gelmiştir.
- «Docker» kelimesinin anlamı liman işçisidir ve konteyner kavramıyla ilişkilidir.
- Açık kaynak kodlu bir sistemdir.
- Docker, ilk sürümlerinde LXC altyapısını kullanmıştır ve daha sonra kendi çalıştırma ortamını geliştirmiştir.

Docker

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Docker, geliştiriciler için herhangi bir amaçla oluşturulacak konteyneri oluşturacak araçlara sahiptir.
- Daha önce geliştirilen konteynerler temel alınarak ve kullanılarak, daha özelleştirilmiş konteynerler geliştirilebilmektedir.
- Docker tarafından sunulan «**`Docker Hub`**» web sitesi (**`https://hub.docker.com`**) konteyner uygulamaları için bir alış/veriş noktasıdır. Çeşitli yazılım geliştiricilerin geliştirdikleri konteynerler burada yayınlamaktadır. Bazıları ücretsiz olabildiği gibi bazıları destek sağlandığı için ücretlidir.
- Konteynerlerin otomatik olarak oluşturulması ve devreye alınması için bir API bulunmaktadır.

https://docs.docker.com/engine/faq/

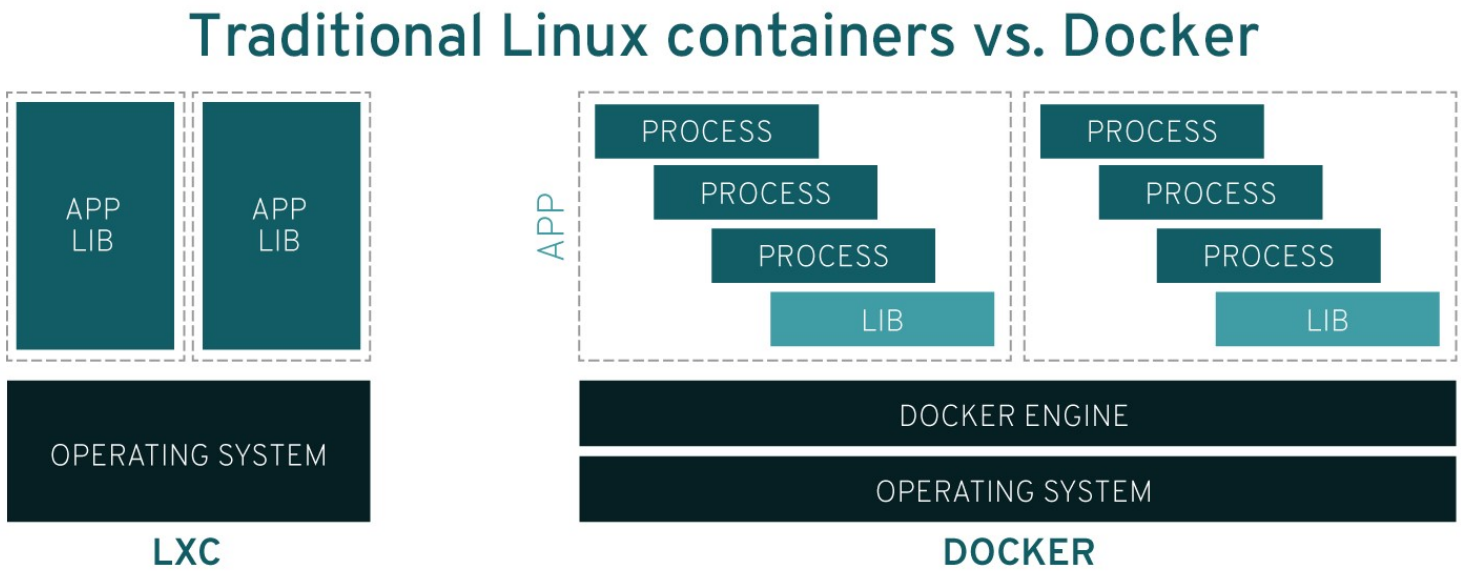
Docker ve LXC

- LXC, Linux çekirdeğinin konteyner uygulamalarını çalıştırmak için sağladığı olanakları (isim alanları ve kontrol grupları) ile izole bir ortam (sandbox) oluşturularak uygulamaların çalıştırılmasını temsil etmektedir.
- Docker, diğer yandan, bir uygulamanın, bütün bileşenleri ve gerek duyduğu diğer parçaların bir araya getirilmesi için bir format/yöntem sunmaktadır. Diğer bir deyişle, geliştirme ortamında hazırlanan Docker imajı «prod» ortamında da aynı şekilde çalıştırılabilir.
- LXC'de ise, LXC konfigürasyonu ile oluşturulan konteyner, sonuçta alttaki donanımın/sistemin özelliklerine (ağ, depolama, loglama) bağlıdır ve taşınabilirlik özellikleri eksiktir. Docker imajları ise, alttaki donanım ve işletim sistemi ne olursa olsun aynı şekilde çalıştırılır.
- Taşınabilirlik ve sağlanan araçlar sayesinde, Docker artık bir endüstri standardı haline gelmiştir.

https://docs.docker.com/engine/faq/

Docker ve LXC

- Docker ve geleneksel LXC arasındaki en önemli fark, Docker sisteminin birden fazla prosesi desteklemesi ama LXC içinde sadece tek bir uygulamanın çalıştırılabilmesidir.



https://www.redhat.com/en/topics/containers/what-is-docker

