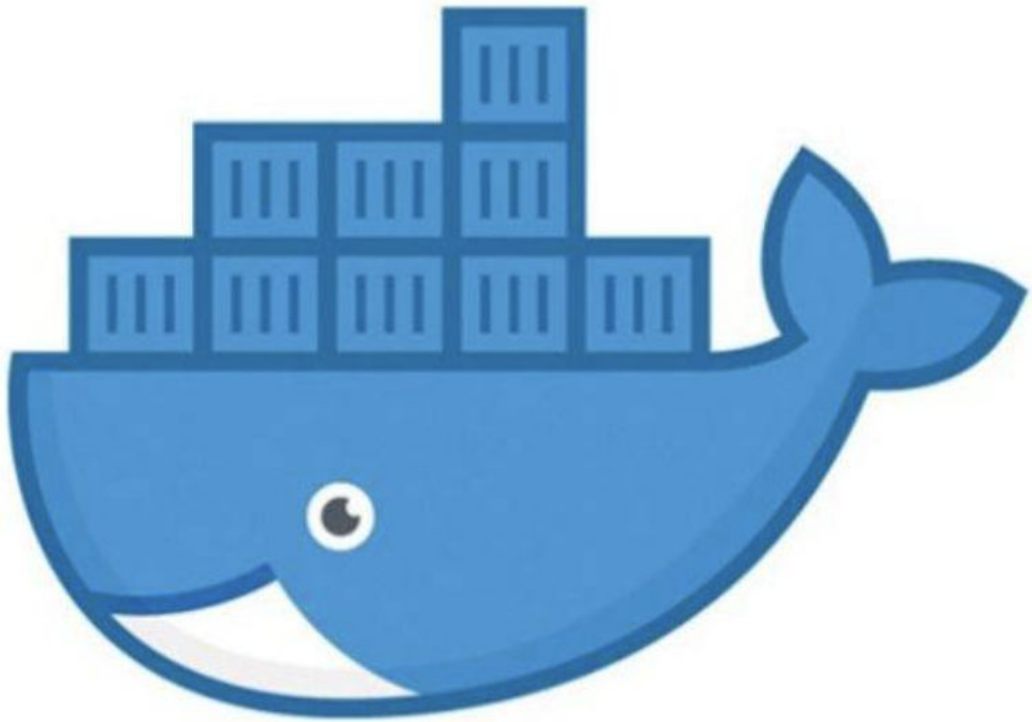
Docker

Hakan BAYRAKTAR

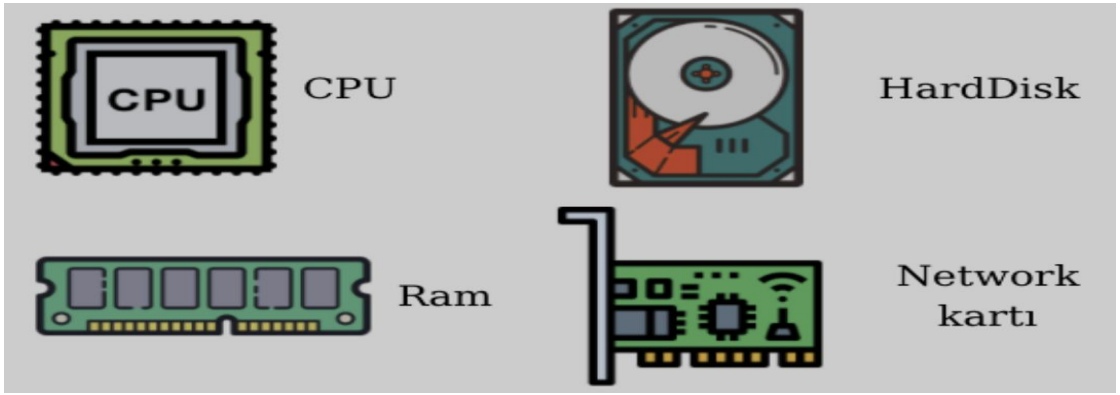
Docker

- Introduction to Docker
- Docker install
- Basic Docker Commands
- Docker Containers & Images
- Docker Volume
- Docker Networking
- Docker image Create
- Docker Compose
- Docker Labs

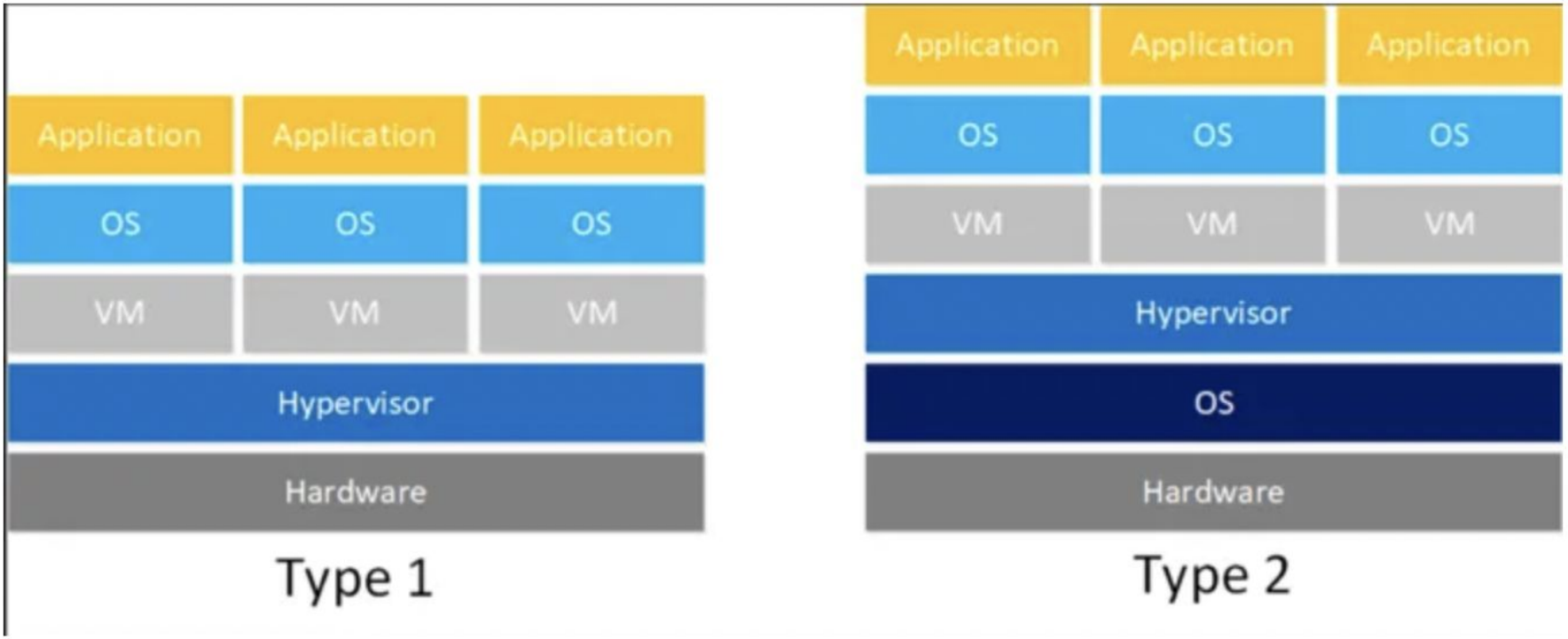
Introduction to Docker



Sanallaştırma Öncesi



Hipervizör tabanlı Sanallaştırma



Şekil 1. Hipervizör Tabanlı Tip 1 ve Tip 2 Sanallaştırma

Physical Machines

Webservers

AppServers

Databases

Libraries

Dependencies

Operating System

Hardware Infrastructure

Virtual Machines

Virtual Machine

Webservers AppServers Databases

Libs

Deps

Operating System

Virtual Machine

Webservers AppServers Databases

Libs

Deps

Operating System

Hypervisor

Hardware Infrastructure

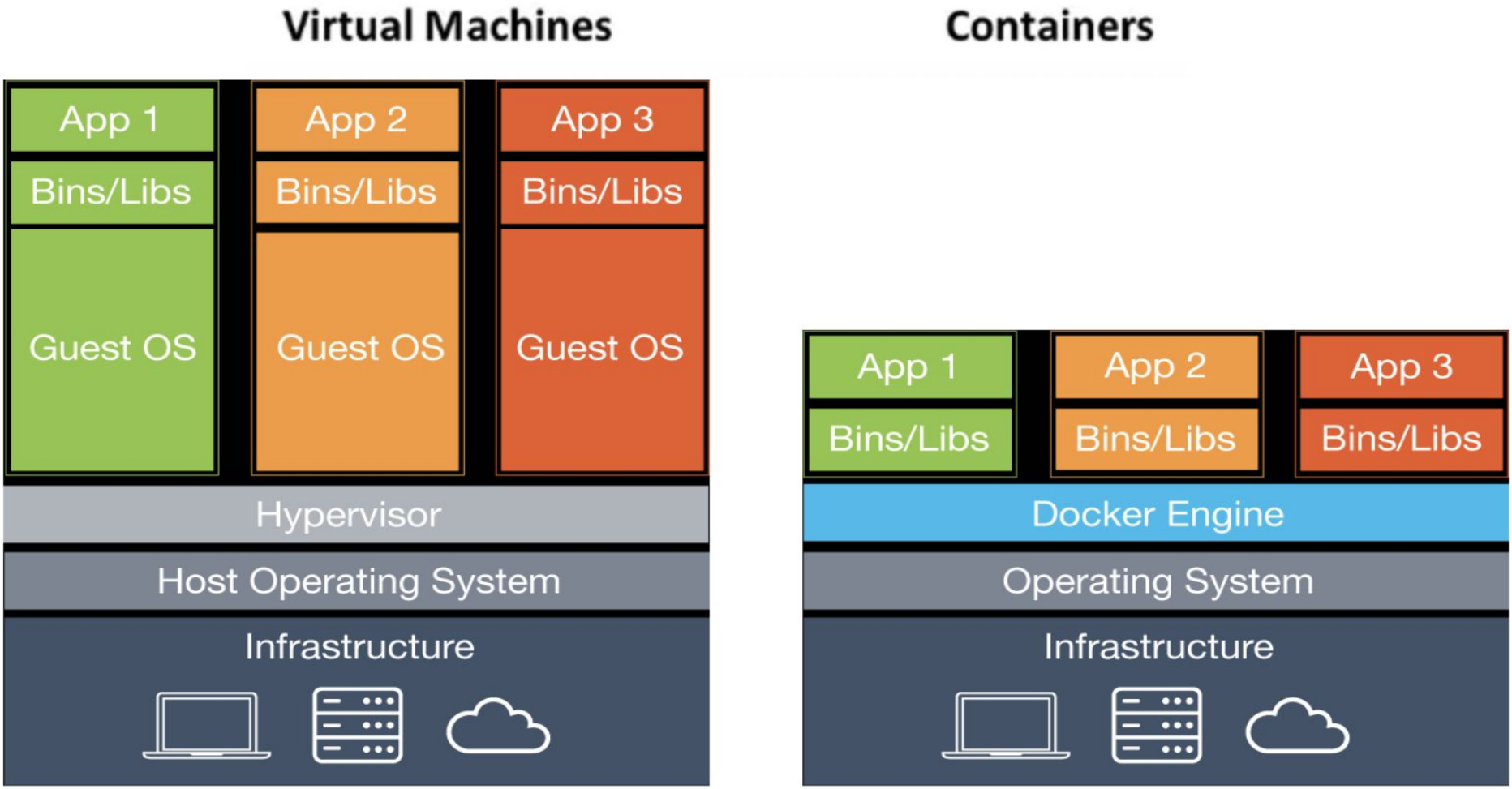
Sanallaştırma öncesi

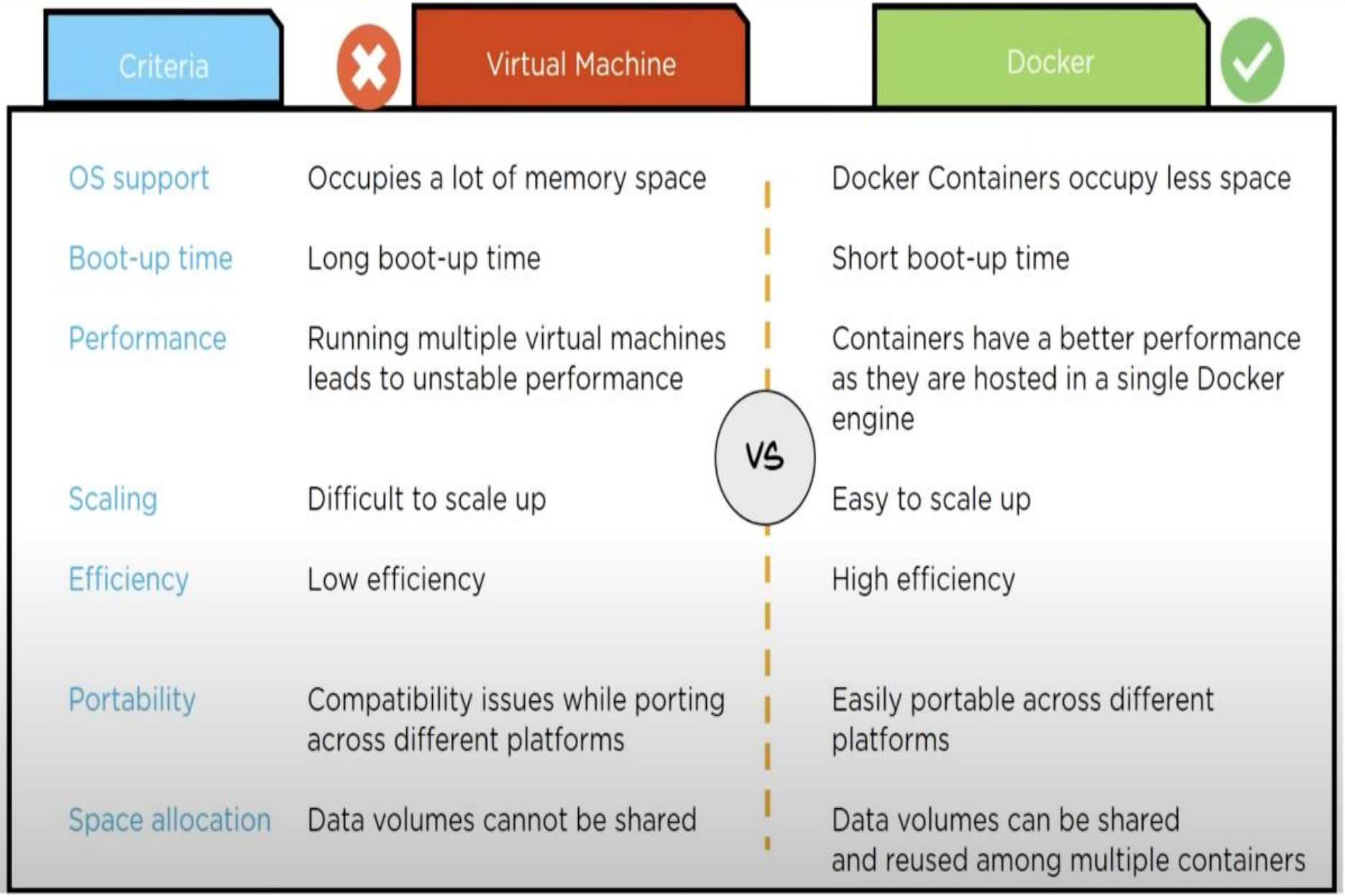
- Her bir rol için ayrı sunucu
- Sunucu üzerinde tek bir OS
- Network, depolama bağımlılık
- Sunucu kaynaklarında atıl durum
- Uygulama uyum problemleri

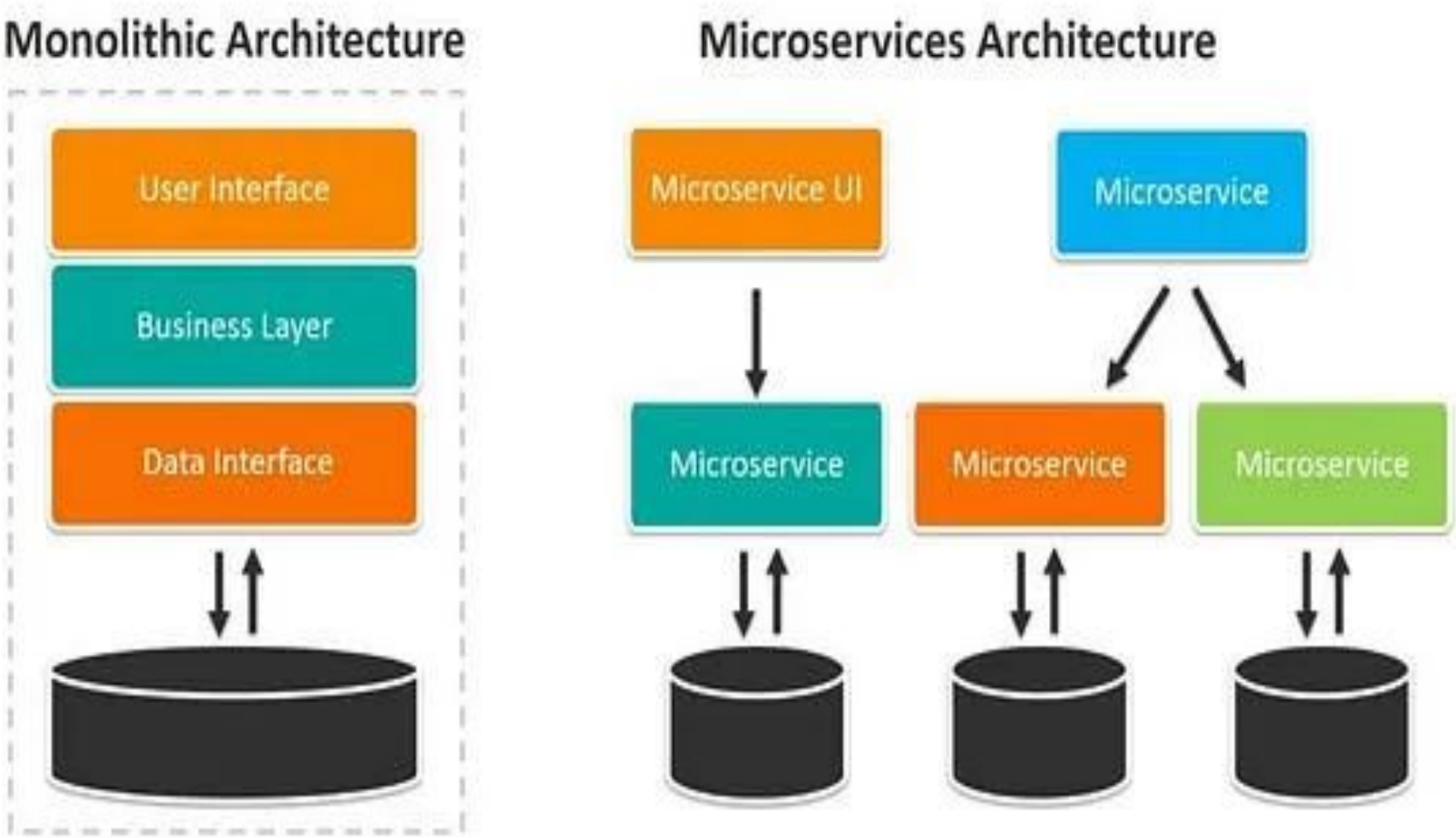
Sanallaştırma sonrası

- İşletim sistemi donanımdan soyutlanmış disk üzerinde bir dosya
- Sunucu üzerinde bir çok OS
- CPU disk Ram Network sanallaştırıldı

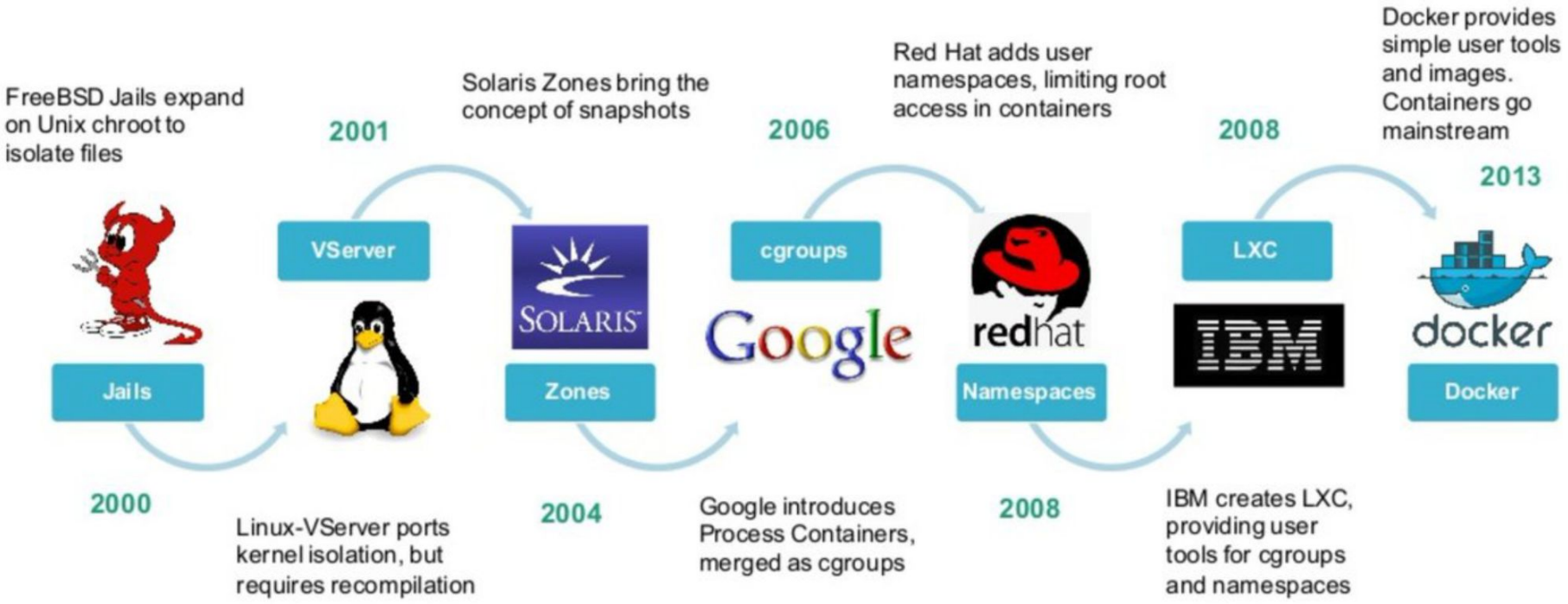
Docker ve Sanallaştırma




Criteria
Virtual Machine
Docker
OS support
Occupies a lot of memory space
Docker Containers occupy less space
Boot-up time
Long boot-up time
Short boot-up time
Performance
Running multiple virtual machines leads to unstable performance
Containers have a better performance as they are hosted in a single Docker engine
VS
Scaling
Difficult to scale up
Easy to scale up
Efficiency
Low efficiency
High efficiency
Portability
Compatibility issues while porting across different platforms
Easily portable across different platforms
Space allocation
Data volumes cannot be shared
Data volumes can be shared and reused among multiple containers


Monolithic Architecture
User Interface
Business Layer
Data Interface
Microservices Architecture
Microservice UI
Microservice
Microservice
Microservice
Microservice

Container History



What is Docker?

