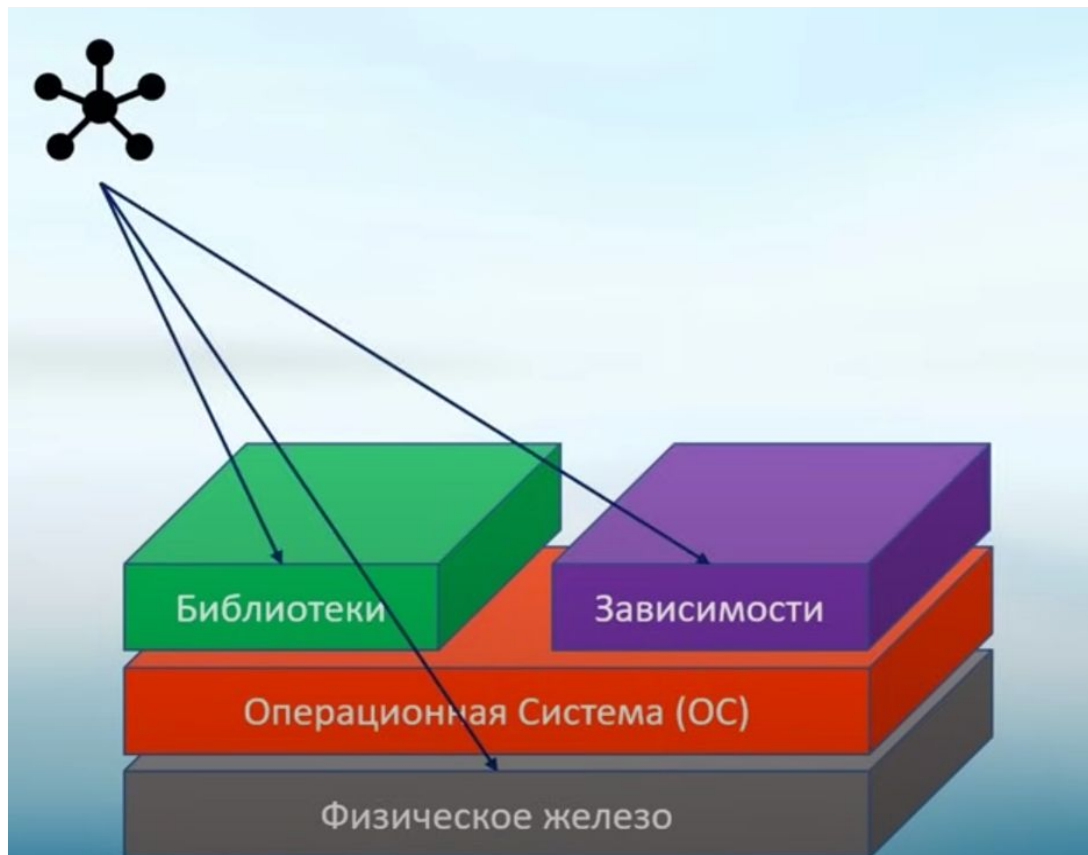


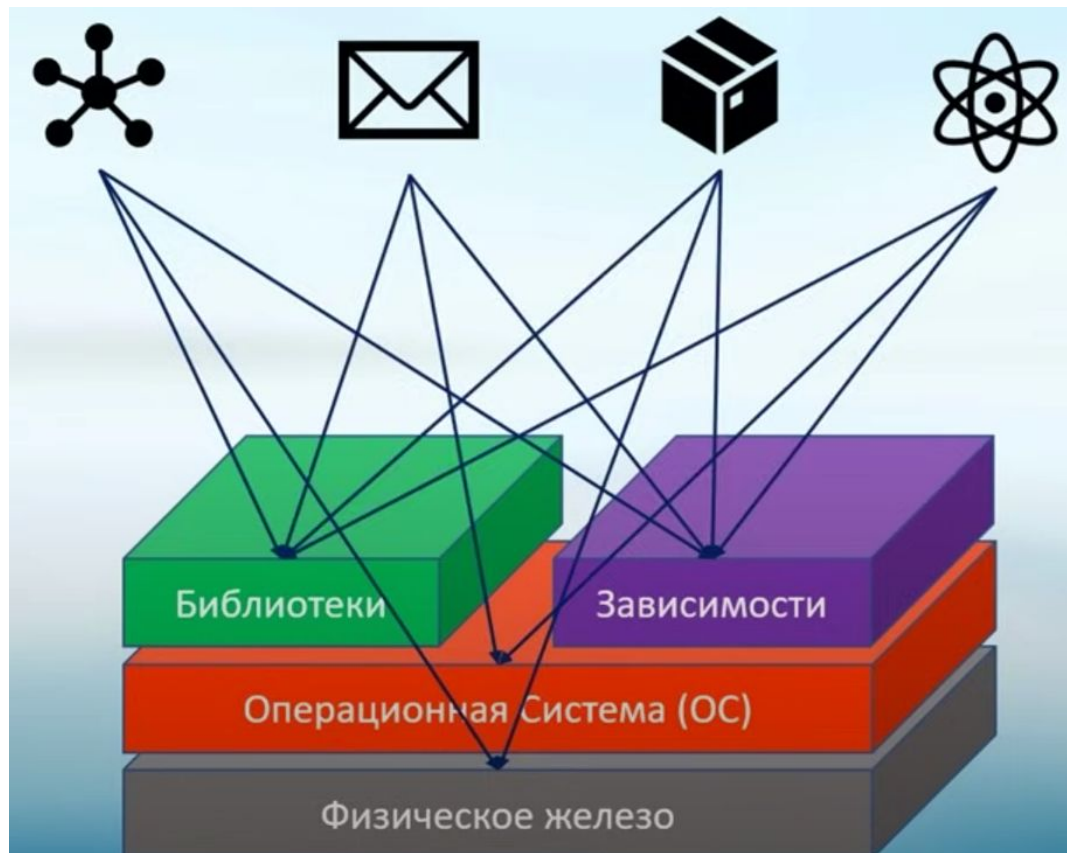
# Docker

Выполнил: Гурин Артур 11-307

# Проблема



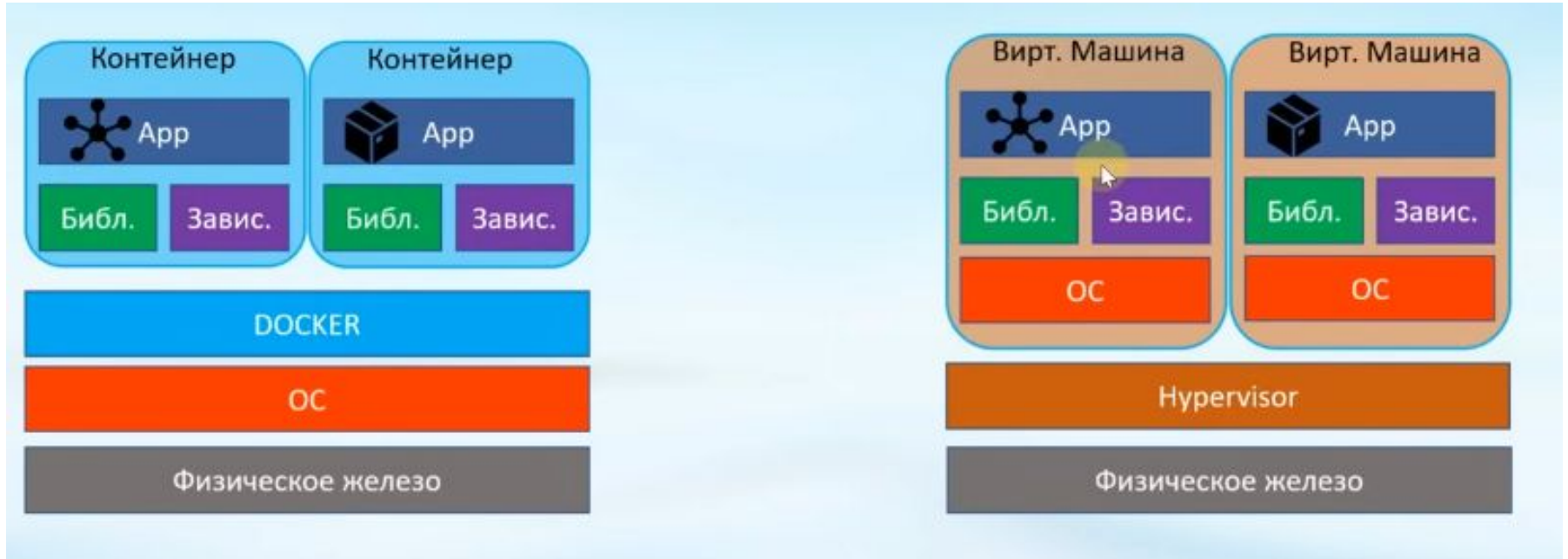
А что если много приложений?