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

Konteynerler ve Docker

Konteyner nasıl oluşturulur?

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Konteyner kavramı, bir bütün olarak Linux çekirdeğinde yer almaz ve sadece konteynerlerin oluşturulabilmesi için çeşitli özellikleri sağlar.
- Docker, konteynerlerin nasıl oluşturulacağından, nasıl çalıştırılacağına kadar olan süreci yönetmek için kullanılan bir yazılımdır.
- Docker, konteynerlerin oluşturulmasında Linux çekirdeğinin üç özelliğini kullanır:
  - **`jail`**: Bir prosesin farklı bir kök dizini görmesinin sağlanmasıdır.
  - **`cgroups`** (kontrol grupları): Bir prosesin veya proses gruplarının işletim sistemindeki kaynakların (CPU zamanı, bellek miktarı, vb) birbirlerinden bağımsız şekilde ne kadarını kullandığını ve ne kadar kullanacağını sınırlandırmak için kullanılan bir özelliktir. Sınırlar aşıldığında, proseslerin dondurulması, vb aksiyonlar alınabilir.
  - **`namespaces`** (isim alanları): her prosesin, işletim sistemini istendiği şekilde görmesinin sağlanmasıdır. Prosesin gördüğü proses ID'leri, ağ parametreleri, vb parametreler değişebilir. Örneğin, istenen proses, kendisiyle birlikte çalışan prosesleri, farklı PID sıralamasıyla görebilir. Her proses, PID'si 1 olan bir init prosesi görmektedir. Kendi kök dizinini veya **`tmp`** dizinini görebilir. Sadece kendisinin görmesine izin verilen kullanıcıları görür.

https://www.youtube.com/watch?v=sK5i-N34im8

Konteyner nasıl oluşturulur?

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Özet olarak, **`jail`** dosya sisteminin konteyner tarafından görüldüğü alanı belirlemekte, **`cgroup`** kaynak tahsisini sınırlandırmakta ve **`namespace`** de konteynerin sistemde neleri görebileceğini belirlemektedir.
- Kısacası, bir konteyner, içinde yer alan bir uygulamanın veya uygulamaların, Linux sisteminden, Linux çekirdek özellikleri olan «**`cpusets`**», «**`namespace`**», «**`cgroups`**» ve «**`chroot`**» gibi öğelerinin kullanılarak soyutlanmasıyla oluşturulmaktadır.
- Bir konteyner içindeki uygulamalar, çekirdek içindeki bir dizi «**`system call`**» kullanılarak izole bir şekilde çalıştırılır.