- Docker as a “Company”
- Docker as a “Product”
- Docker as a “Platform”
- Docker as a “CLI Tool”
- Docker as a “Computer Program”

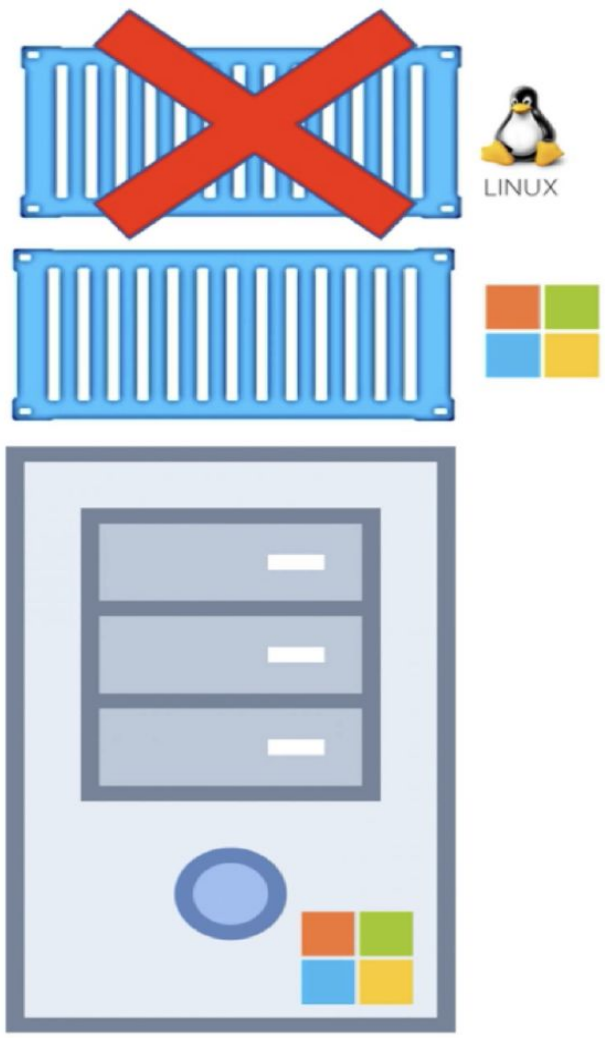
docker

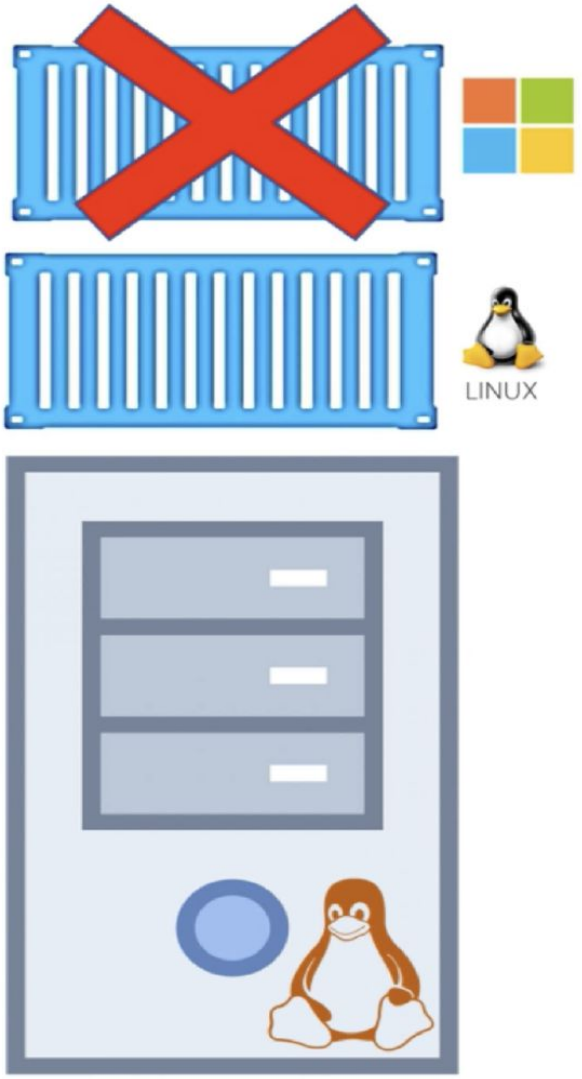
```
PS C:\Users\Ajeet_Raina> docker version
Client: Docker Engine - Community
 Version:           18.09.1
 API version:       1.39
 Go version:        go1.10.6
 Git commit:        4c52b90
 Built:             Wed Jan  9 19:34:26 2019
 OS/Arch:           windows/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          18.09.1
  API version:      1.39 (minimum version 1.12)
  Go version:       go1.10.6
  Git commit:       4c52b90
  Built:            Wed Jan  9 19:41:49 2019
  OS/Arch:          linux/amd64
```

docker Community Edition

docker Enterprise Edition

Docker Konteynerlerinin Platforma Bağımlılığı



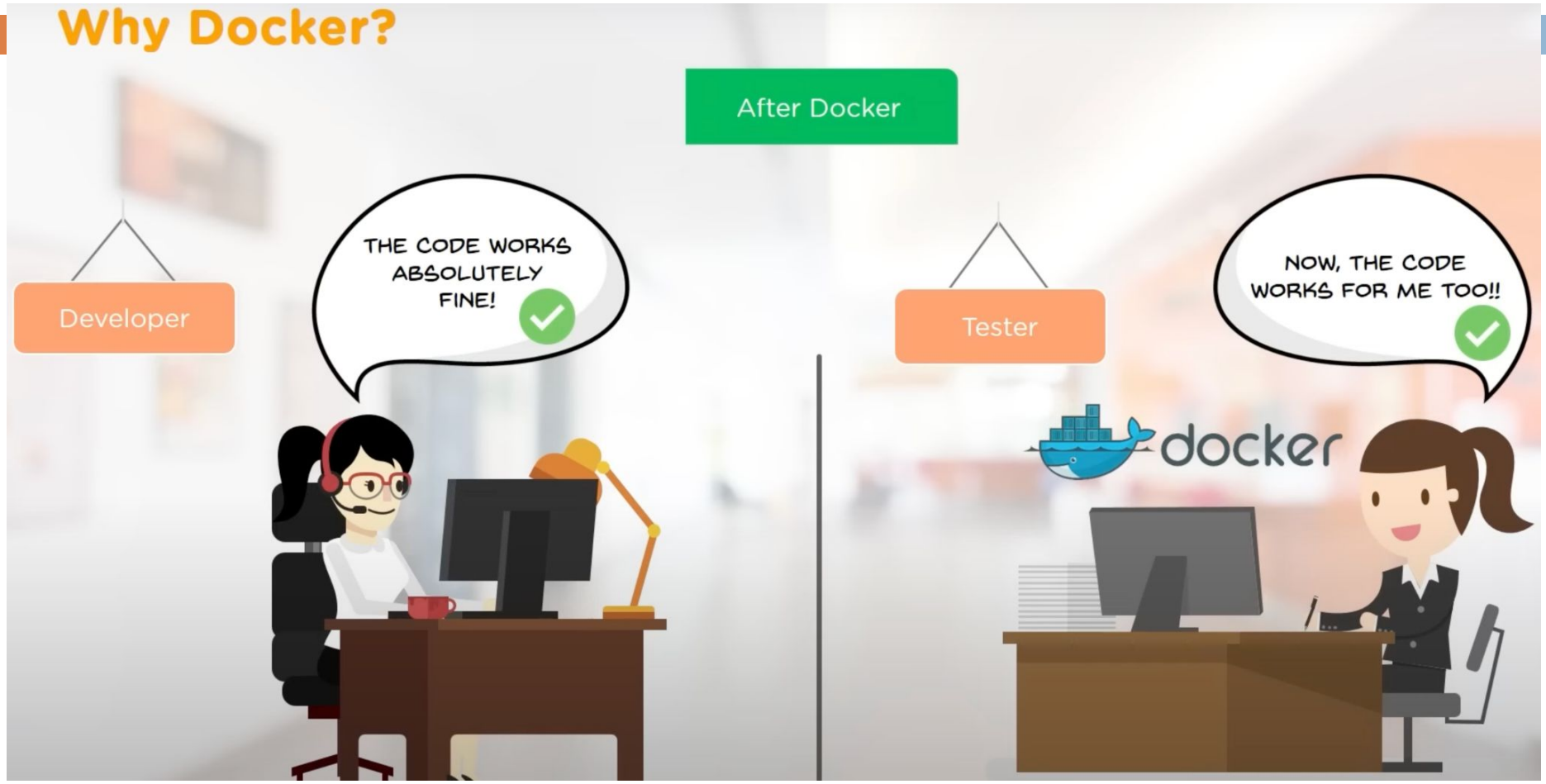


Why Docker?

The code doesn't work on the other system due to the difference in computer environments

So, what could be the solution to this?

Why Docker?


After Docker
Developer
THE CODE WORKS ABSOLUTELY FINE!
Tester
NOW, THE CODE WORKS FOR ME TOO!!
docker

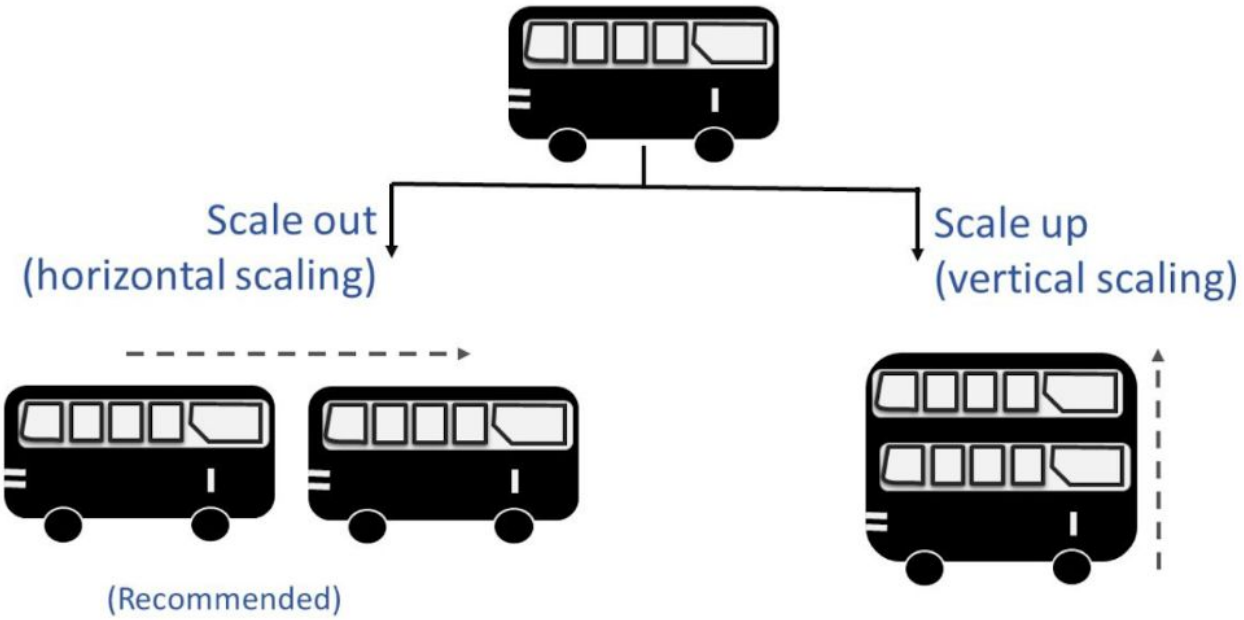
Why Docker?

- Increased Portability
  - Don't have to worry about environment

-

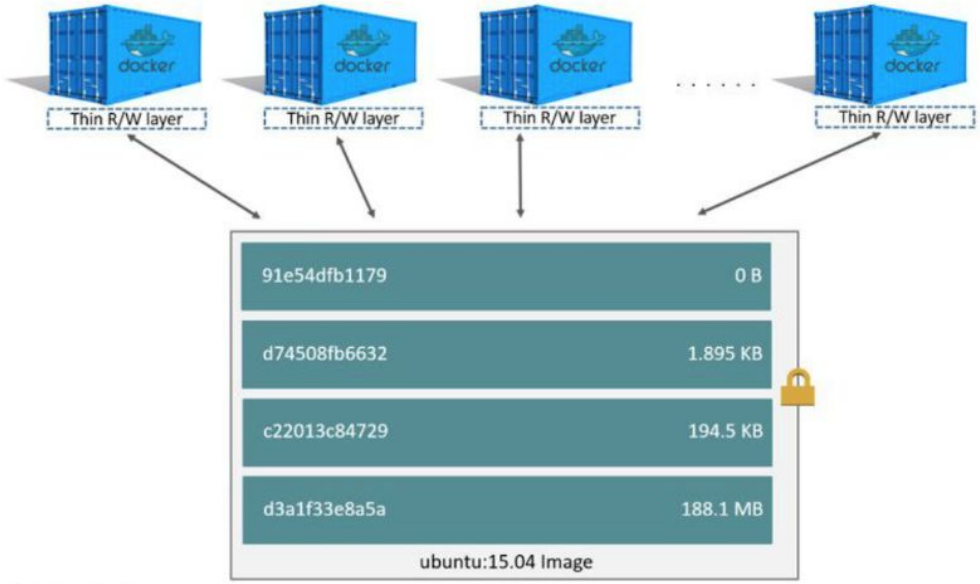
Why Docker?

- Improve Scalability
  - Vertical
  - Horizontal



Why Docker?

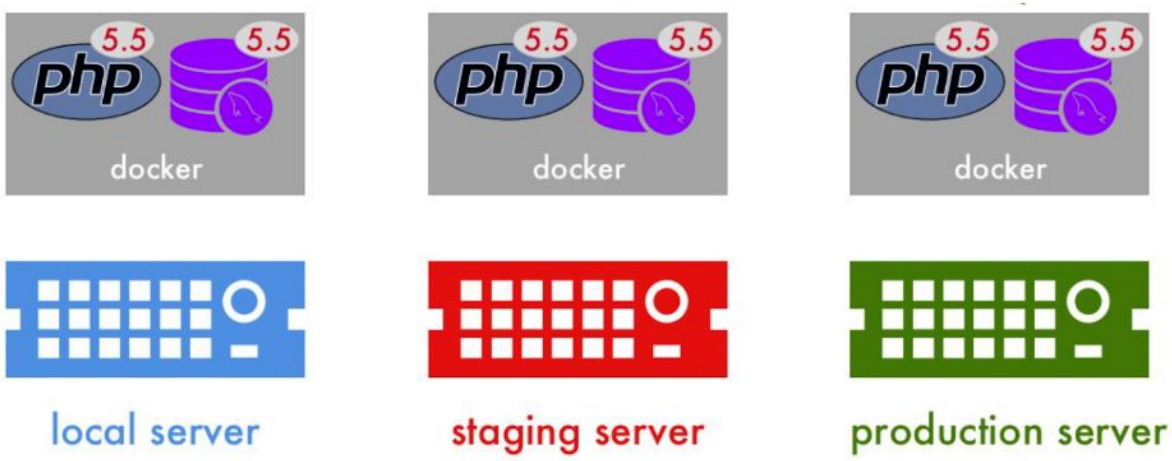
- Simple and fast deployment
  - quickly create new containerized instances or rapidly destroy multiple containers



docker

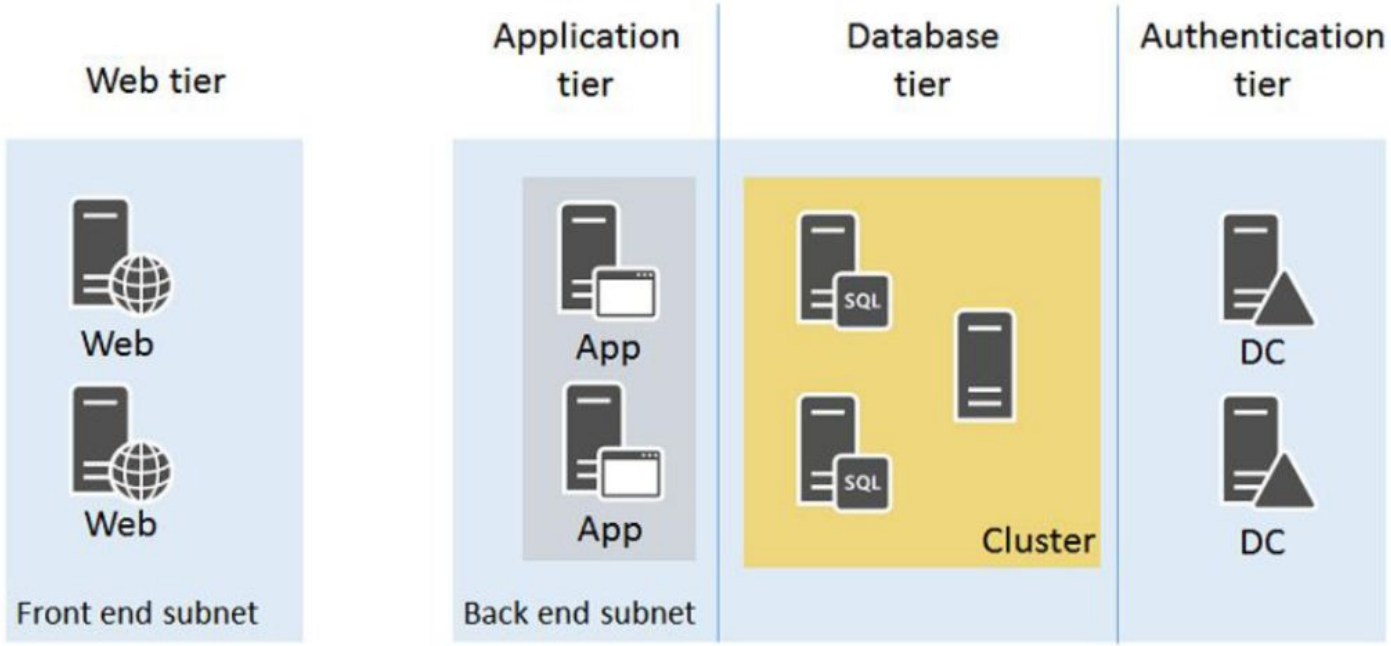
Why Docker?

- Enhance Productivity
  - promotes a rapid development environment
  - simplify the installation process and decrease dependency errors



Why Docker?