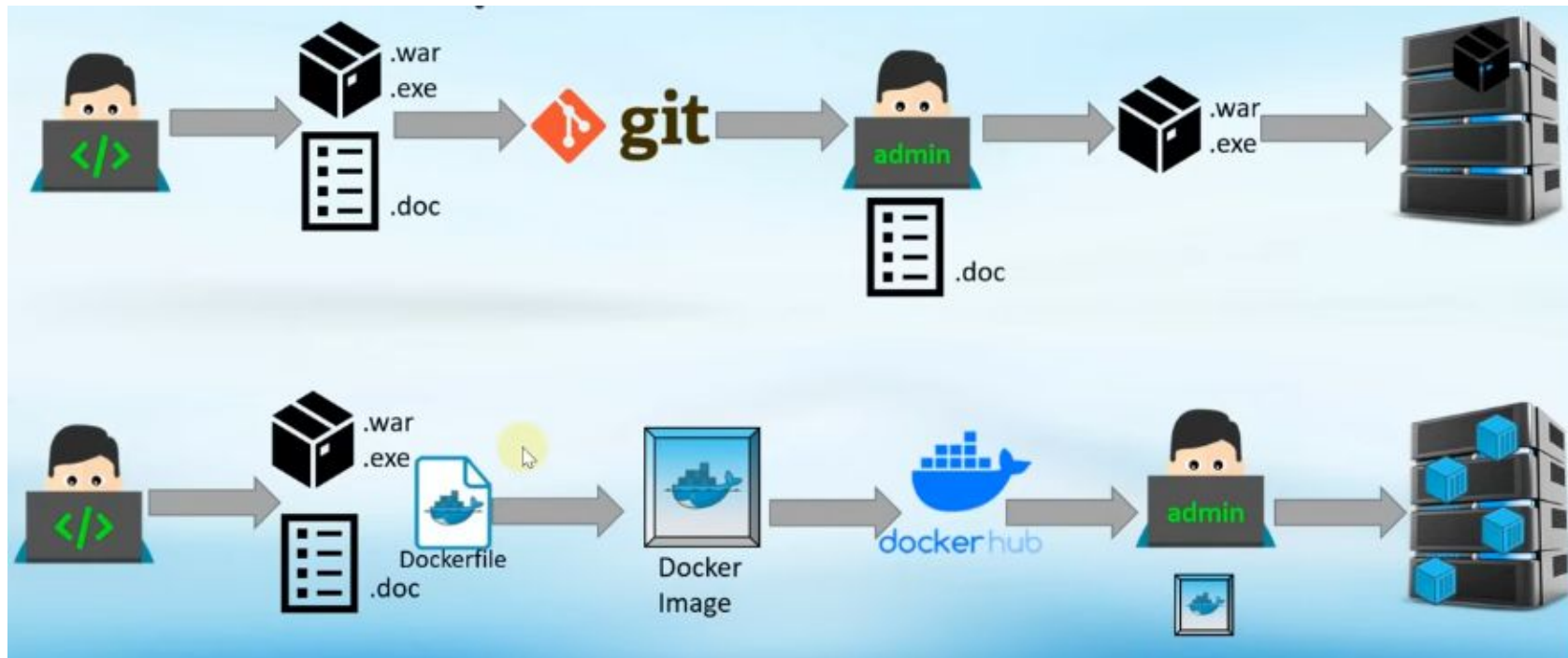
# А как с докером?



# Докер VS Виртуальная машина



В чем еще плюс?



# Терминология

**Images (образы)** - Схемы нашего приложения, которые являются **основой контейнеров**. Можно привести аналогию с iso образом винды например:)

**Containers (контейнеры)** - Создаются на основе образа и запускают само приложение.

**Docker Hub - Регистр** Докер-образов. Грубо говоря, архив всех доступных образов. Если нужно, то можно содержать собственный регистр и использовать его для получения образов.