Konteynerler nasıl çalıştırılır?

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Konteynerlerin oluşturulması yanında, bunların nasıl etkin çalıştırıldığı önemlidir. Bu işten sorumlu olan konteyner çalıştırma yazılımıdır. Bir imaj dosyasını alır ve içindekileri, izole bir proses halinde çalıştırır.
- Her konteyner, «çalıştırma ortamı üzerinden» alttaki işletim sistemi kaynaklarını kullanır. Linux konteynerler, Linux işletim sistemini kullanırlar. Benzer şekilde Windows konteynerler de üzerinde çalıştırıldıkları Windows'un servislerini çağırırlar.
  - Windows konteynerler sadece Windows işletim sisteminde çalışırlar.
  - Linux konteynerler, herhangi bir Linux distro üzerinde çalışabilir. Windows üzerinde WSL2 üzerinde çalışan Ubuntu Linux üzerinde çalışabilirler.
- Çalıştırma ortamlarının (örneğin, Docker, **`containerd`**, vb) amacı, belirli bir dosyada veya dizinde bulunan konteyner içindeki öğeleri, belirli bir düzende çalıştırmaktır.
- Farklı konteyner çalıştırma ortamları (**`container runtime`**) bulunmaktadır.
- Ancak çalıştırma ortamı (**`runtime`**), konteyner teknolojisinin sadece bir yönüdür. Konteynerlerin bulut ortamında çalıştırılması için, **`runtime`** ortamının üstünde bir seviye bulunmaktadır. Docker Swarm, Kubernetes gibi sistemler bulut üzerinde konteynerlerin çalışmasını sağlarken, onların devreye alınmasını, izlenmesini ve diğer bir sunucuya aktarılması gibi işlevler görürler.

https://learn.microsoft.com/en-us/windows/wsl/tutorials/wsl-containers

Docker nasıl çalışır?

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Docker, Linux çekirdeği içindeki isim alanları (**`namespaces`**) ve kontrol grupları (**`cgroups`**) gibi özellikleri kullanarak, konteynerleri (ve içindeki prosesleri) birbirinden bağımsız olarak çalıştırabilmektedir.
- Docker, konteynerlerin içindeki uygulamaları, birbirinden izole şekilde çalıştırır ve bu şekilde güvenliği sağlar.
- Docker, konteyner içinde yer alan bütün prosesleri otomatik olarak çalıştırır.
- Sanal ağ (vlan) oluşturarak, konteynerleri izole ağ ortamlarında çalıştırabilir ve İnternet'e çıkışlarını ağ köprüleri üzerinden sağlayabilir.
- Docker motoru, disk alanlarını (volume) konteynerlere bağlayabilir.