- Improve Security
  - each application's major process apart from one another in separate containers

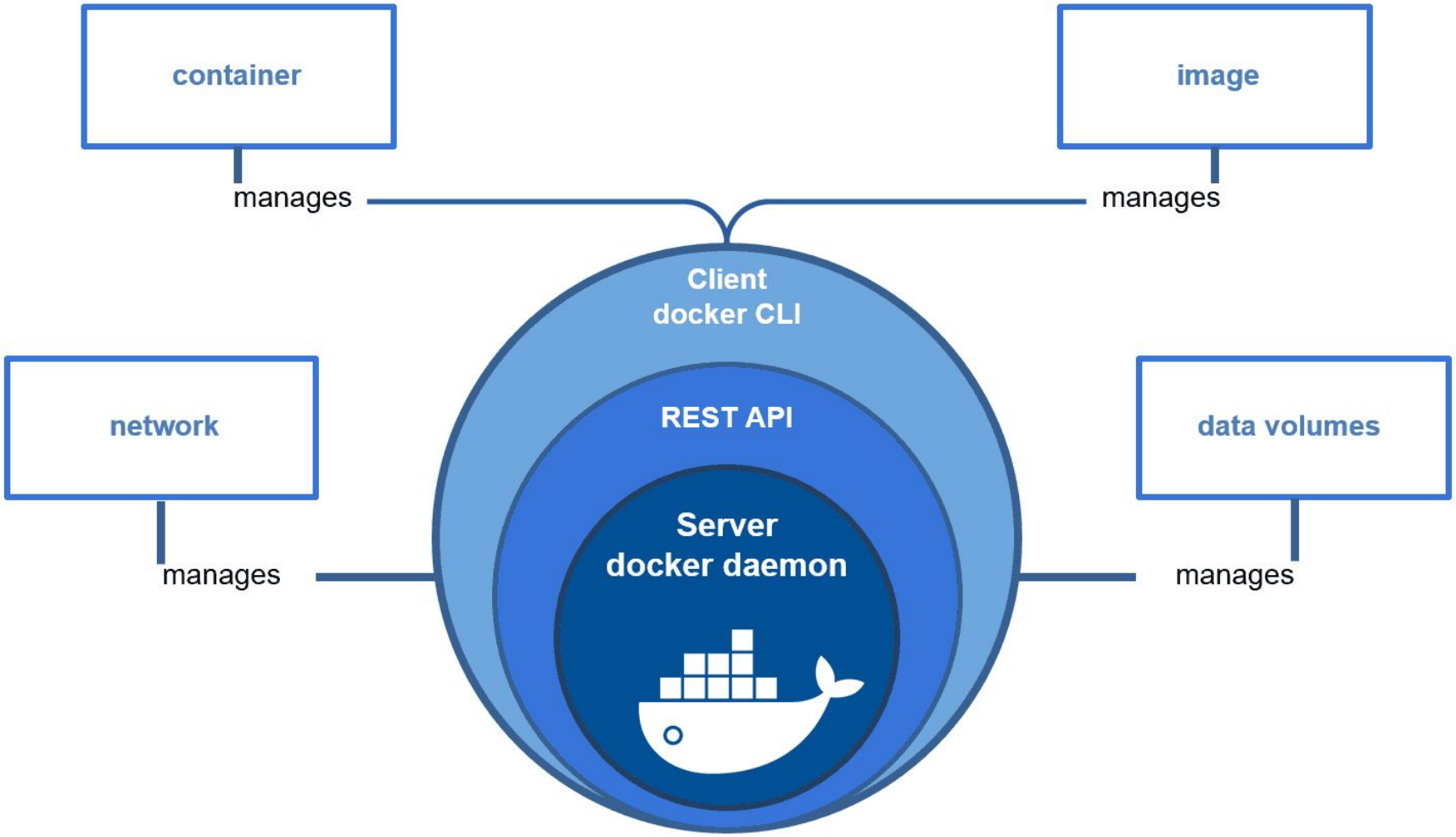


docker

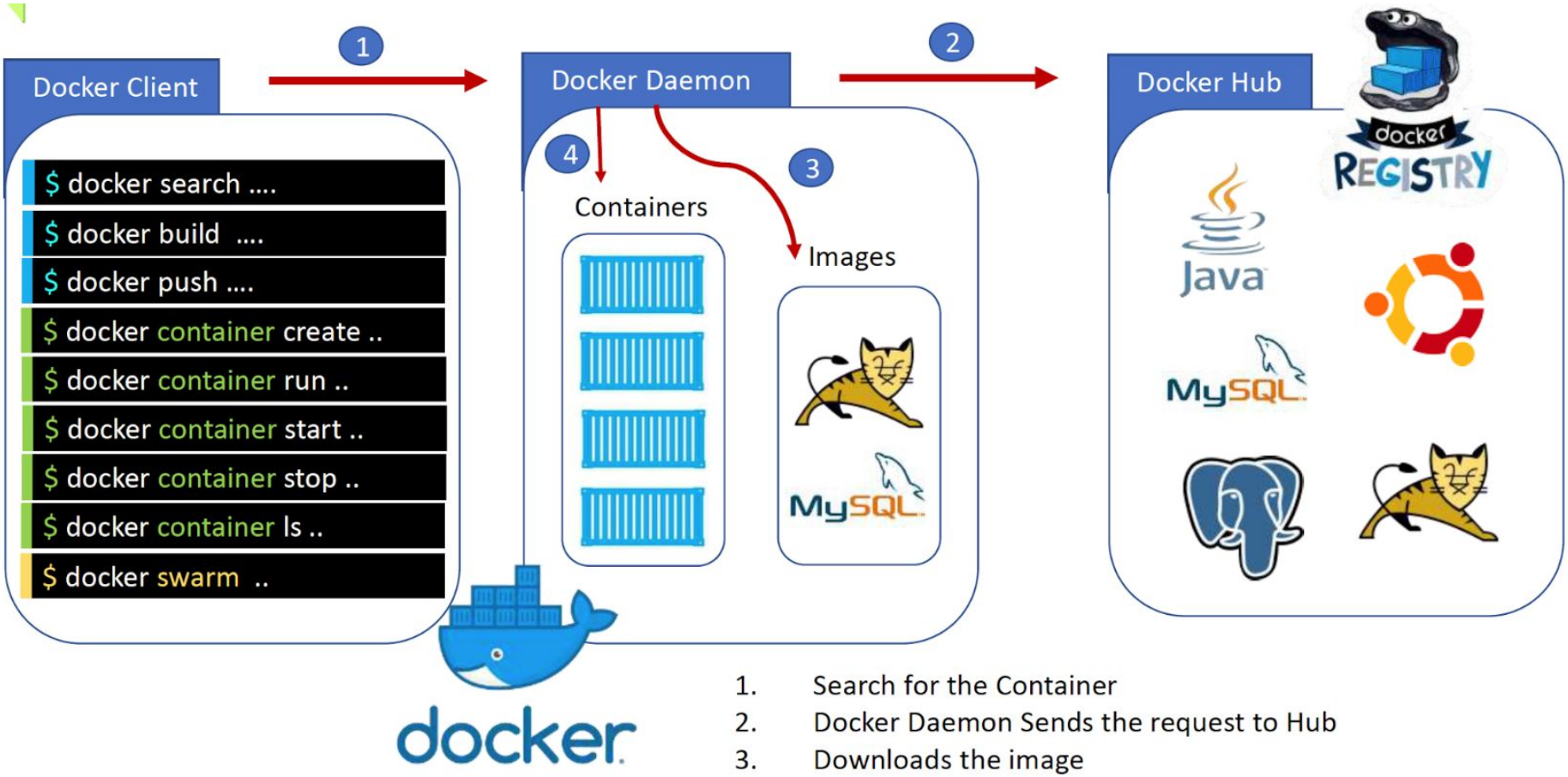
Docker alternatifleri

Podman: Docker'a benzer, daemon'suz, root'suz çalışır.
CRI-O: Kubernetes için optimize, CRI entegrasyonu, hafif.
containerd: Docker bileşeni, Kubernetes uyumlu, basit.
LXC/LXD: Eski konteyner, hypervisor benzeri yönetim.
Buildah: Dockerfile'sız imaj oluşturma, esnek.
Singularity: HPC için güvenlik odaklı konteyner.

Docker Engine Components



How Docker Works..



1. Search for the Container
2. Docker Daemon Sends the request to Hub
3. Downloads the image
4. Run the Container from the image

Docker Registries Overview

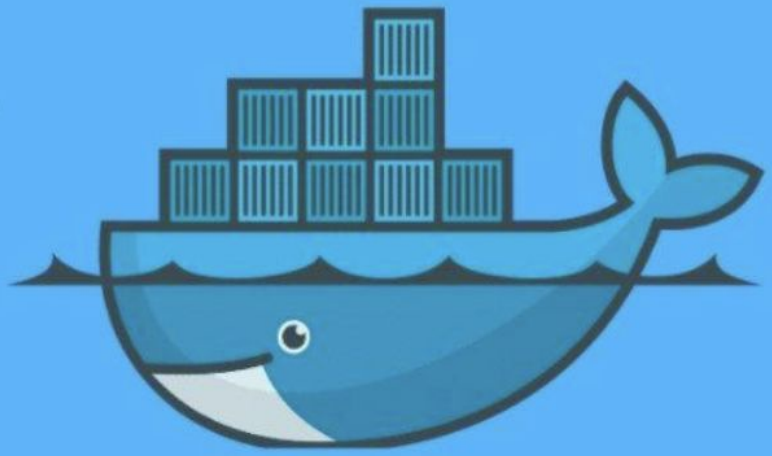
Public Registries

1. **Docker Hub:** The default registry, hosting a vast collection of public images.
2. **Google Container Registry (GCR):** Provided by Google Cloud Platform.
3. **Amazon Elastic Container Registry (ECR):** Fully managed by AWS.
4. **Azure Container Registry (ACR):** Private registry service by Microsoft Azure.

Private Registries

1. **Docker Official Registry:**
   - Can be set up locally using Docker's registry image.
   - Provides control over access and image management.
   - Ideal for storing proprietary or confidential images.
2. **Nexus Repository:**
   - Supports multiple formats (Docker, Maven, npm, etc.).
   - Comprehensive artifact management.
   - Suitable for managing various types of artifacts.
3. **Harbor:**
   - Focused on container images.
   - Offers vulnerability scanning, content signing, and replication.
   - Well-integrated with Kubernetes and cloud-native environments.

Basic Docker Commands



Essential Resources

Labs
http://labs.play-with-docker.com
https://killercoda.com/docker
https://www.docker.com/play-with-docker
https://kodekloud.com/pages/free-labs/
https://engineer.kodekloud.com/

Documents
https://docs.docker.com/
https://github.com/wsargent/docker-cheat-sheet#prerequisites
https://dockerlabs.collabnix.com/workshop/docker/
https://qconsf2017intro.container.training/

Try Server
https://aws.amazon.com/free/
https://cloud.google.com/free/
https://try.digitalocean.com/performance/
https://azure.microsoft.com/en-us/free/

Docker Install

Install Docker Engine

This section describes how to install Docker Engine on Linux, also known as Docker CE. Docker Engine is also available for Windows, macOS, and Linux, through Docker Desktop. For instructions on how to install Docker Desktop, see:

- Docker Desktop for Linux
- Docker Desktop for Mac (macOS)
- Docker Desktop for Windows

Supported platforms

Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	ppc64le	s390x
CentOS	✅	✅		✅	
Debian	✅	✅	✅	✅	
Fedora	✅	✅		✅	
Raspberry Pi OS (32-bit)			✅		
RHEL	🚧	🚧			✅
SLES					✅
Ubuntu	✅	✅	✅	✅	✅
Binaries	✅	✅	✅		

🚧 = Experimental

Basic Commands

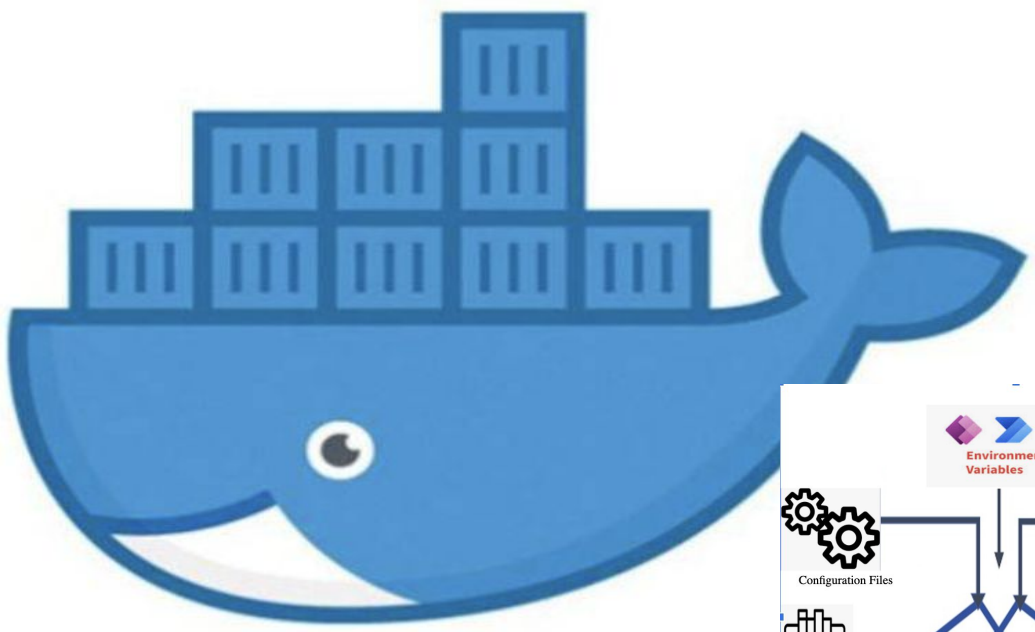
Komut	Açıklaması
`docker run`	**Yeni bir konteyner oluşturur ve başlatır.**
`docker ps`	**Çalışmakta olan konteynerleri listeler.**
`docker ps -a`	**Tüm konteynerleri (çalışan ve durdurulmuş) listeler.**
`docker stop <container_id>`	**Belirtilen konteyneri durdurur.**
`docker start <container_id>`	**Durdurulmuş bir konteyneri başlatır.**
`docker restart <container_id>`	**Bir konteyneri durdurur ve yeniden başlatır.**
`docker rm <container_id>`	**Belirtilen konteyneri siler.**
`docker exec -it <container_id> <command>`	**Çalışmakta olan bir konteynerde komut çalıştırır.**
`docker logs <container_id>`	**Belirtilen konteynerin log (günlük) bilgilerini gösterir.**

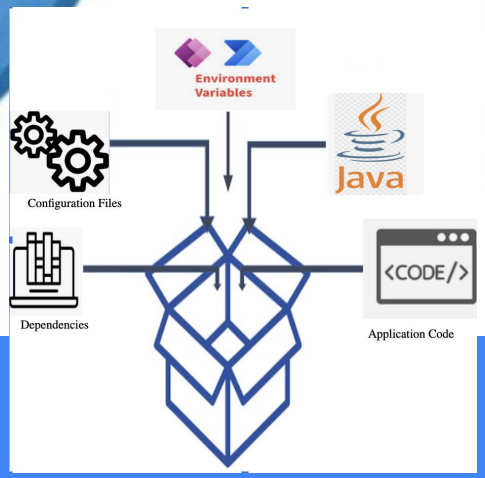
Basic Commands

Komut	Açıklaması
`docker logs <container_id>`	**İlgili Container ‘in loglarını gösterir**
`docker exec <container_id> <command>`	**Çalışan bir Container içinde bir komut koşturmak için kullanılır**
`docker exec -it <container_id> /bin/bash`	**Çalışan bir Container içinde terminal açmak için kullanılır.**
`docker pull <image_name>`	**Docker Hub'dan belirtilen imajı indirir.**
`docker network ls`	**Docker ağlarını listeler.**
`docker volume ls`	**Docker volume’ leri listeler**
`docker rm <container_id>`	**Belirtilen konteyneri siler.**
`docker info`	**Docker'ın genel durum ve yapılandırma bilgilerini gösterir.**

Docker basic commands

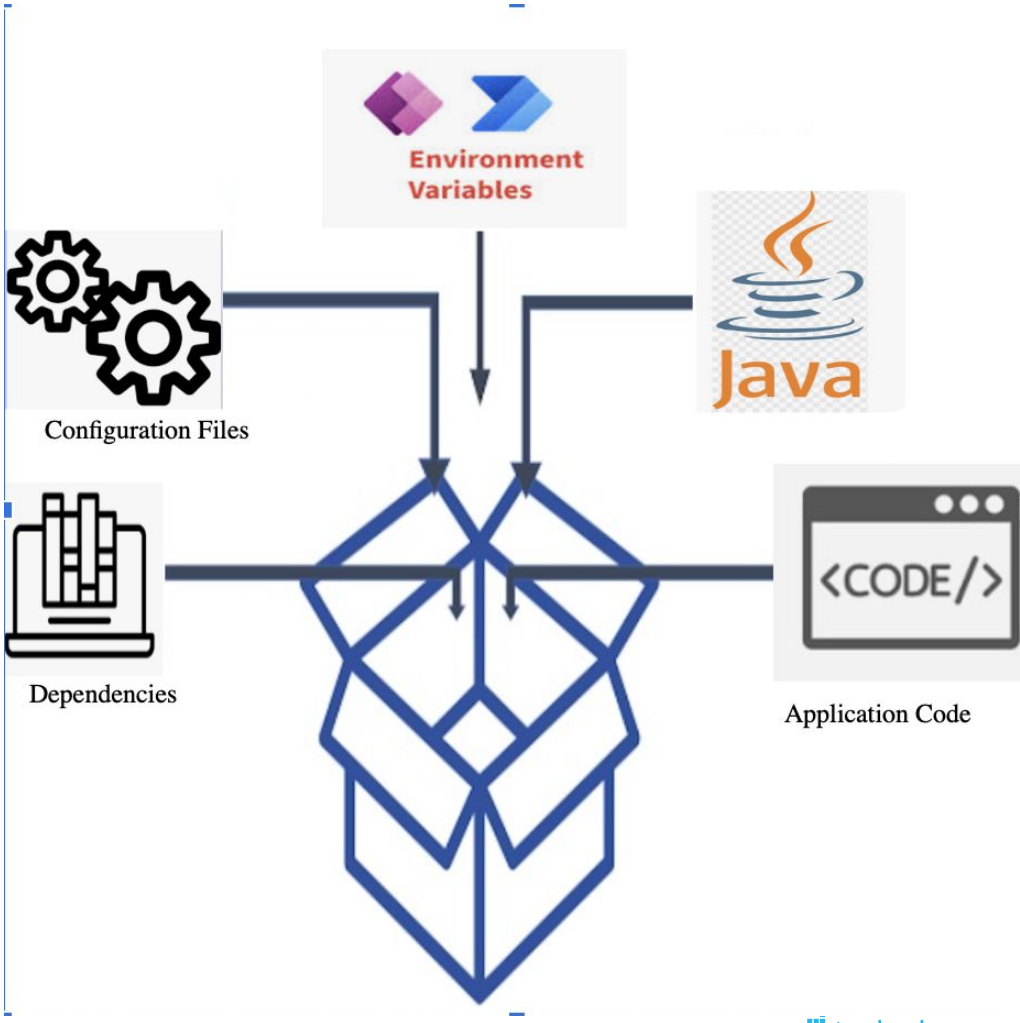
Docker Containers & Docker Images





What is a docker image?

Docker image: Bir uygulamanın çalışması için gereken her şeyi (kütüphaneler, bağımlılıklar, yapılandırma dosyaları vb.) tek bir paket içinde bir araya getirir.

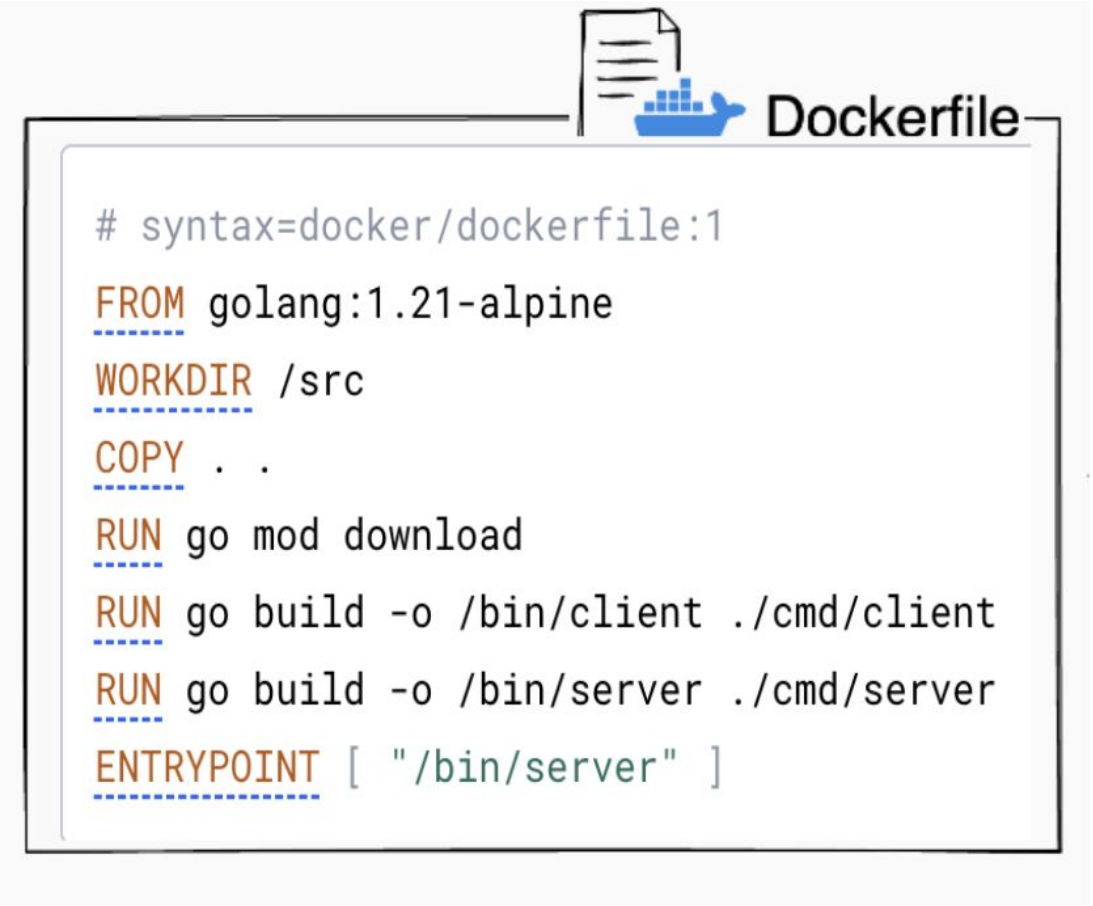


docker

Docker Images

- Read only template used to create containers
- Stored in the Docker Hub or in your local registry
- Image is a Read Only Template and is use to create container
- You can’t Edit , But you can delete
- 2 Method to create Image (Interactive Method ,Dockerfile Method)

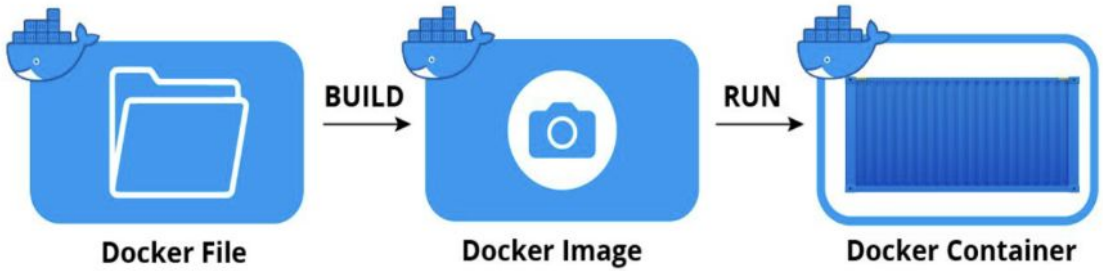
Dockerfile

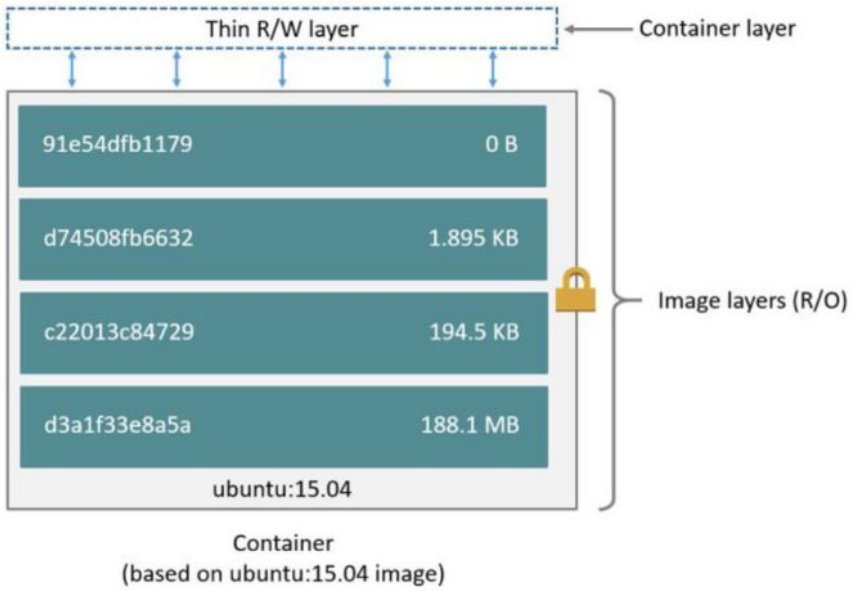


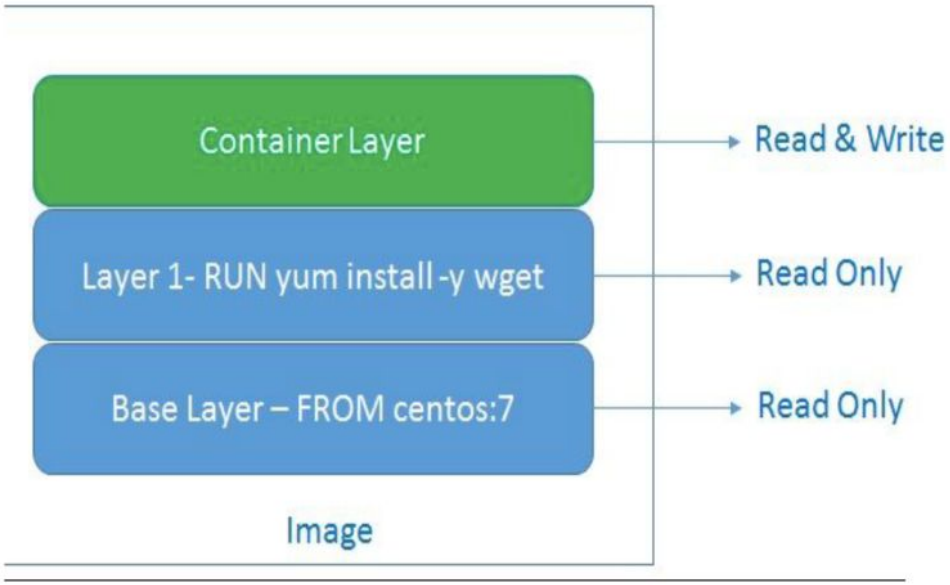
The application

```
.
├── Dockerfile
├── cmd
│   ├── client
│   │   ├── main.go
│   │   ├── request.go
│   │   └── ui.go
│   └── server
│       ├── main.go
│       └── translate.go
├── go.mod
└── go.sum
```