<https://hub.docker.com/explore/>

# Регистр, образ, контейнер

Сервер

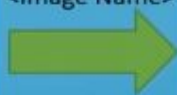
`docker pull <Image Name>`

Local Docker Registry

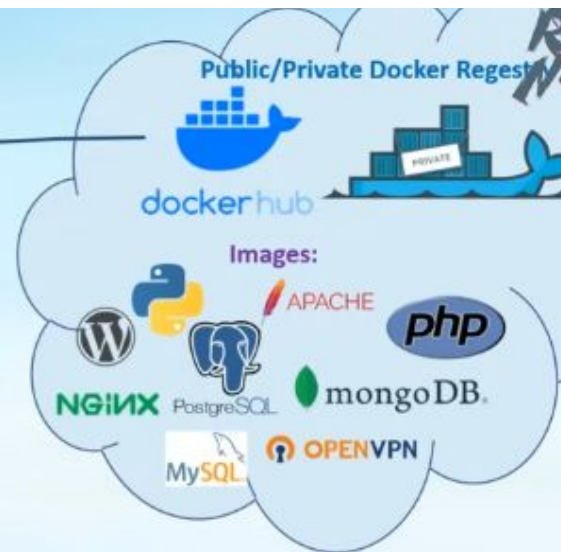
Images:



`docker run <Image Name>`



Docker Containers





# Практика!

Перед началом надо установить докер на свой пк ([тык](#))

Попробуем сделать свой первый контейнер

`docker run hello-world` - из image делает контейнер.

**(В случае если данного image не было локально, он берет его из докер хаба)**

`docker ps` - проверяет запущенные контейнеры

`docker ps -a` - все ранее запущенные контейнеры

`docker rm <начало id контейнера>` - удаление контейнера из списка

`docker images` - все images

`docker rmi <image id>` - удаляем image

# Попробуем Ubuntu

`docker pull ubuntu` - скачать image с докер хаб

(если запустить image ubuntu и сделать из него контейнер, то **он запустится и сразу остановится**, но мы можем передать в run команду sleep, который указывается сколько нужно работать контейнеру)

**`docker run <image name> sleep <seconds>`**

`docker run -d <image name> sleep <seconds>` + консоль доступна

`docker start <container id>` - заново запустить контейнер, не создавая его заново из image.