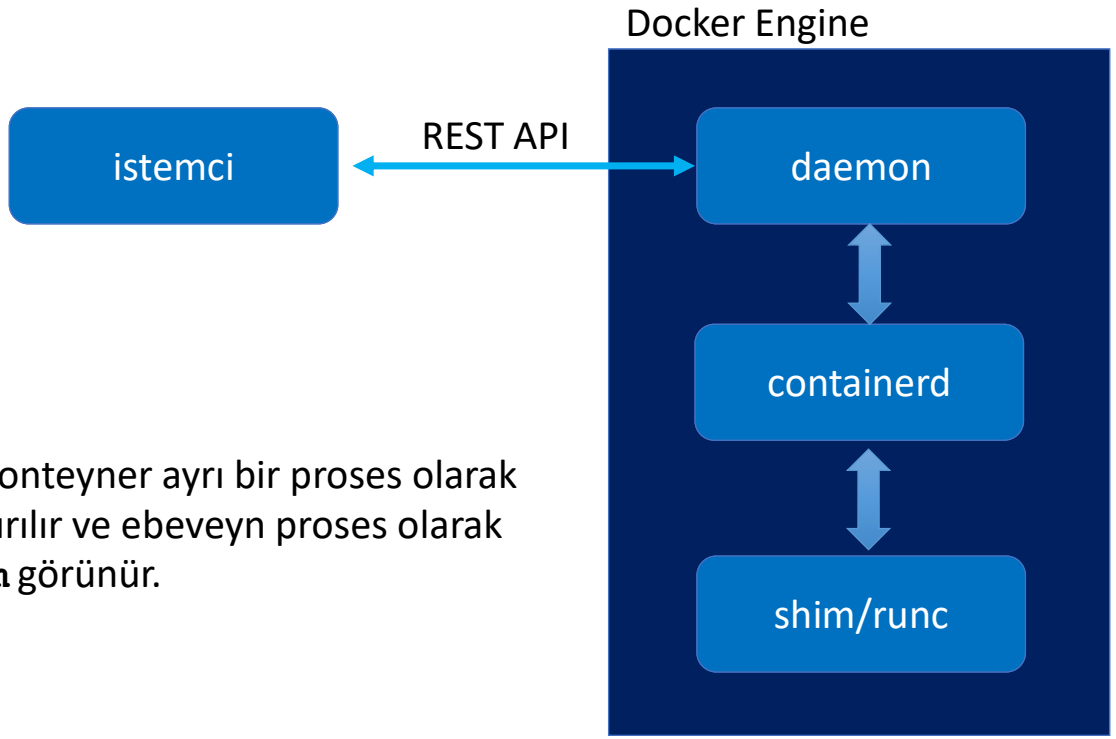
Docker

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Docker, istemci-sunucu mimarisi üzerinde çalışmaktadır.
- İstemci program (**`docker`** programı) Docker daemon'a (**`dockerd`**) bağlanarak komut verir.
- İstemci istenirse farklı sunuculardaki Docker daemon'lara ulaşabilir.
- Docker daemon konteynerler üzerinde şu işlemleri yapar
  - Oluşturma
  - Çalıştırma
  - Dağıtım
- İstemci ve daemon arasında haberleşmede «UNIX soketleri» ve Ağ ara yüzü kullanılır. İletişimde REST API kullanılmaktadır.

Docker Mimari

- «Docker» beş ana parçadan oluşmaktadır:
  - Docker istemci (docker komut satırı programı)
  - Docker daemon
  - containerd
  - shim
  - runc



Her konteyner ayrı bir proses olarak çalıştırılır ve ebeveyn proses olarak **`shim`** görünür.