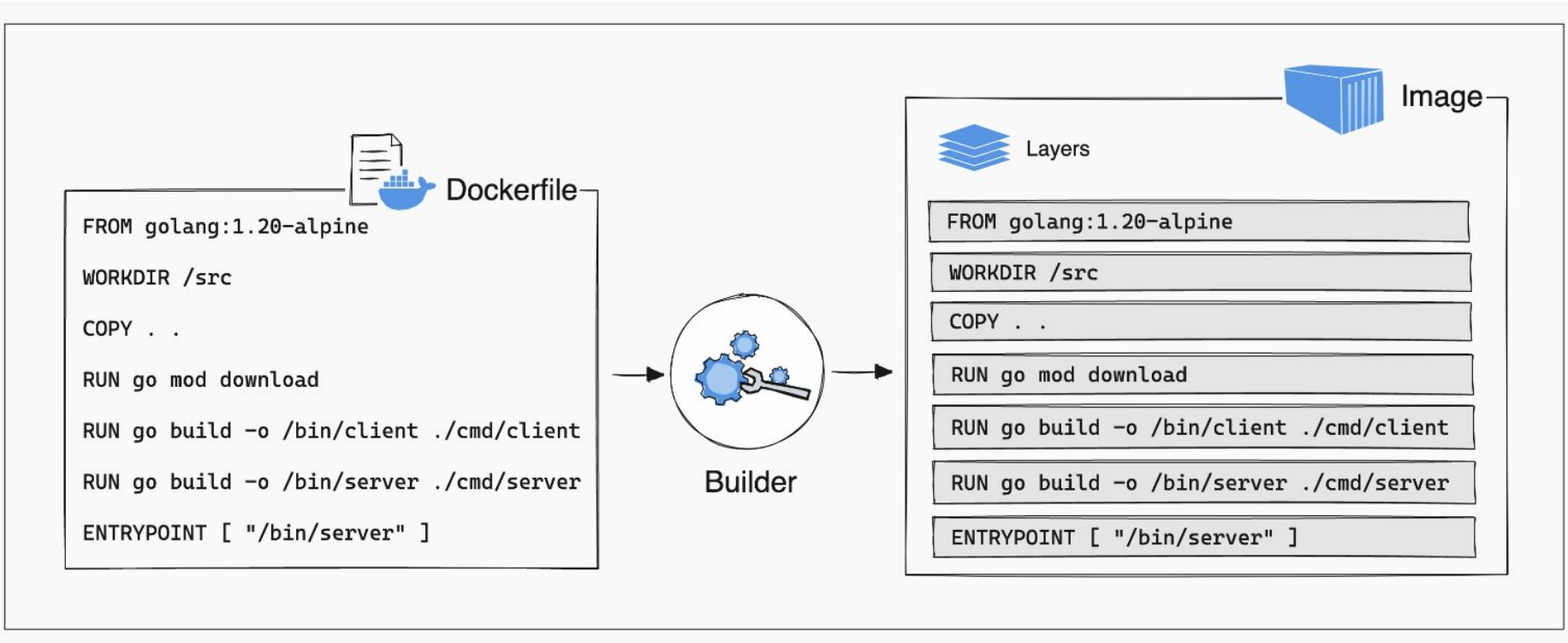
Build Docker Image Using Dockerfile



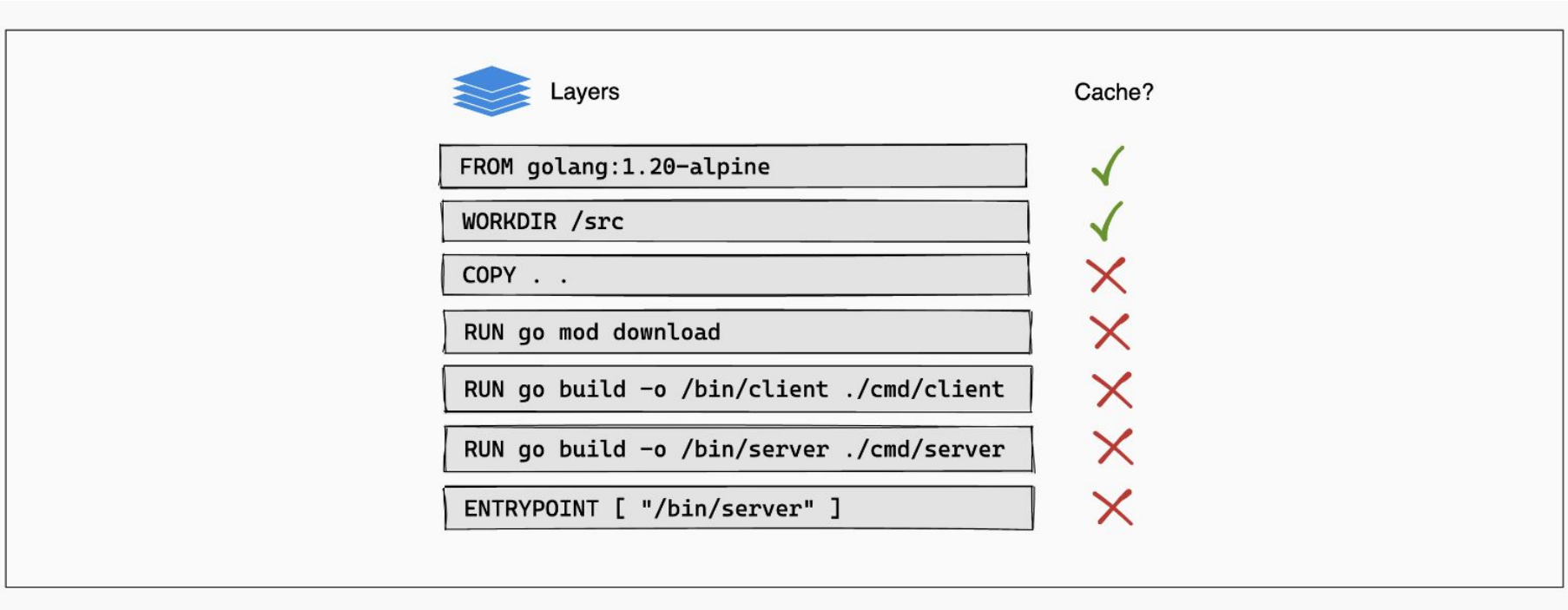




Docker Layers

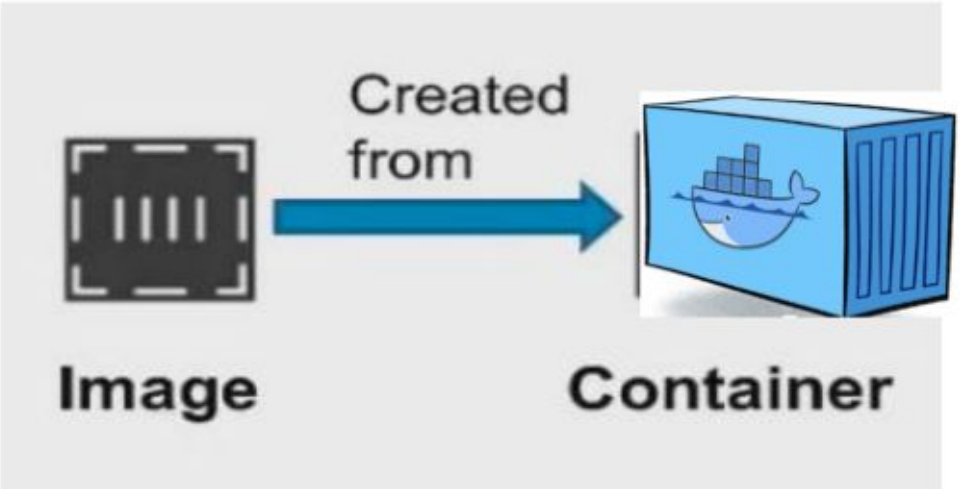


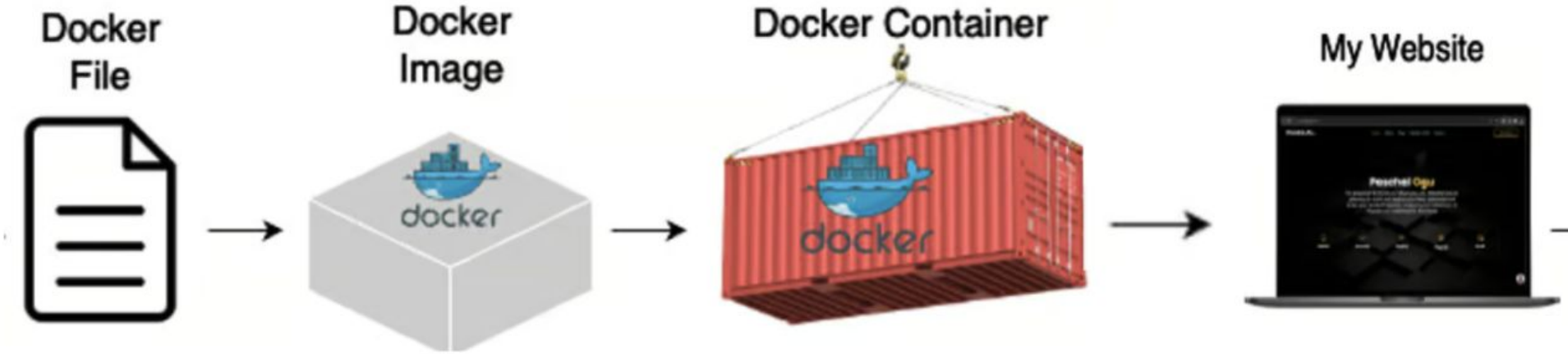
Cached Layers



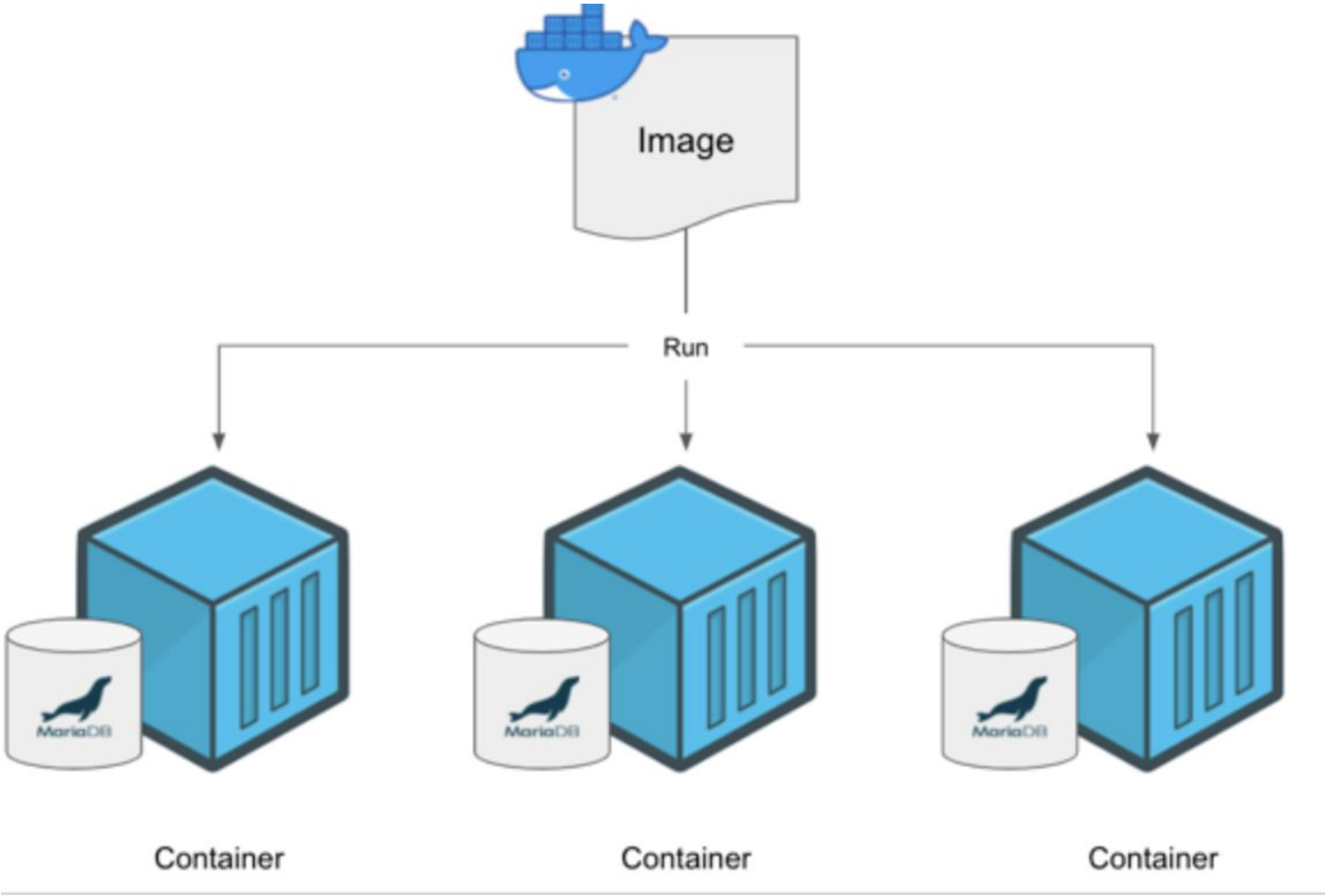
What is a container?

Container: Docker image'ın çalıştırılmış halidir.