`docker pause <container id>` - ставит на паузу

`docker unpause <container id>` - продолжить выполнение

`docker stop <container id>` - остановка контейнера

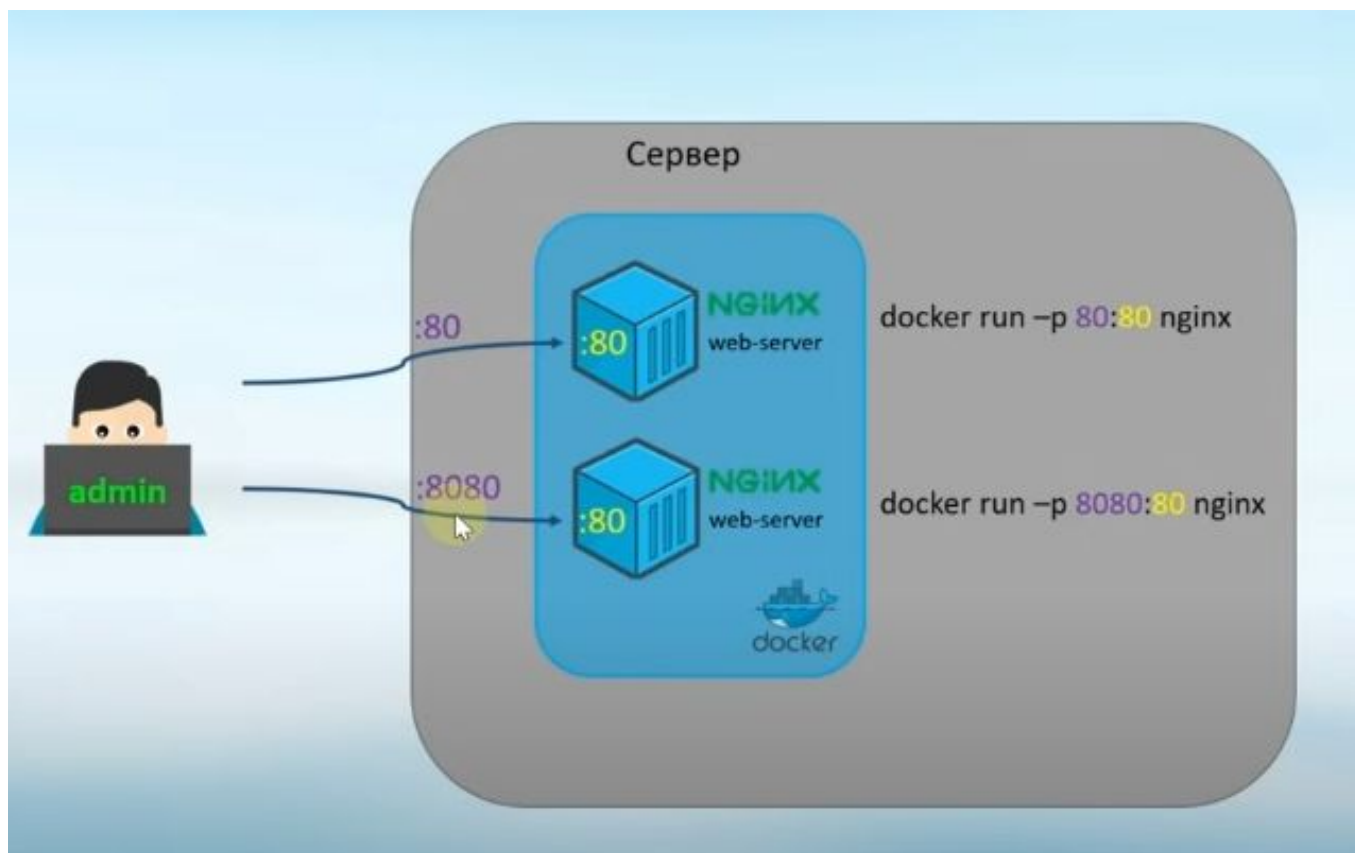
`docker kill <container id>` - убивает процесс

`docker inspect <container id>` - показана вся информация к контейнеру

`docker stats <container id>` - память, нагрузка на процессор, сеть и тд

# Порты

Люди  
подключаются к  
серверу по порту  
**80**, а сервер  
перекидывает  
запросы в  
соответствующие  
контейнеры по  
указанному порту



`docker run -p 8080:80 <container name>`

# Работа с переменными окружения

Добавить свою переменную окружения:

```
docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=<my-secret-pw> -d mysql:tag
```

Исследование переменных окружения:

```
docker exec -it <container id> /bin/bash
```