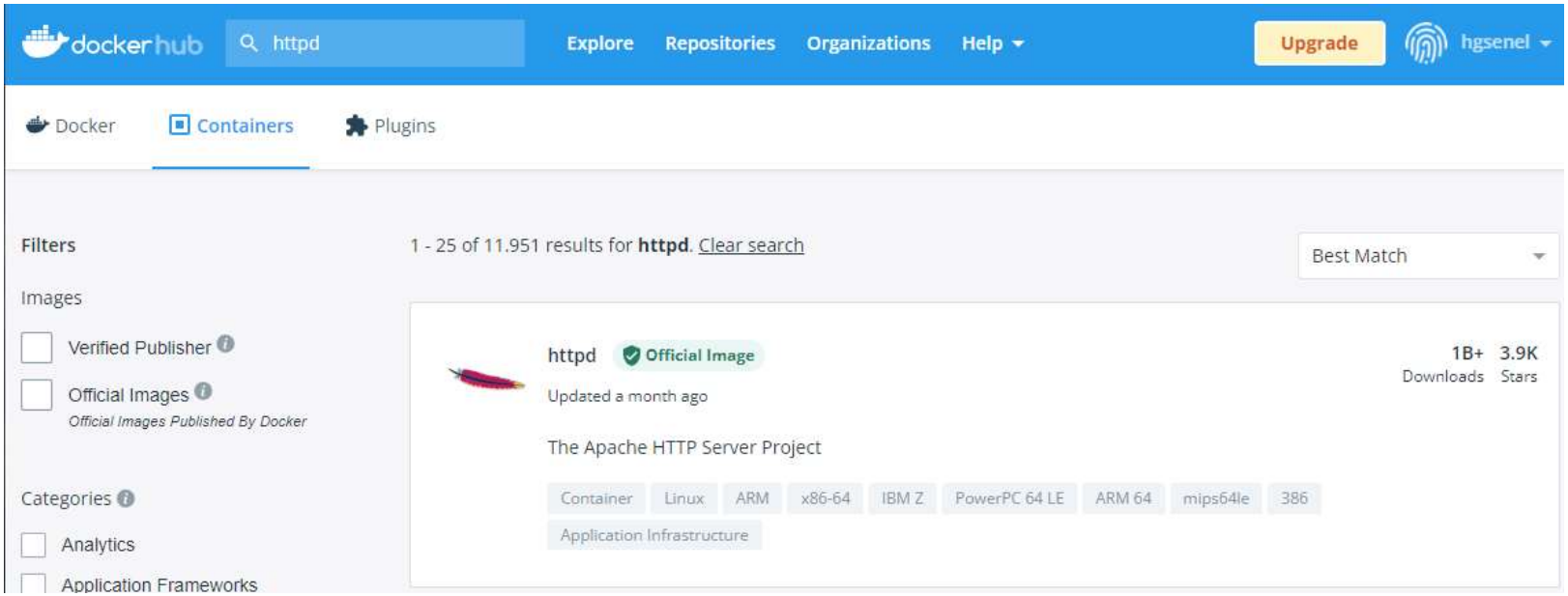
Docker Daemon (**`dockerd`**)

GANTEK ACADEMY
40+ YILLIK TECRÜBE İLE

- Docker API taleplerini dinler ve istemciden «**`docker`**» gelen komutlara göre Docker nesnelerini yönetir:
  - imajlar (Images)
    - Sadece okunabilen ve içindeki komutlarla çalıştırılabilir konteyner haline getirilebilen bir şablondur.
    - OCI (Open Containers Initiative) adlı kuruluşun çabaları sayesinde, imajların yapısı ve konteynerlerin nasıl çalıştırılacağı konusu artık endüstri standardı haline gelmiştir.
  - Konteynerler (Containers)
    - Çalıştırılabilir yapılardır.
  - Ağlar (network)
  - Yığınlar (volume)
- «**`docker`**» komutu Docker daemon'la etkileşime giren ve komut gönderilmesini sağlayan istemci programıdır.