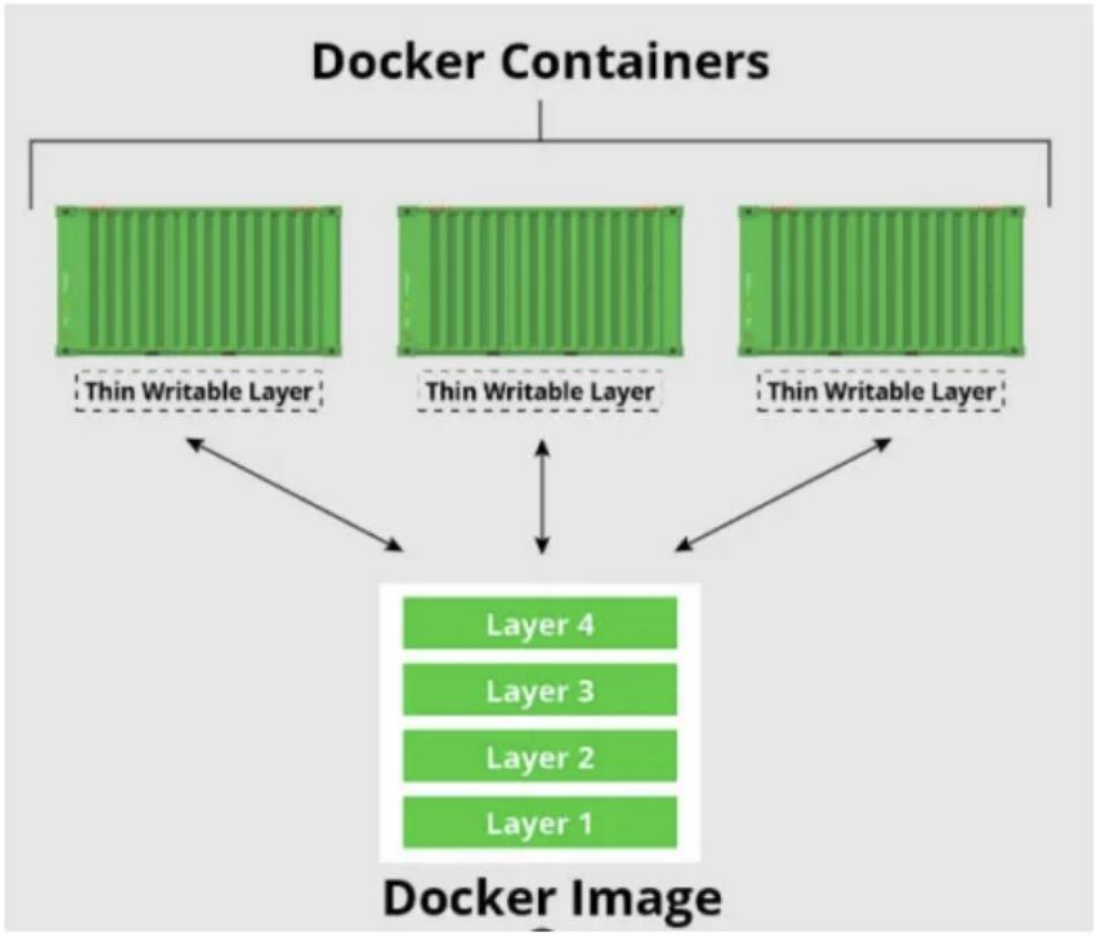


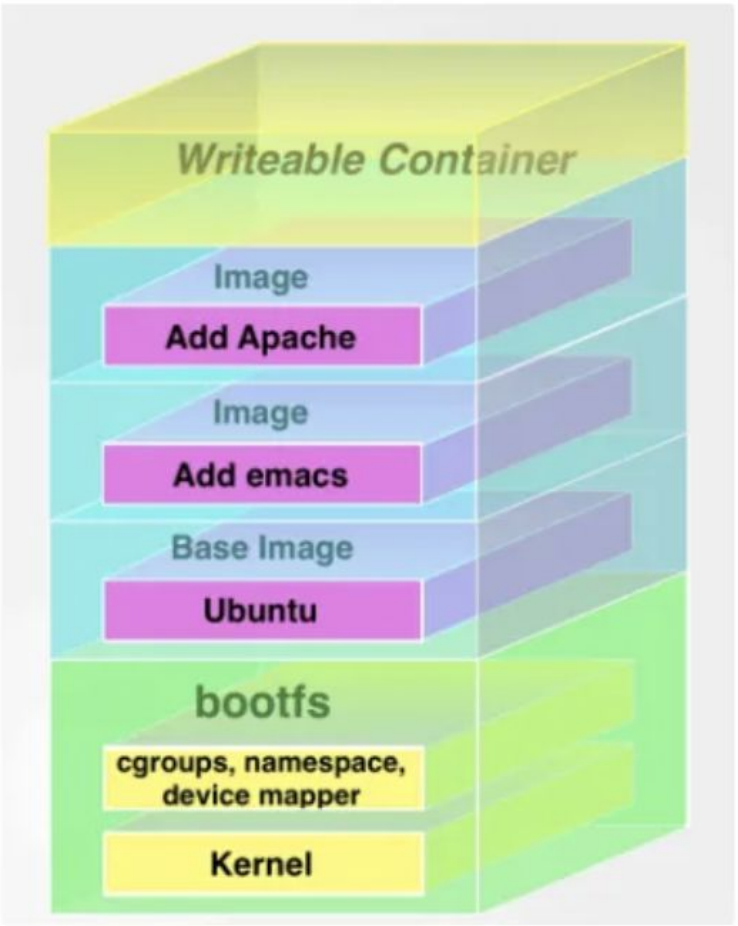


Docker Container Docker image

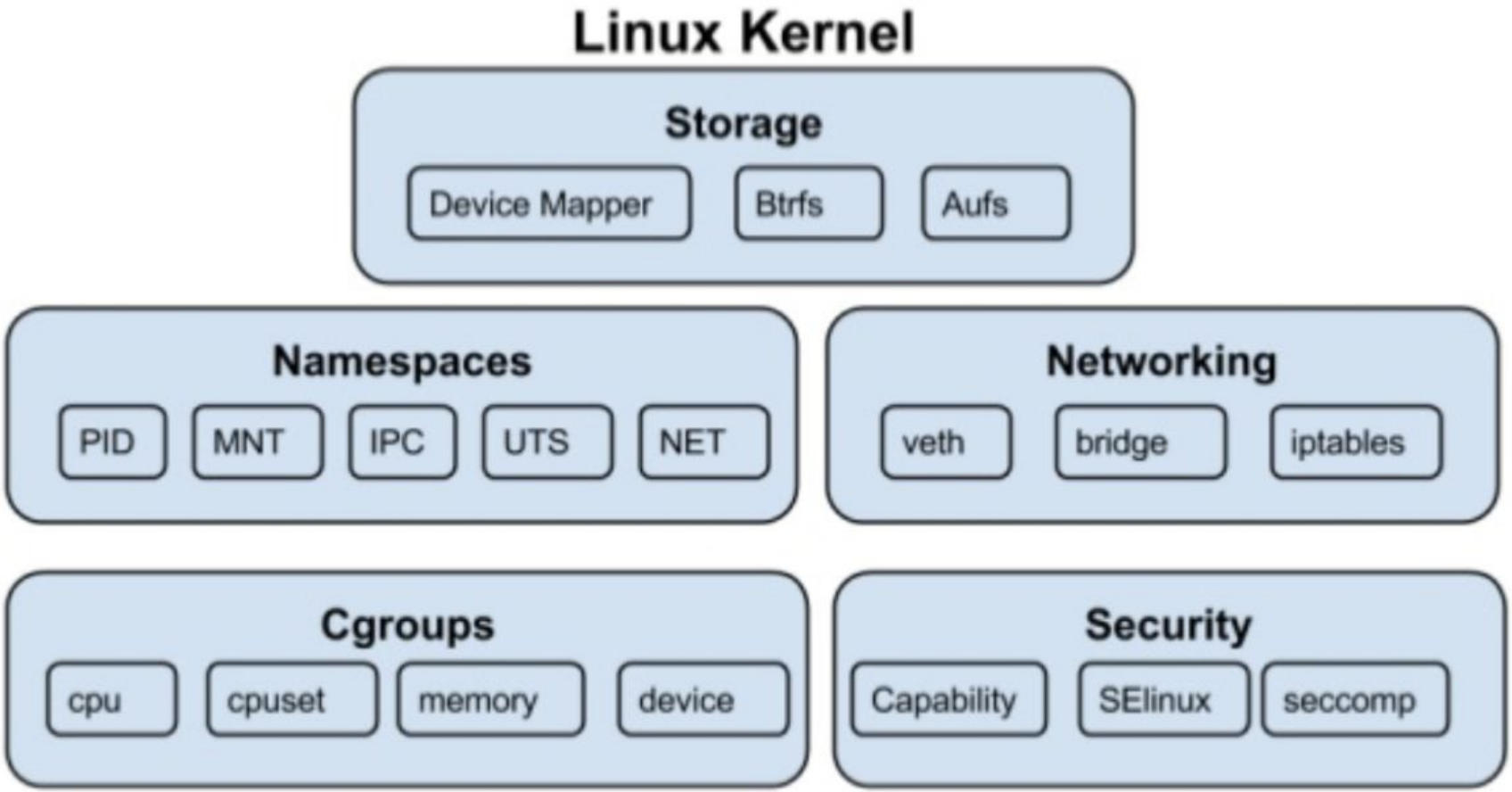


Docker Container Docker image

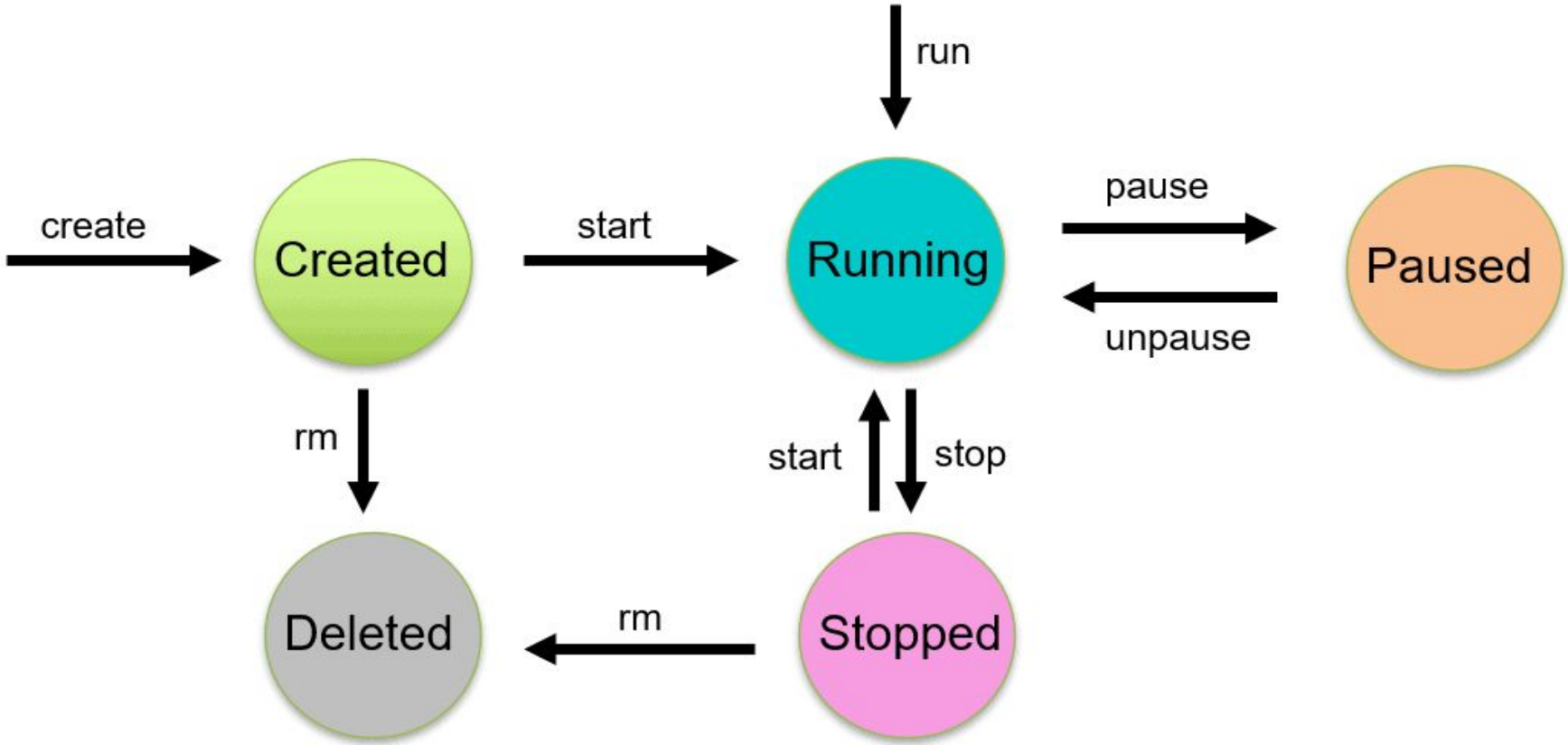




Docker works by using linux containers and kernel facilities



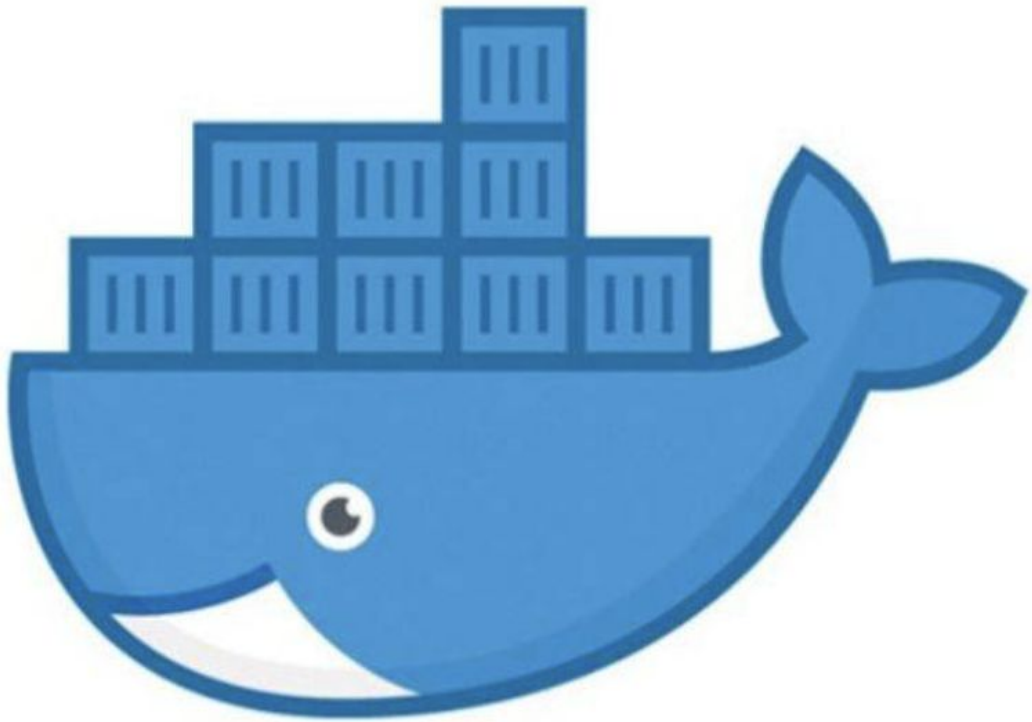
Docker Container Lifecycle



Docker Image ve Container

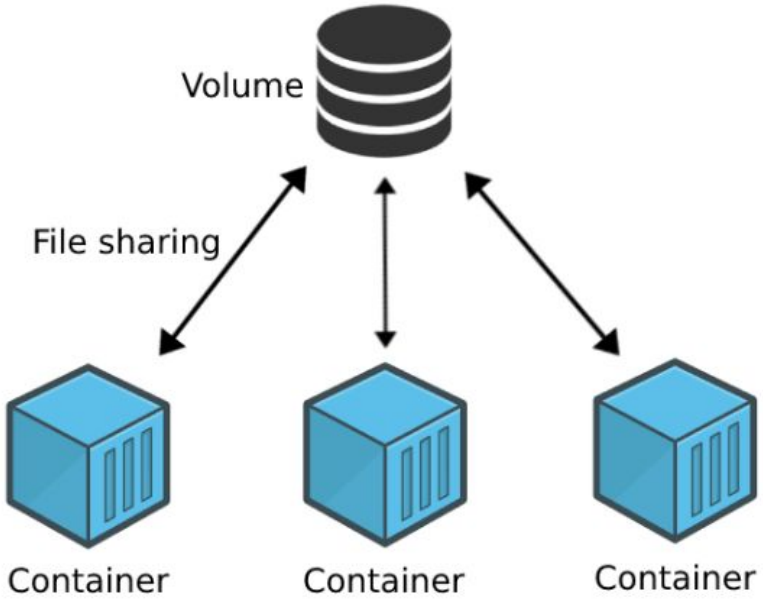
Özellik	Docker Image	Docker Container
Amaç	Uygulamanın tutarlı bir şekilde dağıtımını sağlar.	Uygulamanın izole bir ortamda çalışmasını sağlar.
Tanım	Uygulamanın kodu ve bağımlılıkları ile oluşturulmuş bir şablondur.	Docker image'ının çalıştırılan bir örneğidir.
Değiştirilebilirlik	Değiştirilemez (salt okunur).	Çalışırken değiştirilebilir (okunur-yazılır).
Oluşturulma Yöntemi	`Dockerfile` kullanılarak ve `docker build` komutu ile oluşturulur.	`docker run` veya `docker create` komutları ile oluşturulur.
Paylaşılabilirlik	Başkalarıyla paylaşılabilir ve Docker Hub gibi kayıtlı bir yerde saklanabilir.	Paylaşım için önce bir image olarak kaydedilmelidir; çalışan bir container yalnızca çalıştığı hostta bulunur.
Depolama	Yerel olarak veya Docker registry'de saklanır.	Çalıştığı host makinede saklanır.

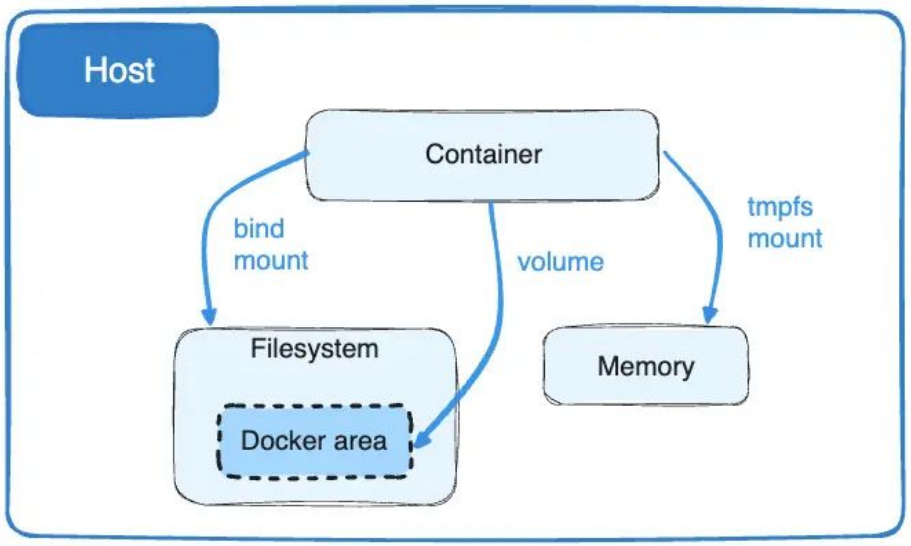
Docker Volume



Docker Volume

Docker volume, konteynerler arasında veri paylaşımını ve veri sürekliliğini sağlamak için kullanılan bir mekanizmadır. Konteyner silinse bile veriler kaybolmaz.





Docker Volume Türleri

1. Docker Managed Volumes:

Docker tarafından yönetilen ve izole edilmiş veri depolama alanı.

- `docker volume create <volume-name>`
- `docker run -v my-volume:/data my-image`

2. Bind Mounts (Host Directory Mounts):

Bind mount, host sistemindeki bir dizinin konteyner içine birebir bağlanmasıdır.

- `docker run -v /host/dir:/container/dir <image-name>`

Volume Storage Location

Linux: */var/lib/docker/volumes/*

Windows: *\\wsl$\docker-desktop-data\version-pack-data\community\docker\volumes*

Docker Volume Komutları

Komut	Açıklama
`docker volume create <volume-name>`	Yeni bir Docker volume oluşturur. İsim vermezseniz, Docker otomatik bir isim atar.
`docker volume inspect <volume-name>`	Belirtilen volume hakkında detaylı bilgi verir, nerede saklandığı, hangi konteynerlerin kullandığı gibi.
`docker volume ls`	Sistemdeki tüm volume'leri listeler.
`docker volume rm <volume-name>`	Belirtilen volume'ü siler. Kullanılmayan (boşta olan) volume'leri silerken kullanılır.
`docker volume prune`	Kullanılmayan tüm volume'leri temizler. Dikkatli kullanılmalıdır, çünkü geri döndürülemez.

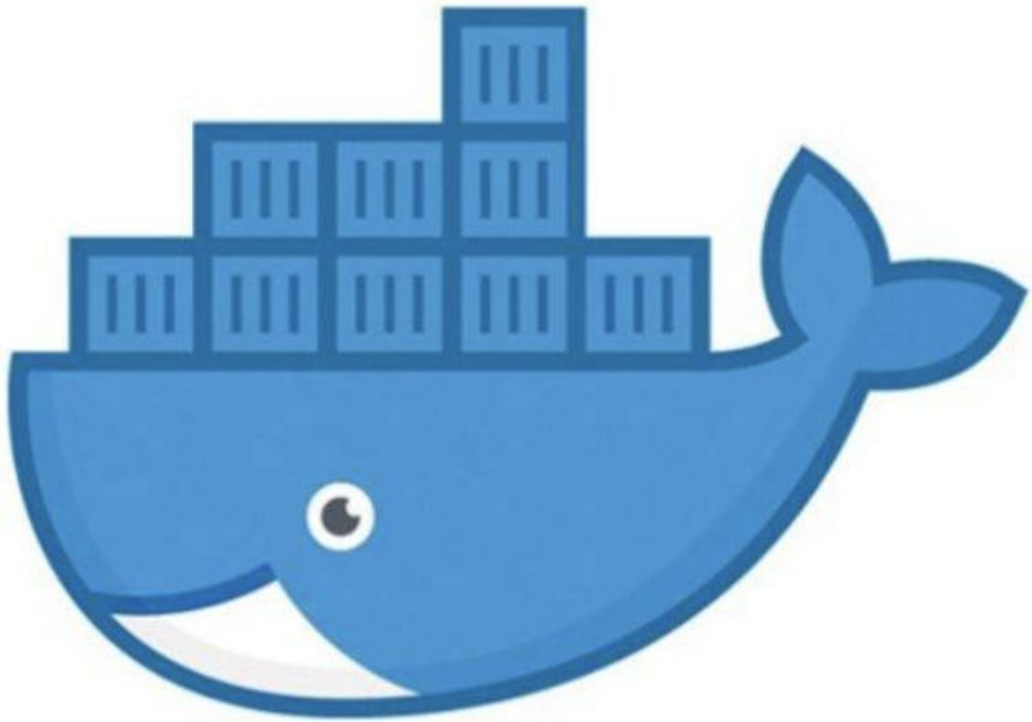
docker

Docker Volume Komutları

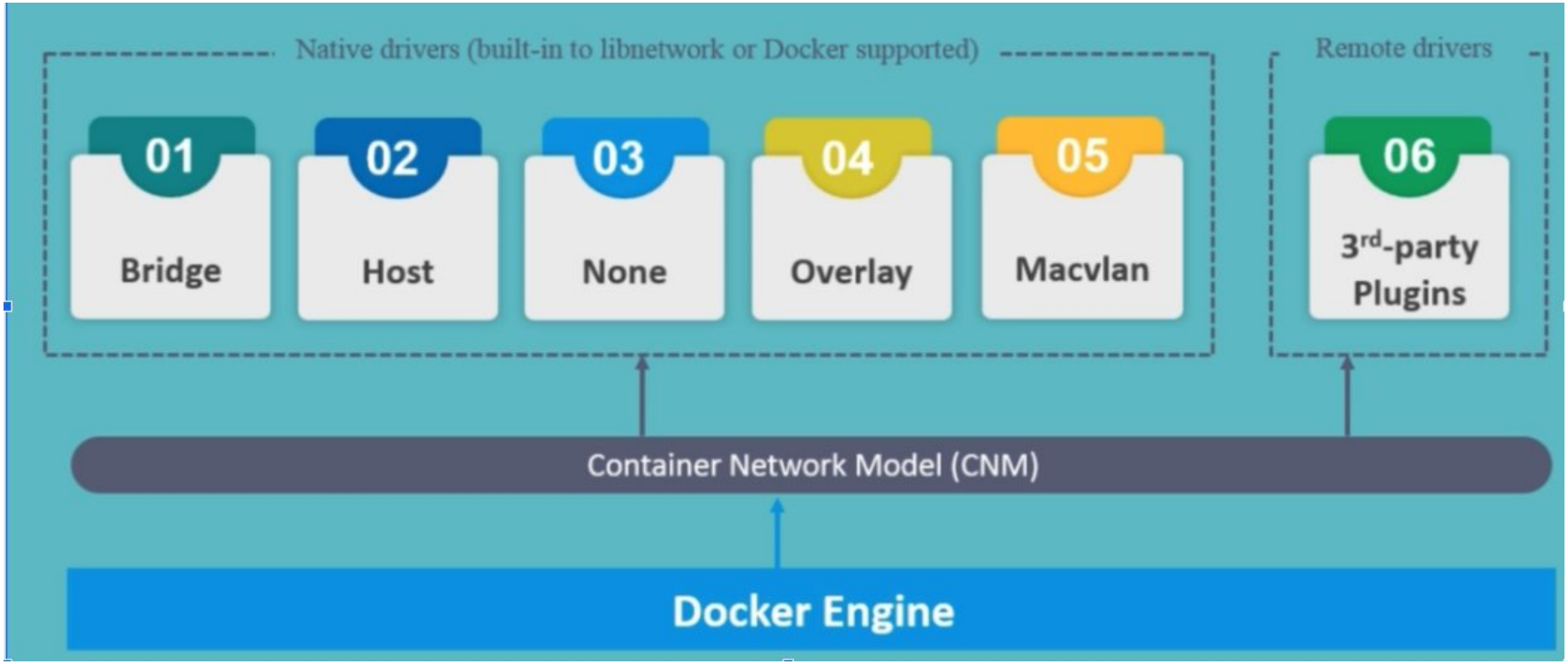
Komut	Açıklama
`docker run -v <volume-name>:/path/in/container <image-name>`	Volume'ü bir konteyner içine bağlayarak çalıştırır. Bu komutla volume konteynere bağlanır ve veriler korunur.
`docker volume create --driver <driver-name> <volume-name>`	Özel bir storage driver kullanarak volume oluşturur. Örneğin, NFS veya başka bir üçüncü parti storage sistemi ile entegrasyon için.
`docker volume ls -f dangling=true`	Yalnızca "boşta kalan" volume'leri (hiçbir konteyner tarafından kullanılmayan) listeler.
`docker run --mount source=<volume-name>,target=/path/in/container <image-name>`	Mount komutunu kullanarak volume'ü konteyner içinde bir yol ile ilişkilendirir. `--mount` seçeneği daha gelişmiş kullanımlar için tercih edilir.