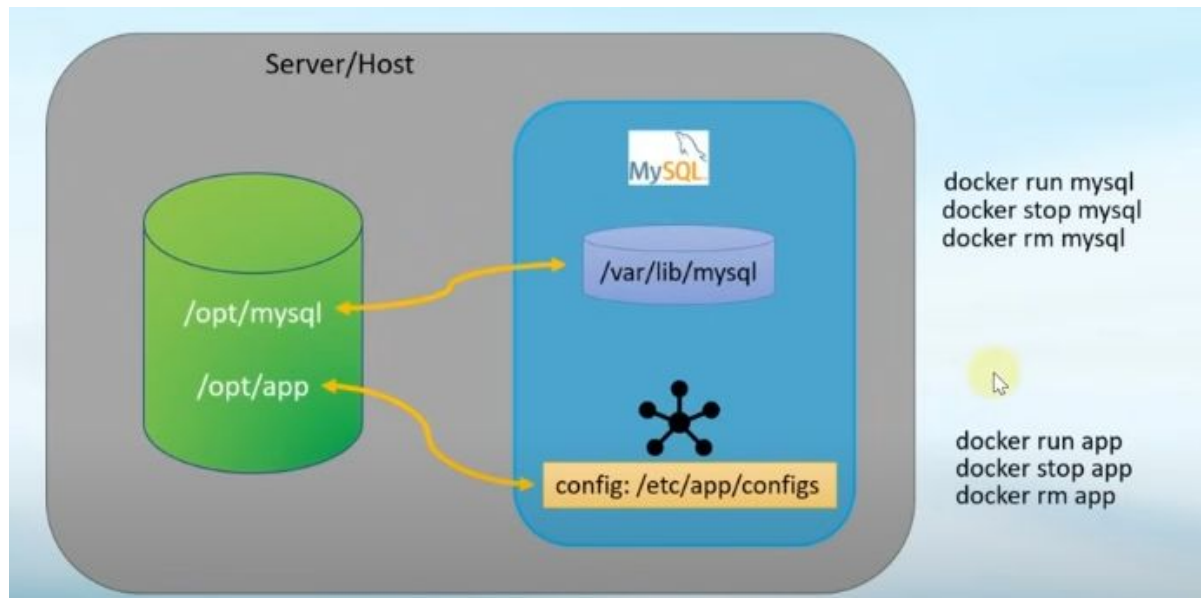
и ввести `env`

Обычно они указаны в документации.

**Переменные окружения** — это набор значений, которые определяют поведение и настройки операционной системы, а также других программ, работающих в этой среде

# Постоянные переменные

После команды `kill` удаляются все переменные окружения вместе с контейнером. Но что если нам нужно где то постоянно хранить переменные из контейнера. Для этого используются Docker **Volumes** и **Mounting**

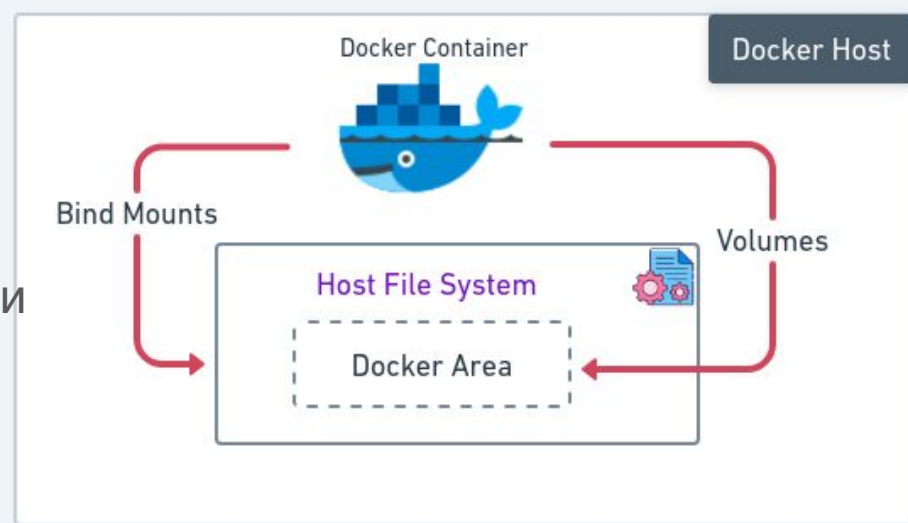


# Docker Volume

В Docker **том** или **volume** — это механизм для хранения и управления данными, которые используются внутри контейнеров. Том можно рассматривать как отдельную файловую систему, которая существует вне контейнера и может быть подключена к одному или нескольким контейнерам.