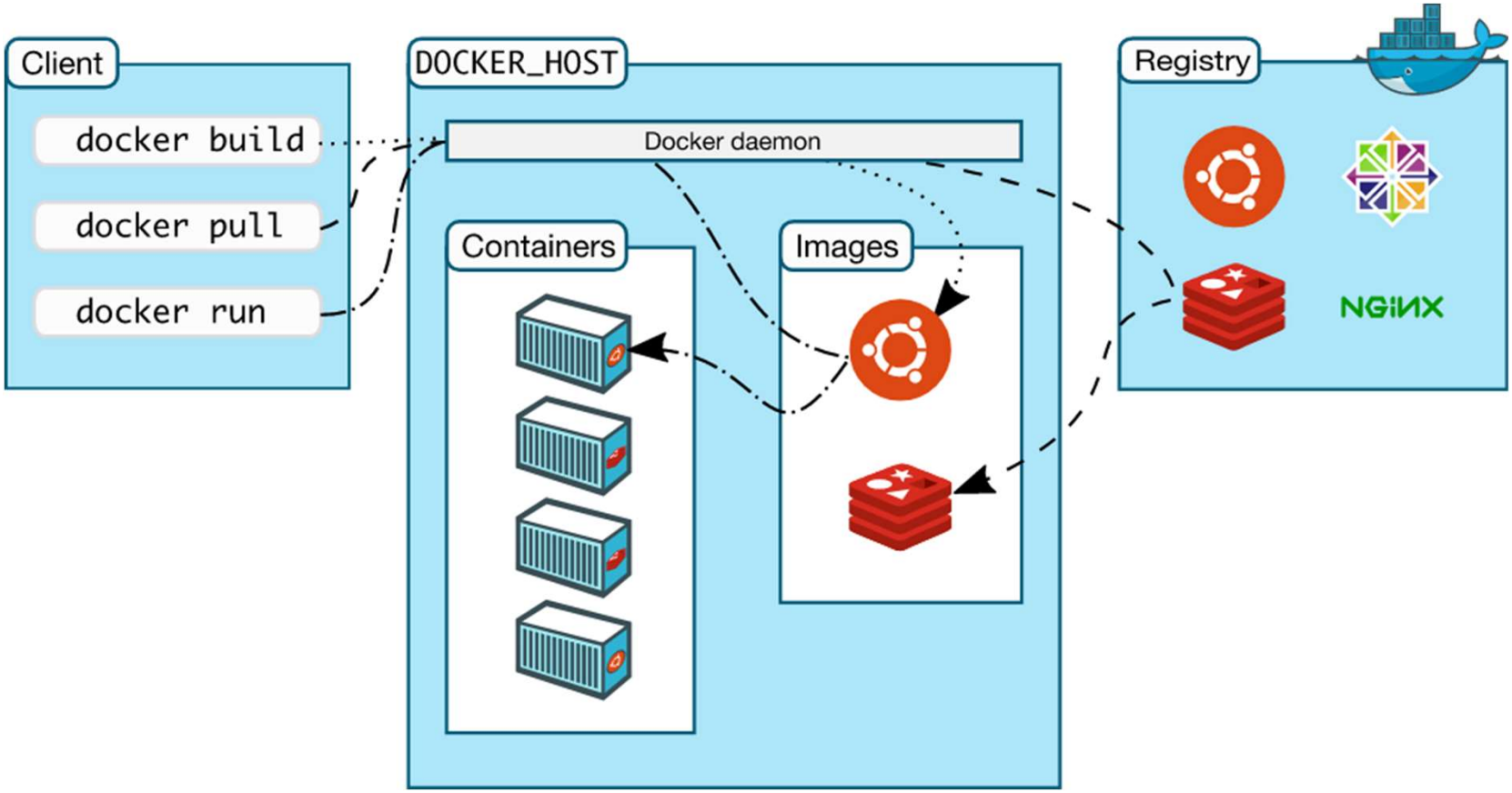
Docker Registry (hub)

- Docker hub (**`https://hub.docker.com`**), çeşitli amaçlarla oluşturulmuş bir çok imajın tutulduğu bir web sitesidir.
- Docker istemcisi, «**`docker pull`**» veya «**`docker run`**» komutları, Docker Hub'dan istenen imajı indirerek işlem yapabilir.



Docker Hub

- Aşağıdaki grafikte Docker sisteminin genel hatlarıyla nasıl çalıştığını görebiliriz:



https://docs.docker.com/get-started/overview/

Güvenlik Nasıl Sağlanıyor?

- Olgunlaşmış teknolojiler olarak ele alınan konteyner çalıştırma ortamlarının güvenilir sistemler oldukları düşünülmektedir.
- Ancak, Linux çekirdeğinin **`cgroups`** özelliğindeki bir açıktan faydalanarak, konak bilgisayarda (host) izinsiz ve yetkisiz kod çalıştırabileceği konusu 6 Mart 2022'de duyurulmuştur.
- Ancak AppArmor ve SELinux güvenlik özelliklerini kullanan çekirdeklerde bu sorun bulunmamaktadır.

https://securityaffairs.co/wordpress/128742/security/cve-2022-0492-linux-kernel-flaw.html