Docker volume

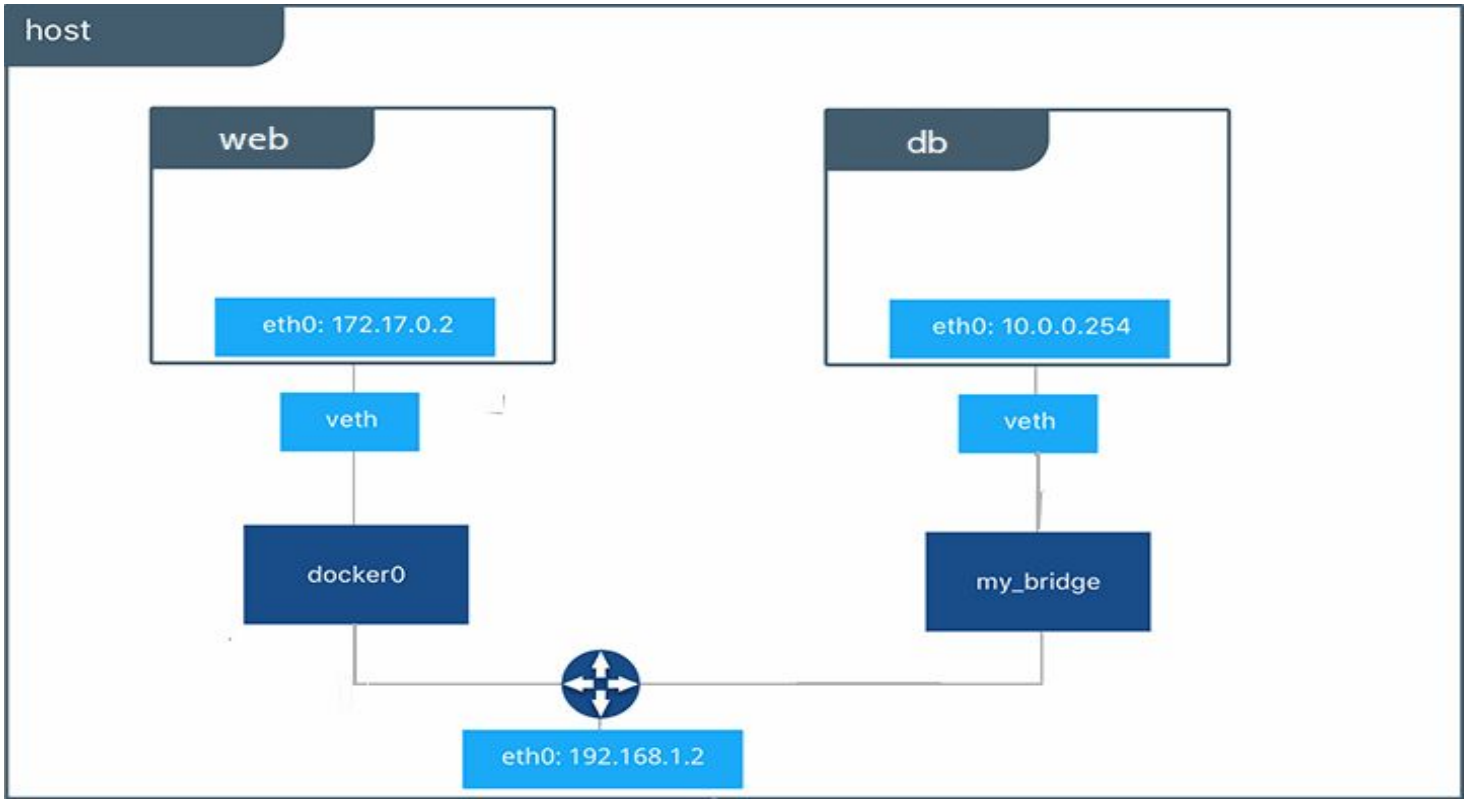
Docker Networking



Network Drivers



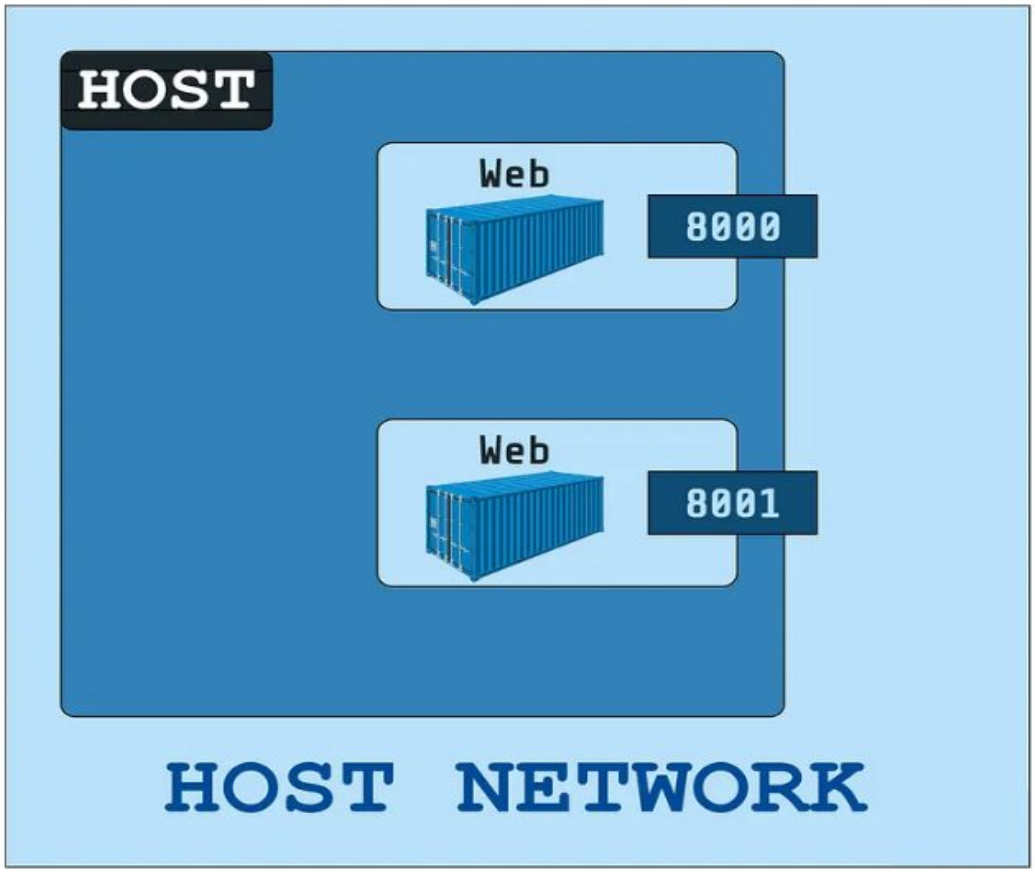
Bridge Network



Bridge Driver:

- ➔ Varsayılan Docker network driverıdır ve konteynerleri bridge ağına bağlar.
- ➔ Her bir bridge ağı, Docker ağına ait ayrı bir IP adres aralığına sahiptir.

Host Network

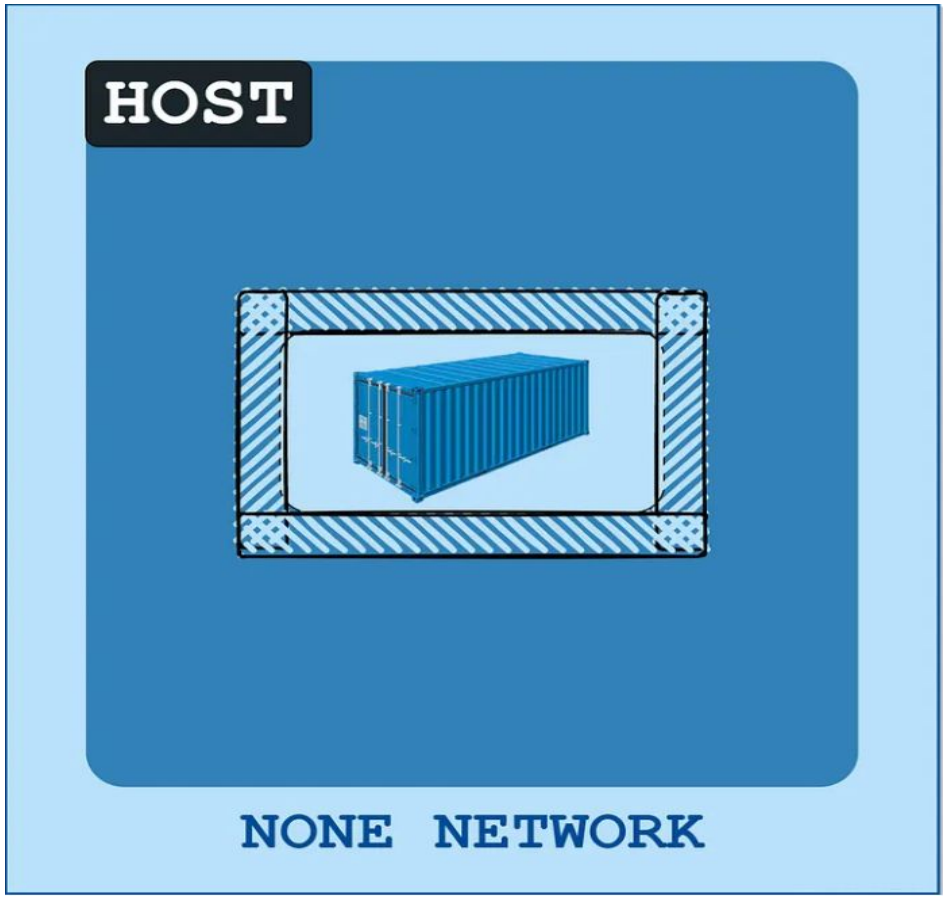


Host Driver:

➔ Konteyneri ana makinenin ağına direkt olarak bağlar. Bu, konteynerlerin ana makinedeki ağa erişim sağlar ve izolasyonu azaltır.

```
$  docker run -d --name nginx --network host nginx:latest
```

None Network

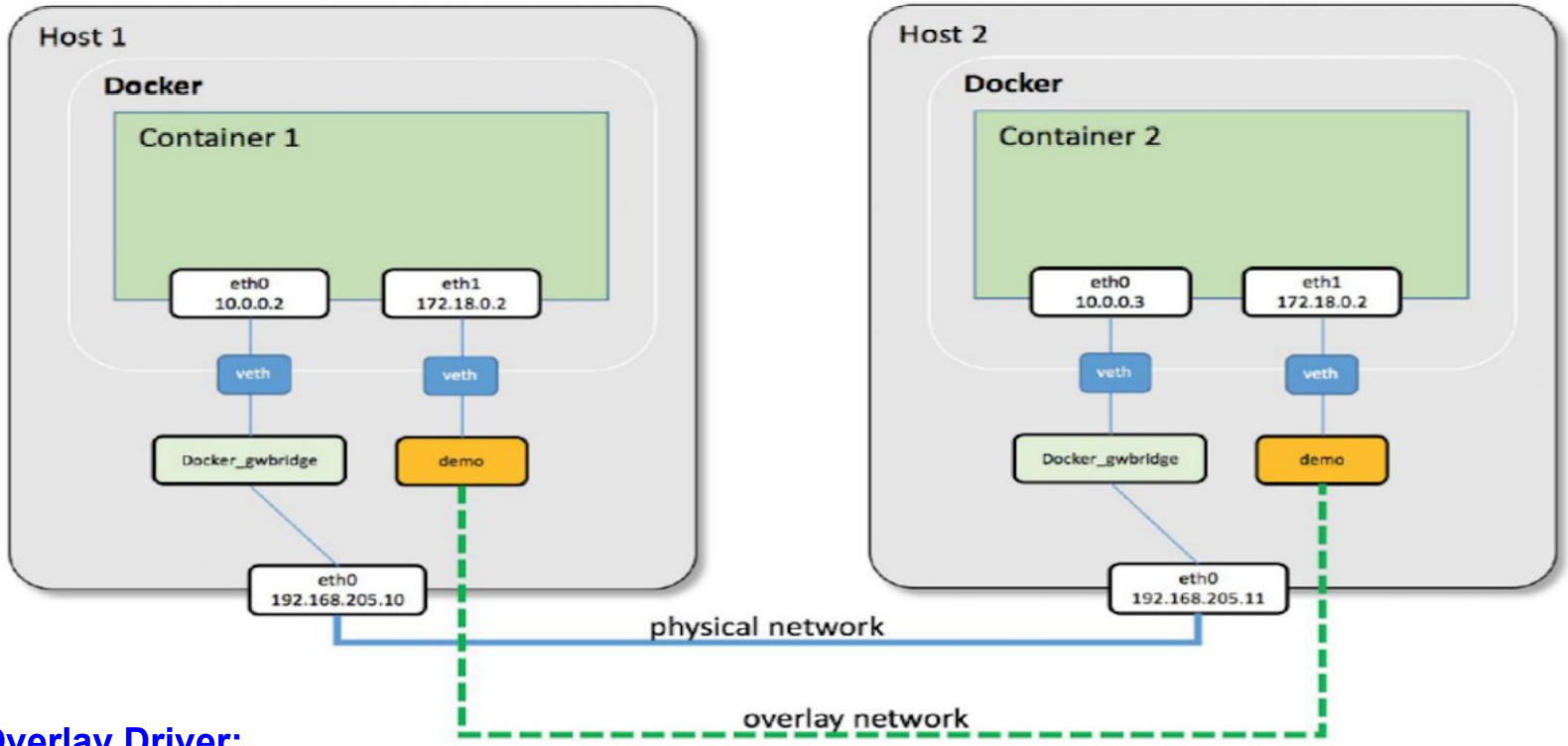


None Driver:

➔ Bu driver ile çalışan konteynerler, dış ağa veya diğer konteynerlere erişim sağlayamazlar.

```
$  docker run -d -p 80:80 --net=none --name networksuz busybox sleep 3000
```

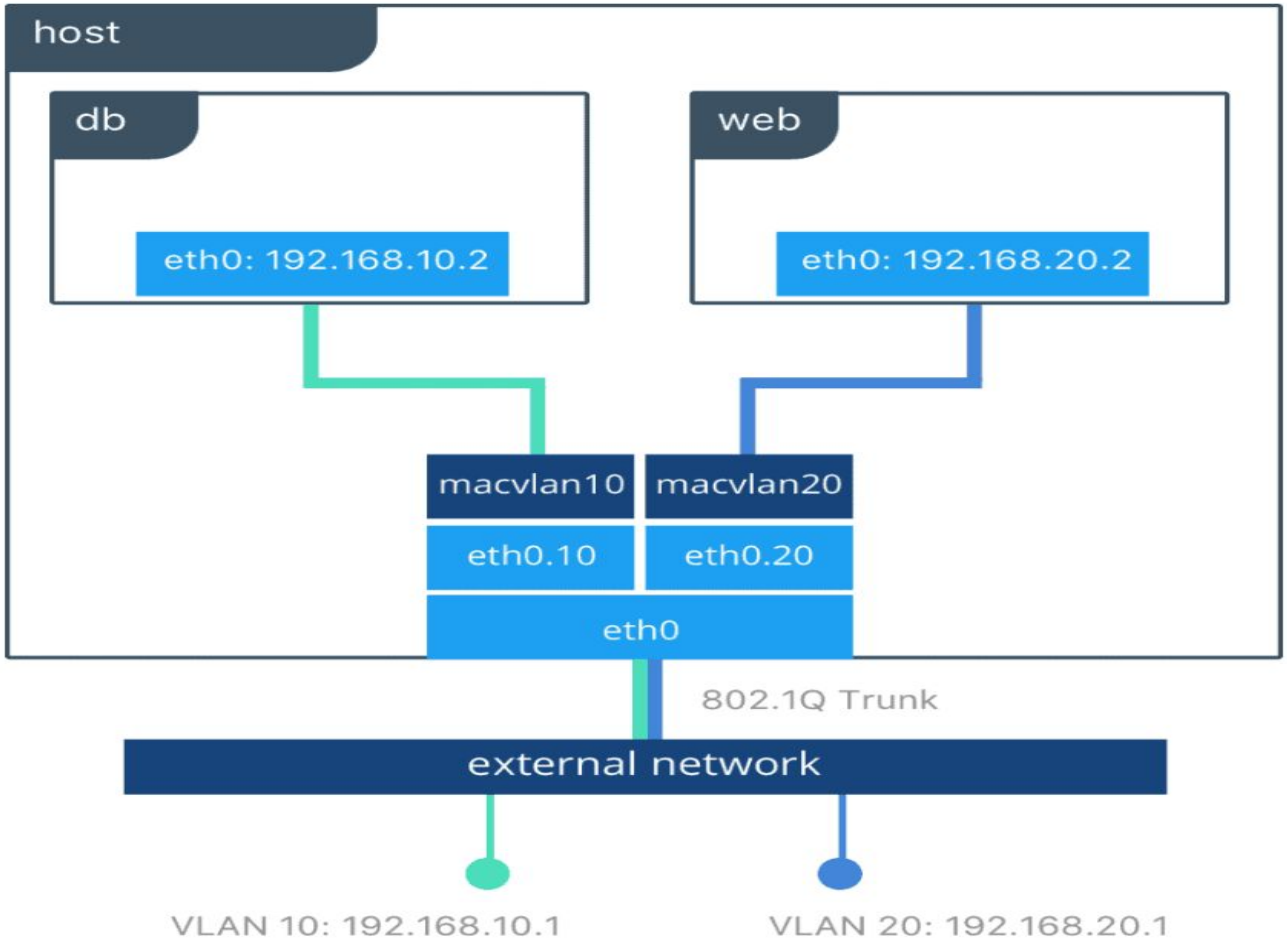
Overlay Network



Overlay Driver:

- ➔ Çoklu Docker hostları arasında konteyner iletişimini sağlar.
- ➔ Docker Swarm gibi orkestrasyon araçları ile sıkça kullanılır.

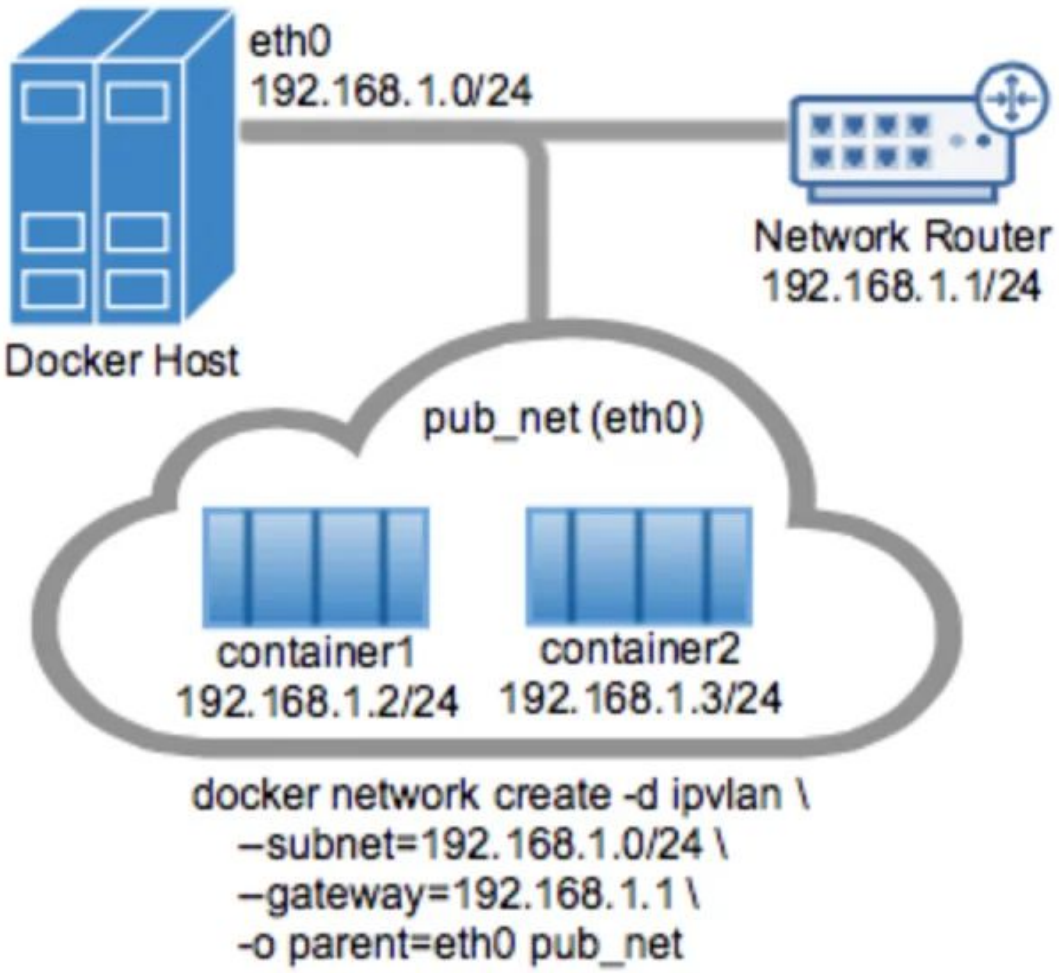
Macvlan Network



Macvlan Driver:

- ➔ Macvlan, konteynerlere fiziksel ağ benzeri IP ve MAC adresleri atar, böylece konteynerler fiziksel ağa yakın çalışır ve doğrudan ağ kaynaklarına erişebilir.
- ➔ Her Macvlan ağı, bağımsız IP ve MAC adres aralıklarına sahip olduğundan konteynerler arasında yüksek derecede izolasyon sağlar

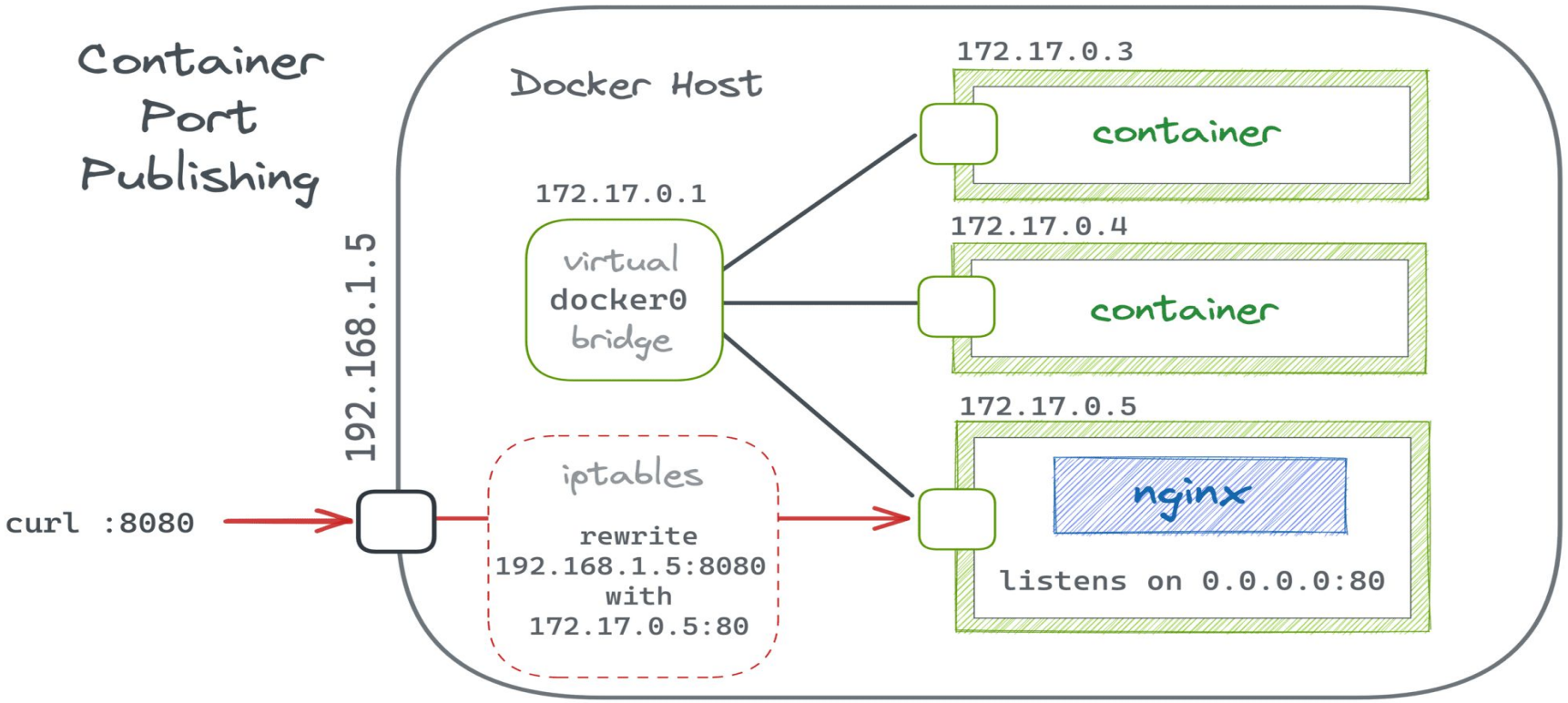
IPVlan Network



IPVlan Driver:

➔ IPVLAN aracılığıyla containerların yerel ağda gerçek bir cihaz gibi davranması sağlanırken MacVLAN ağları aracılığıyla containerlar yerel ağda gerçek bir mac adresi kullanır.

Docker Port Forwarding




```
$ docker run -d -p 8080:80 --name nginx nginx:latest
```

Docker Network Komutları

```
Usage:  docker network COMMAND

Manage networks

Commands:
  connect     Connect a container to a network
  create      Create a network
  disconnect  Disconnect a container from a network
  inspect     Display detailed information on one or more networks
  ls          List networks
  prune       Remove all unused networks
  rm          Remove one or more networks

Run 'docker network COMMAND --help' for more information on a command.
```

Örnek:

docker network create mynet
docker network connect mynet nginx
docker network disconnect bridge nginx
docker network inspect mynet
docker network rm mynet
docker network prune

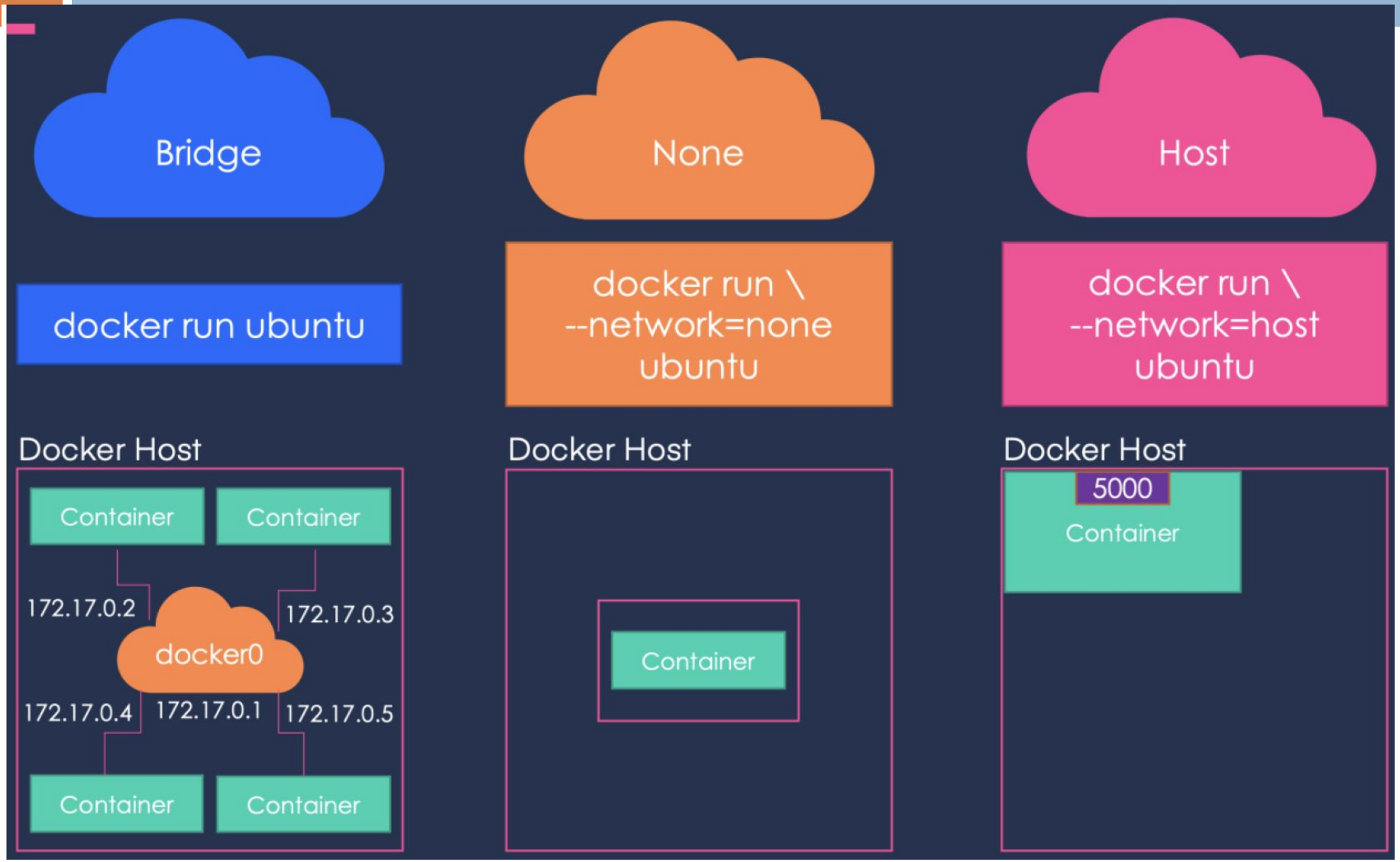
```
$ docker network create --driver bridge network1
$ docker run -d -network host nginx:latest
$ docker run -d -p 80:80 --net=none --name networksuz busybox sleep 3000
```

Docker Network DNS ile erişim

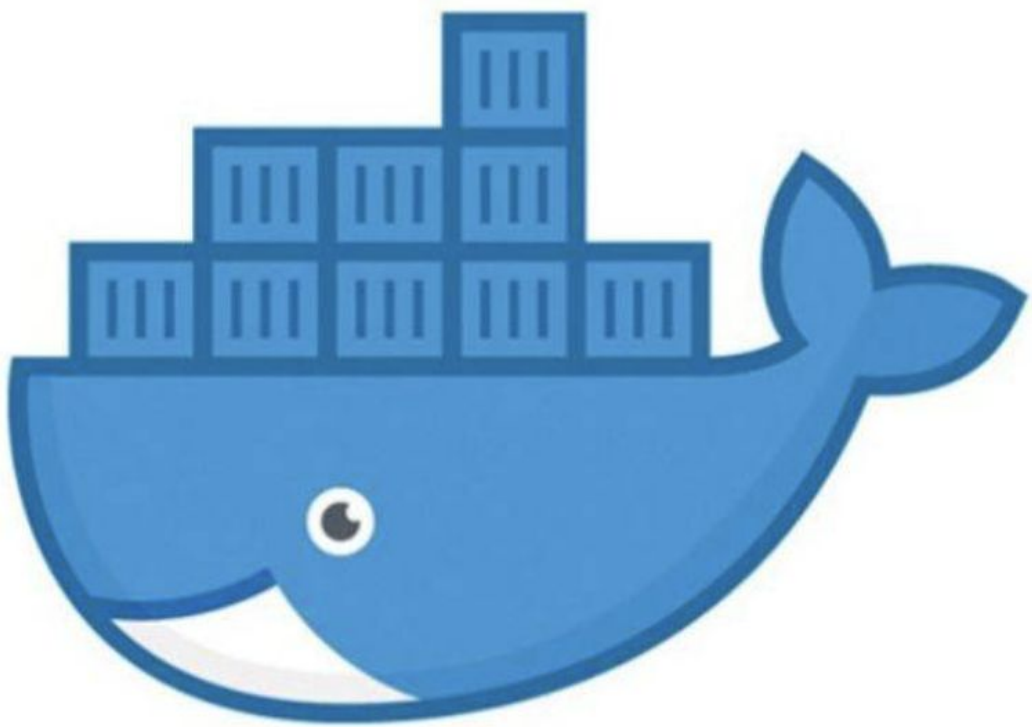
```
docker network create dns_net
docker container run -d --name nginx --network dns_net nginx
docker container run -d --name apache --network dns_net httpd
docker container ls
docker network inspect dns_net

docker container exec –it apache bash
ap-get update
apt-get install iputils-ping -y
ping nginx
docker container exec –it nginx bash
ping apache
```

Docker Network

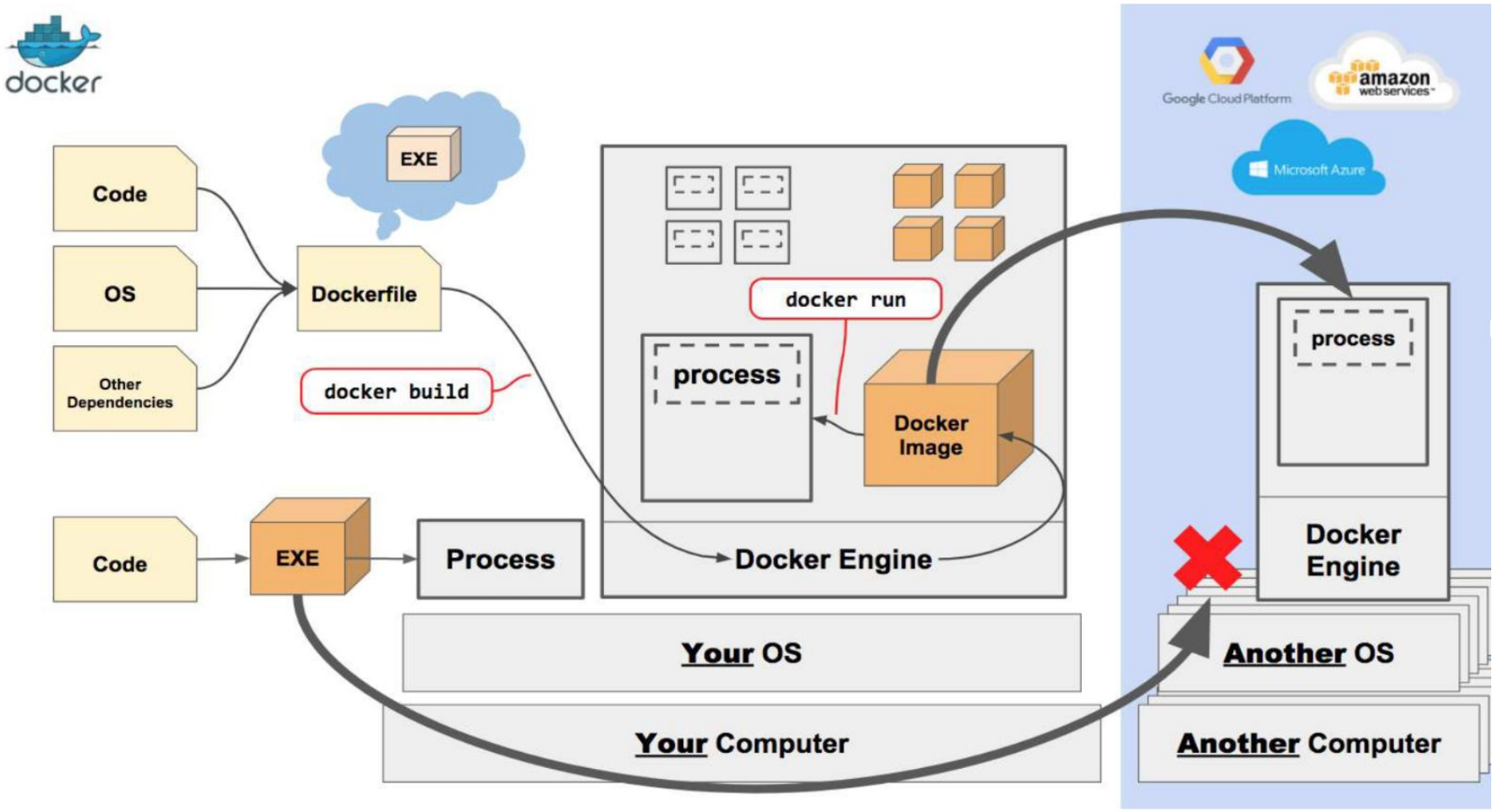
Docker Network

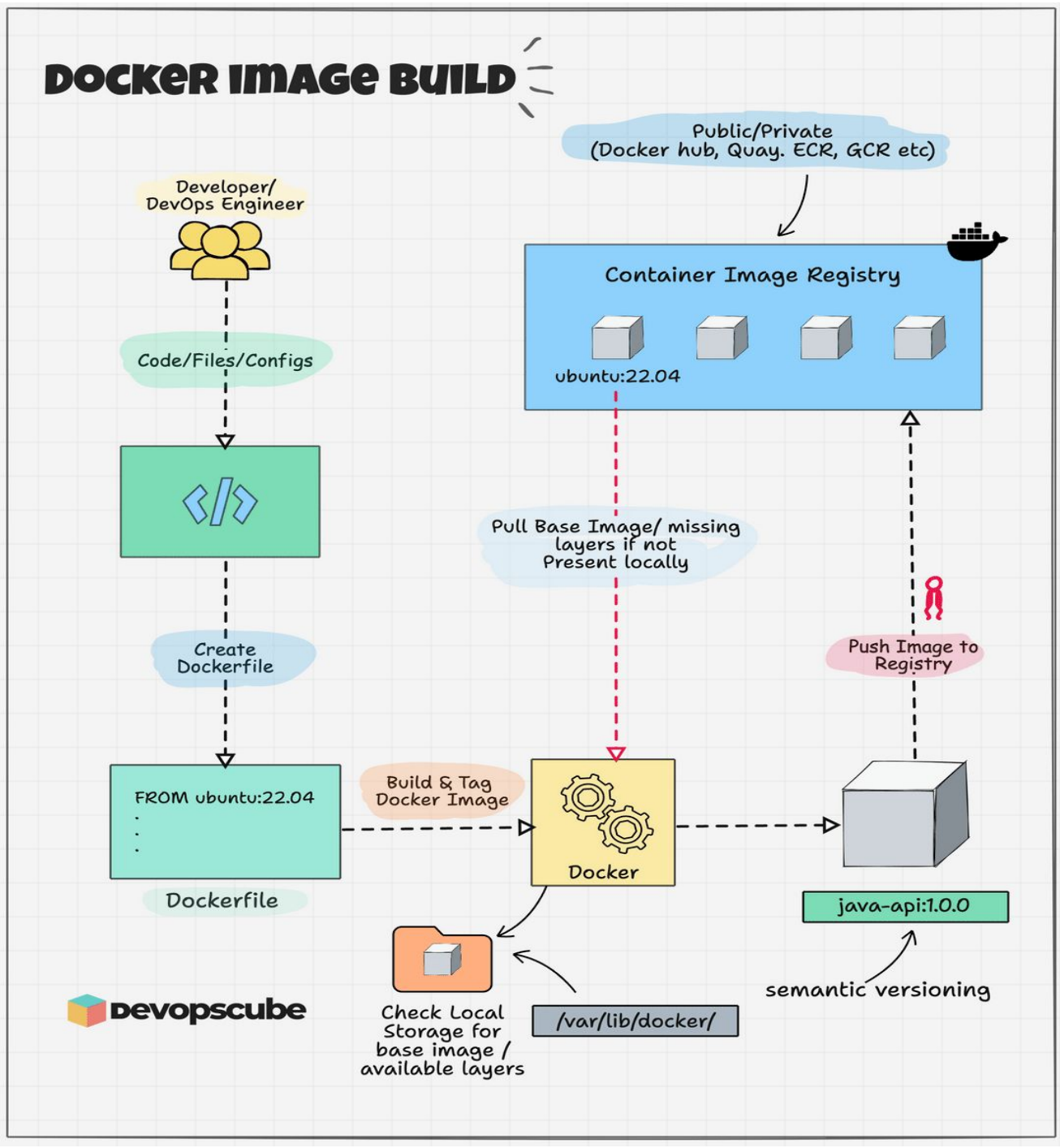


Working with Docker Images



Docker image build




DOCKER IMAGE BUILD
Public/Private
(Docker hub, Quay. ECR, GCR etc)
Developer/
DevOps Engineer
Container Image Registry
ubuntu:22.04
Code/Files/Configs
Pull Base Image/ missing
layers if not
Present locally
Create
Dockerfile
Push Image to
Registry
FROM ubuntu:22.04
Build & Tag
Docker Image
Docker
Dockerfile
java-api:1.0.0
devopscube
Check Local
Storage for
base image /
available layers
/var/lib/docker/
semantic versioning

Dockerfile

Table 1: Dockerfile Instructions (1)

Instruction	Explanation
FROM	Specifies the base image from a registry (Docker Hub, GCR, etc.).
RUN	Executes commands during image build.
ENV	Sets environment variables available during build and runtime. Use `**ARG**` for build-time only.
COPY	Copies local files/directories to the image.
EXPOSE	Specifies the port to expose for the container.
ADD	Similar to `**COPY**`, but can handle URLs and auto-extract tar files. Use `**COPY**` unless you need `**ADD**` features.
WORKDIR	Sets the working directory for subsequent instructions.

Dockerfile

Table 2: Dockerfile Instructions (2)

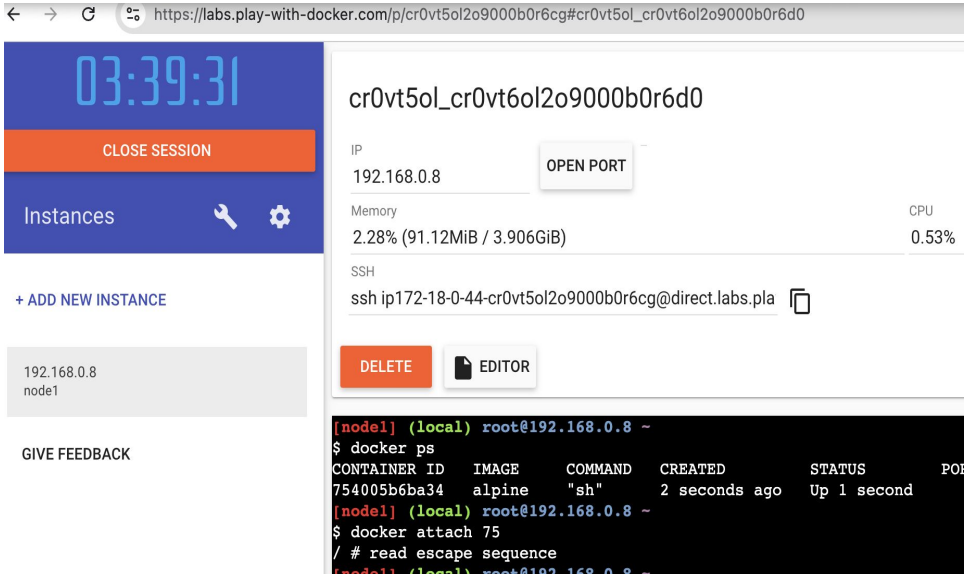
Instruction	Explanation
VOLUME	Creates or mounts a volume to the container.
USER	Sets the user name and UID for running the container.
LABEL	Specifies metadata for the Docker image.
ARG	Sets build-time variables (not available at runtime). Use `` `ENV` `` for runtime variables.
SHELL	Sets shell options and default shell for `` `RUN` ``, `` `CMD` ``, and `` `ENTRYPOINT` ``.
CMD	Executes a command in the running container; only one `` `CMD` `` is allowed. It can be overridden.
ENTRYPOINT	Specifies commands to execute when the container starts. Defaults to `` `/bin/sh -c` ``. Can be overridden.

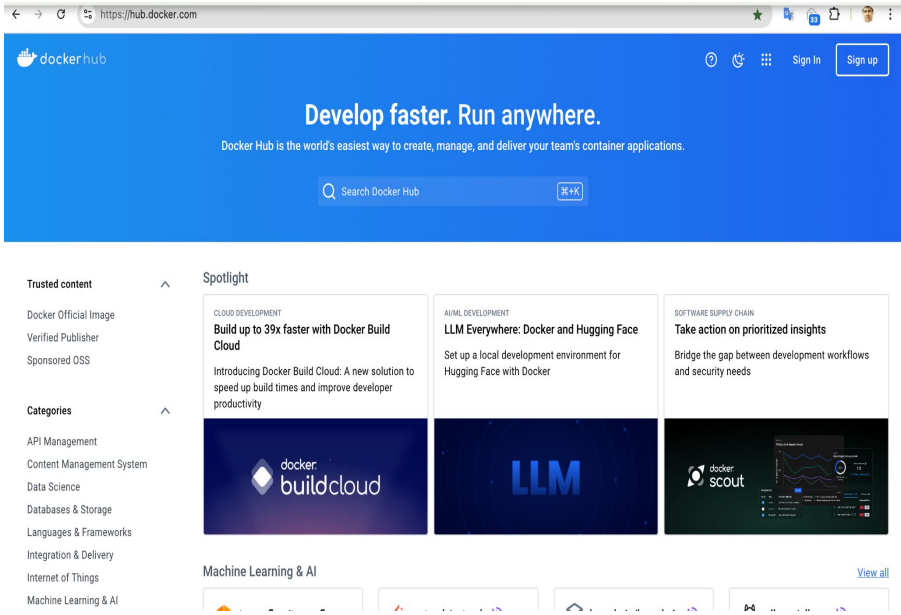
Docker Image build & Push it to DockerHub

Lab-1 How to build Your First Docker Image and Push it to DockerHub

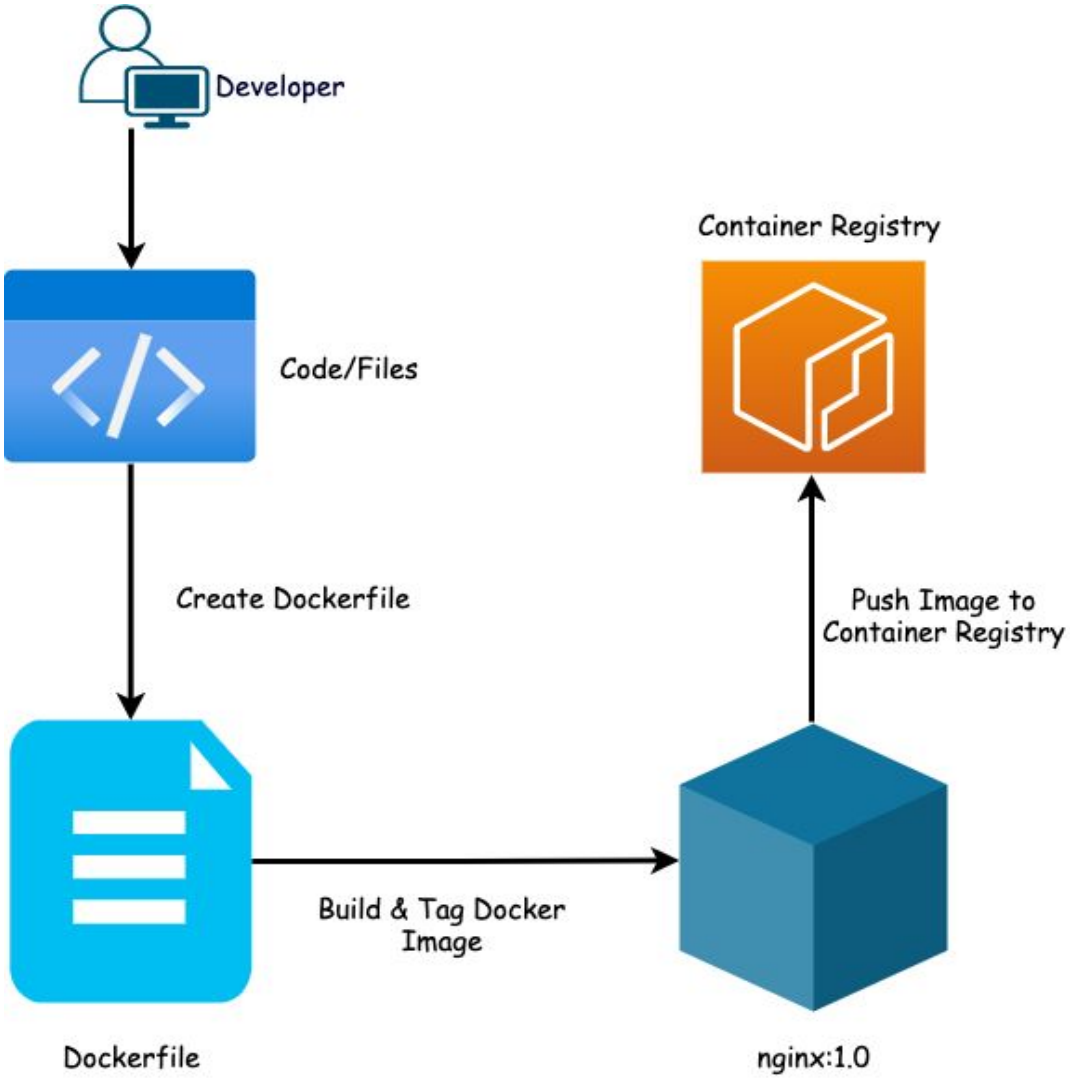
Pre-requisite

- Create an account with DockerHub
- Open Play with Docker Platform on your browser

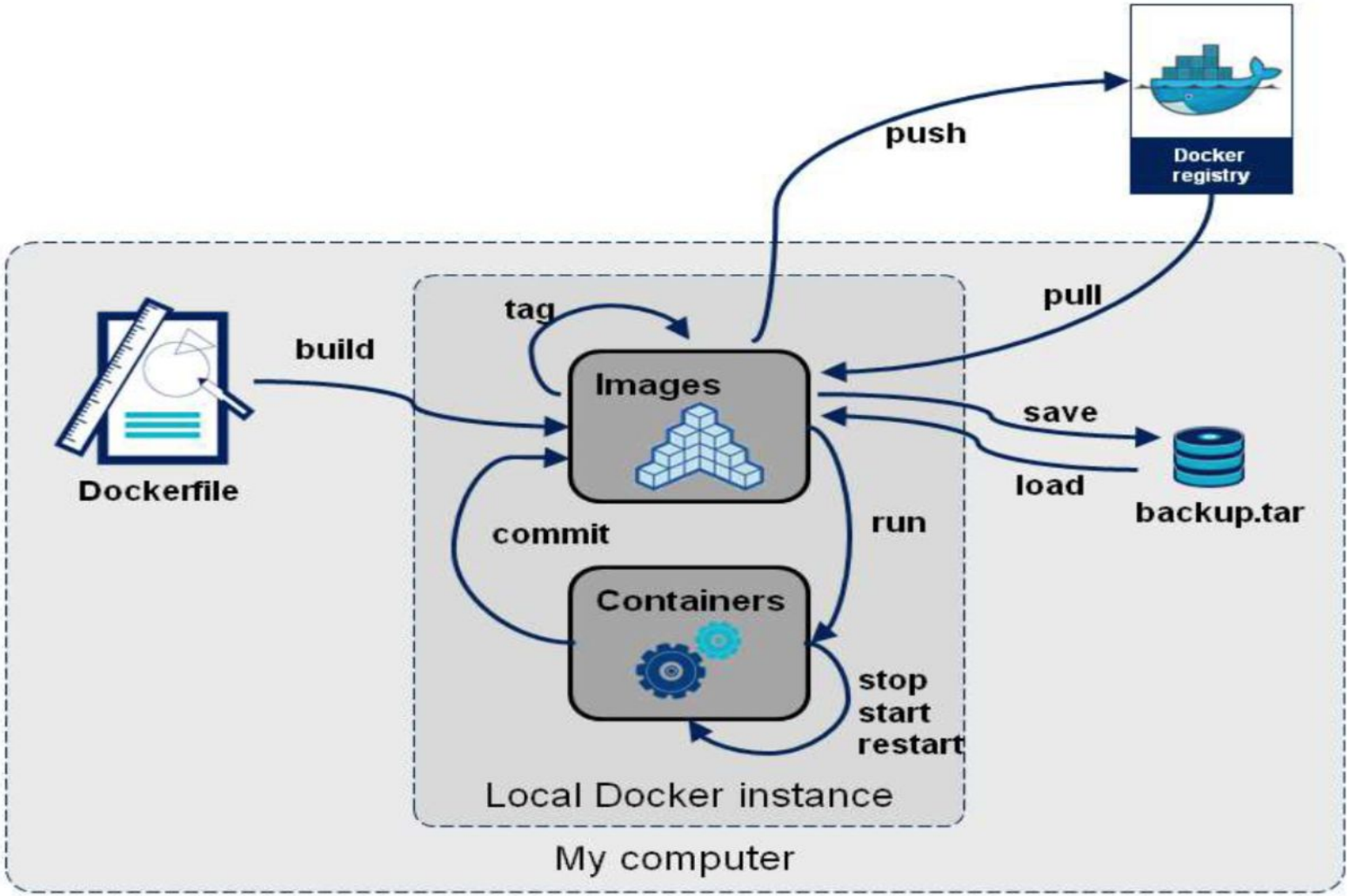




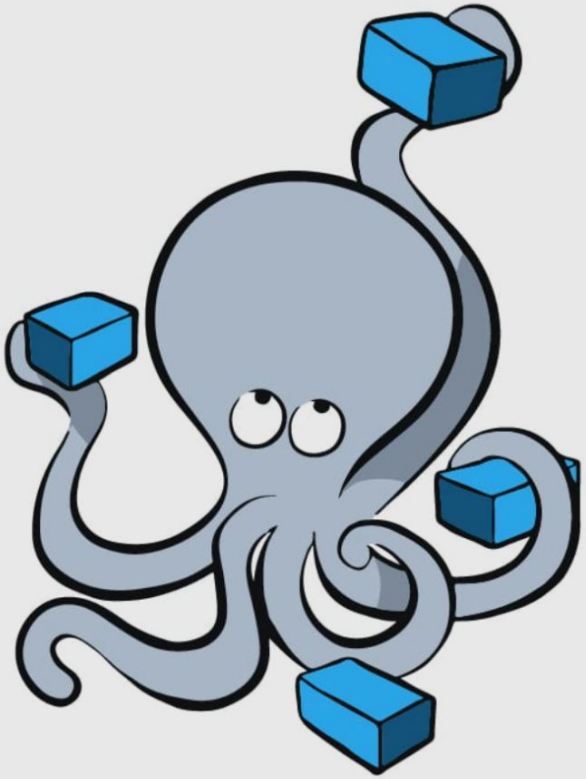
Lab-2 Build Docker Image Using Dockerfile



Lab-3 Docker images sharing



Docker Compose

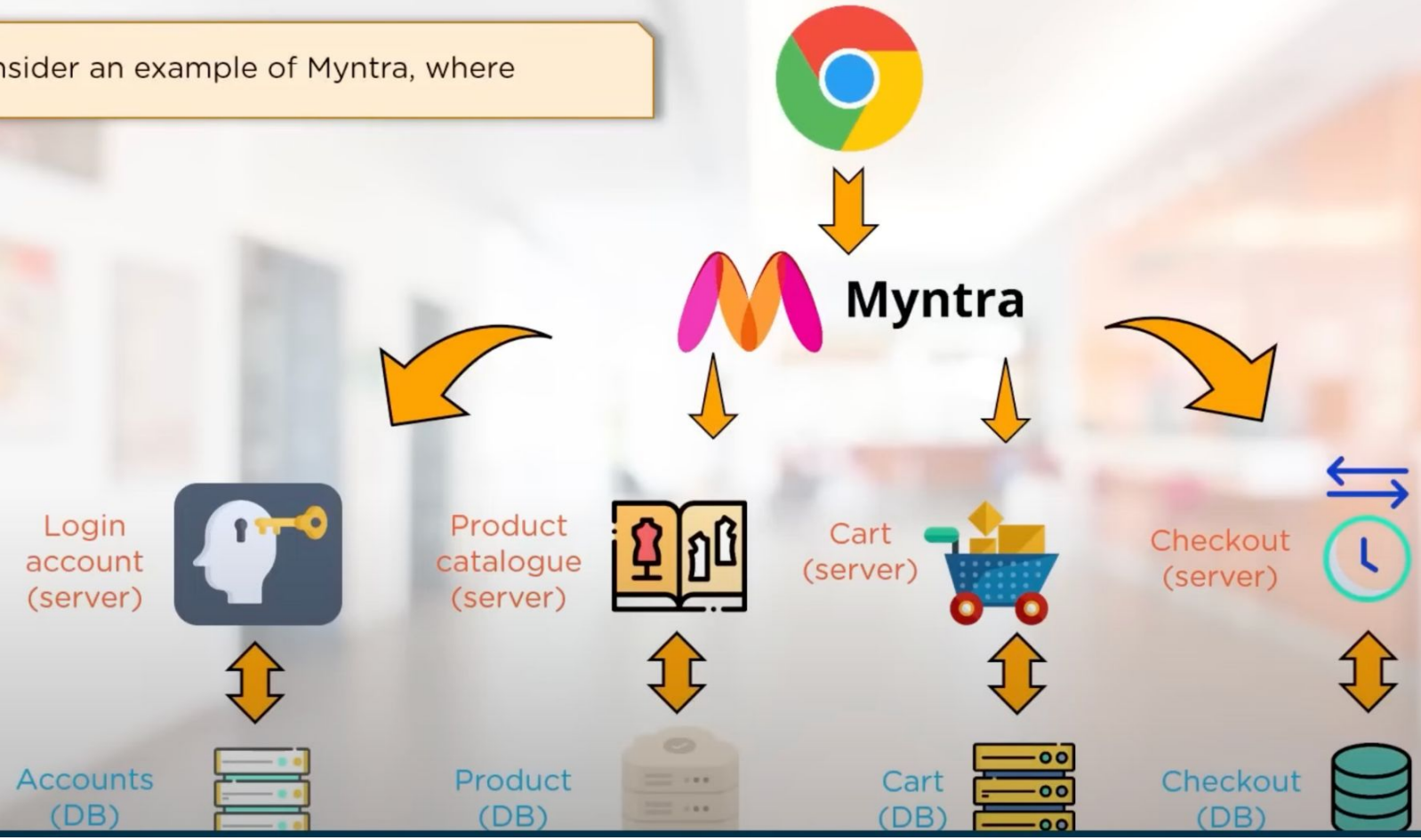


docker

Compose

What is Docker Compose?

Consider an example of Myntra, where



Docker Compose

Docker Compose : Docker Compose is used to run multiple containers as a single service. Compose provide relationship between multiple Containers.

➢ Example : User can Start MySQL and Tomcat Container with one YML file without starting each separately.

➢ Docker Compose an Three Step Process:

➢ Define your app's environment with a dockerfile so it can be reproduced anywhere.

➢ Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.

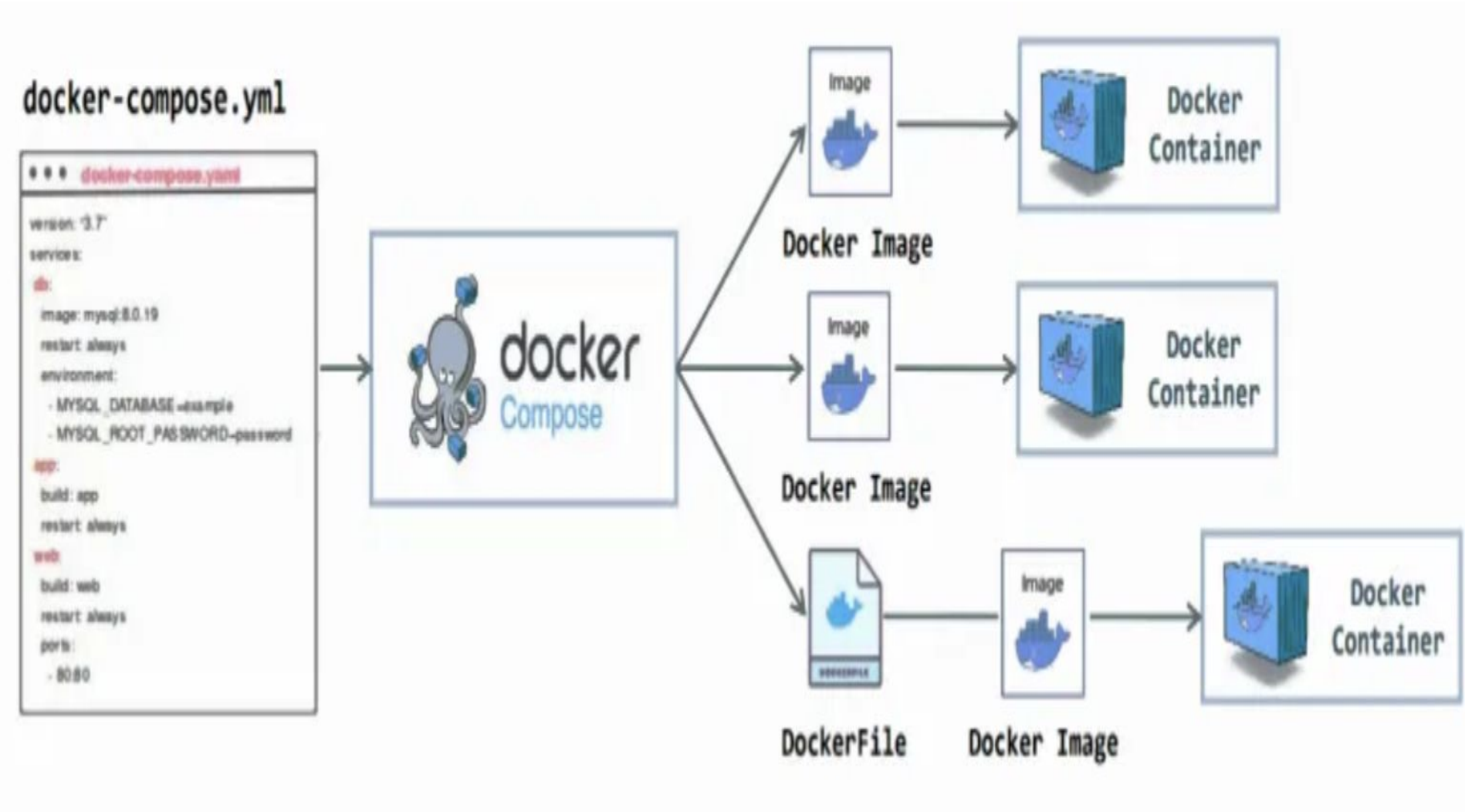
➢ Run docker-compose up and Compose starts and runs your entire app.

docker-compose.yml

Docker Compose : YML is formatted file in Docker compose which describe the Solution for:

- Containers
- Networks
- Volumes
- Container Relation

docker-compose.yml



Docker Compose Commands

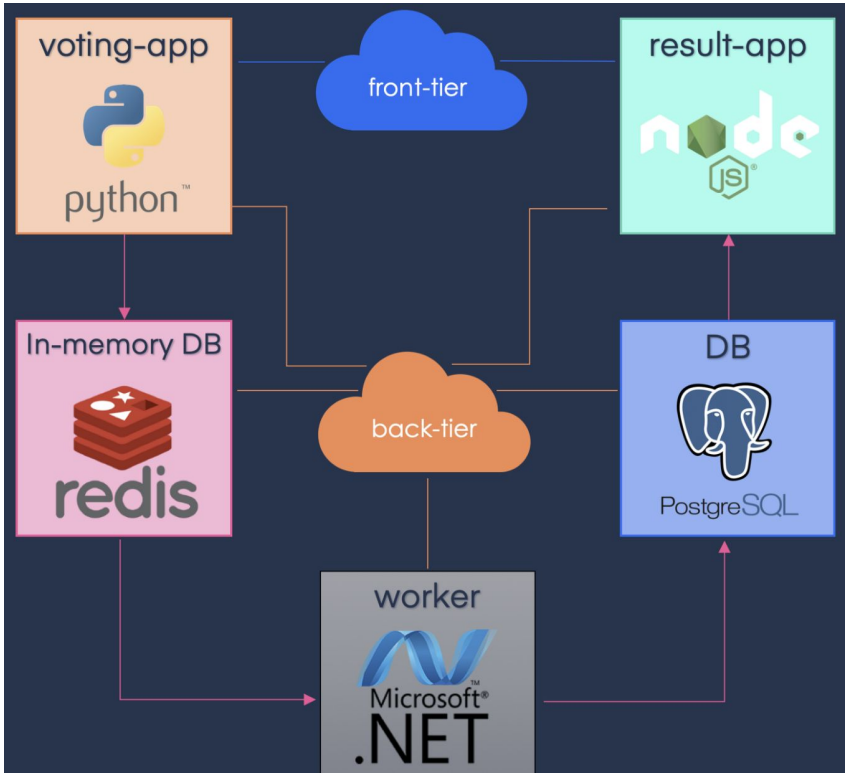
Command	Description
`docker-compose up`	Starts and runs the containers defined in the `docker-compose.yml` file. If the containers are not yet built, they will be built before starting.
`docker-compose down`	Stops and removes the containers, networks, and volumes created by `docker-compose up`.
`docker-compose build`	Builds or rebuilds the services defined in the `docker-compose.yml` file without starting them.
`docker-compose start`	Starts existing containers without creating or recreating them.
`docker-compose stop`	Stops the running containers without removing them.
`docker-compose restart`	Restarts the containers.
`docker-compose ps`	Lists all running containers defined in the `docker-compose.yml` file.
`docker-compose logs`	Displays the logs of all containers. You can use `-f` to follow the logs in real-time or `--tail` to show only the last few lines.

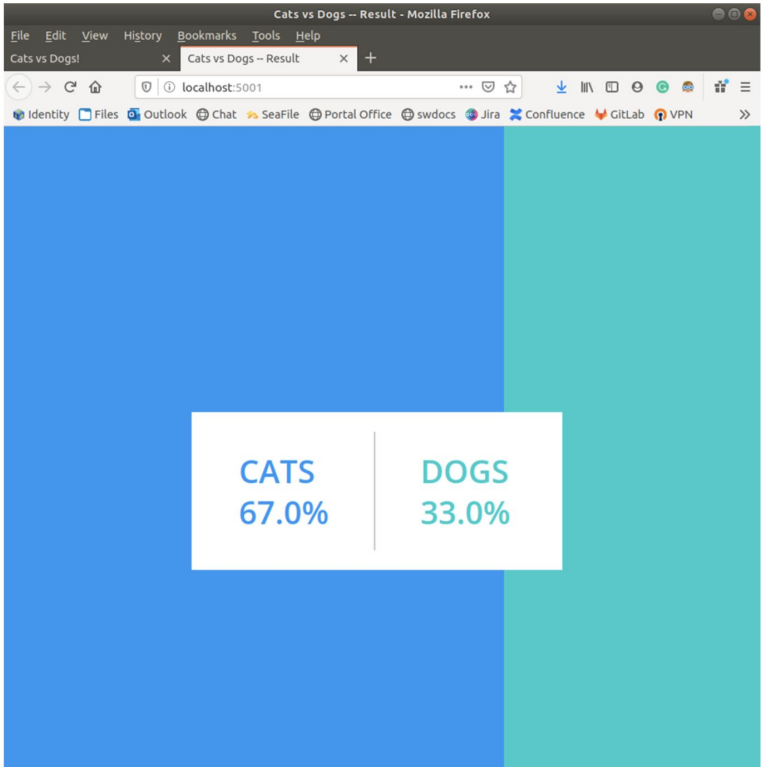
Docker Compose Commands

Command	Description
`docker-compose exec <service>`	Runs a command in a running container, for example, `docker-compose exec web bash` to open a shell in the `web` container.
`docker-compose run <service>`	Runs a one-time command in a new container instance of a specified service, useful for running tasks like migrations or tests.
`docker-compose pull`	Pulls the latest version of the images defined in the `docker-compose.yml` file from a registry.
`docker-compose rm`	Removes stopped service containers. You can add `-f` to force removal without confirmation.
`docker-compose config`	Validates and displays the current Compose file configuration. Useful for checking for syntax errors or verifying your setup.
`docker-compose version`	Displays the current version of Docker Compose.
`docker-compose scale`	Sets the number of container instances for a service. This feature is deprecated in favor of `docker-compose up --scale <service>=<num>` in newer versions.
`docker-compose up --build`	Builds the images before starting the containers. Useful when you want to ensure the latest code changes are included.
`docker-compose port <service> <port>`	Displays the mapped port of a container, showing which port on the host is mapped to a specific port inside the container.

Docker compose

voting






```
git clone https://github.com/dockersamples/example-voting-app
cd example-voting-app/
docker-compose up
```

Docker examples

- https://github.com/deviantony/docker-elk
- https://docs.docker.com/compose/wordpress/
- https://docs.docker.com/compose/gettingstarted/
- https://dockerlabs.collabnix.com/intermediate/workshop/DockerCompose/Create_first_docker-compose_file_with_ngnix_and_mysql.html
- https://cheatography.com/gauravpandey44/cheat-sheets/docker-compose/
- https://gist.github.com/jonlabelle/bd667a97666ecda7bbc4f1cc9446d43a
- https://docs.docker.com/compose/compose-file/#compose-file-structure-and-examples
- https://github.com/docker/awesome-compose
- https://github.com/docker/awesome-compose/tree/master/portainer
- https://github.com/docker/awesome-compose/tree/master/django
- https://github.com/docker/awesome-compose/tree/master/nginx-nodejs-redis