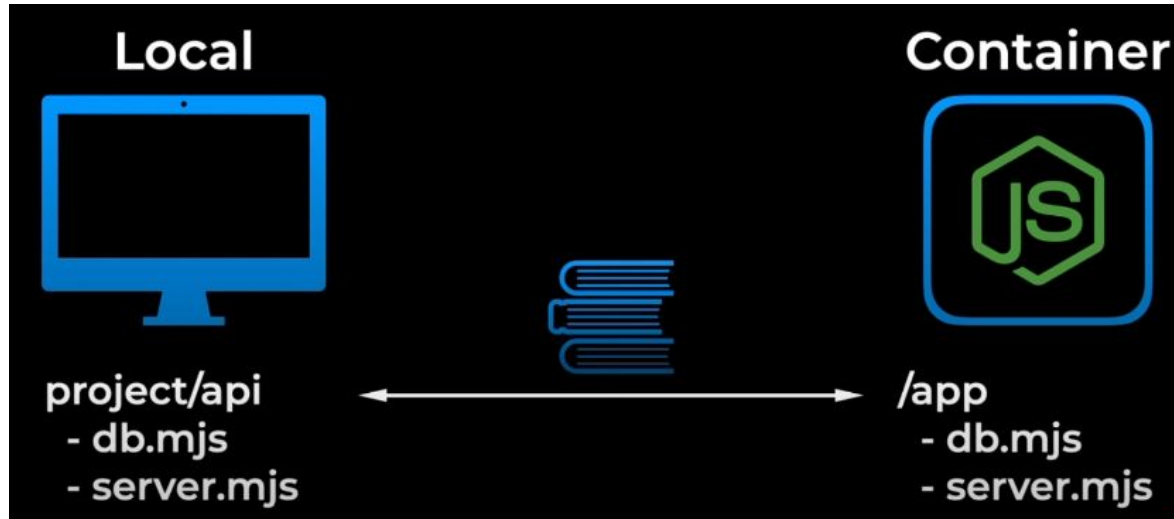
Также существует **Bind Mount**, он подразумевает подключение **ЛЮБОЙ** директории на хосте.

**Volumes** создает сам докер и сам ими управляет, а при **bind mounting** мы сами управляем расположением данных.



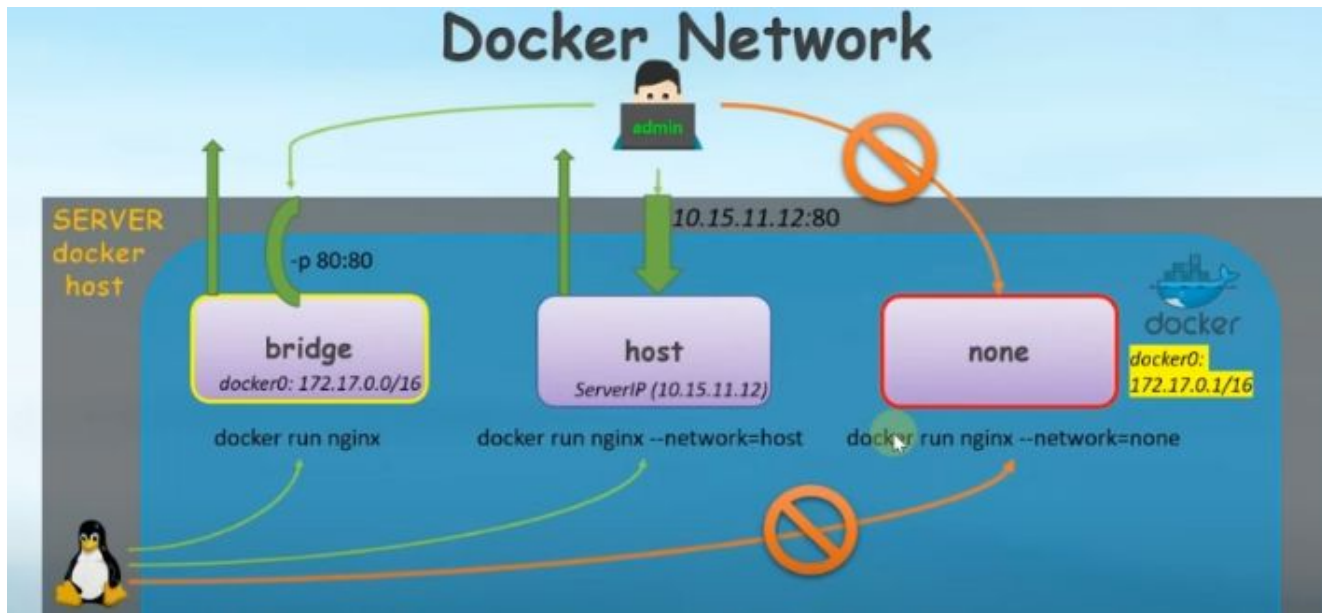
# Volumes и Bind Mount

Проще говоря: происходит сопоставление папок на хосте и папок внутри контейнера. Если что-то изменить в папке на хосте, то это же изменение отразится и в папке внутри контейнера. Этот подход позволяет вносить изменения в проект, не создавая каждый раз новый image и новый контейнер соответственно. Сам образ не меняется!



# Сети в докере

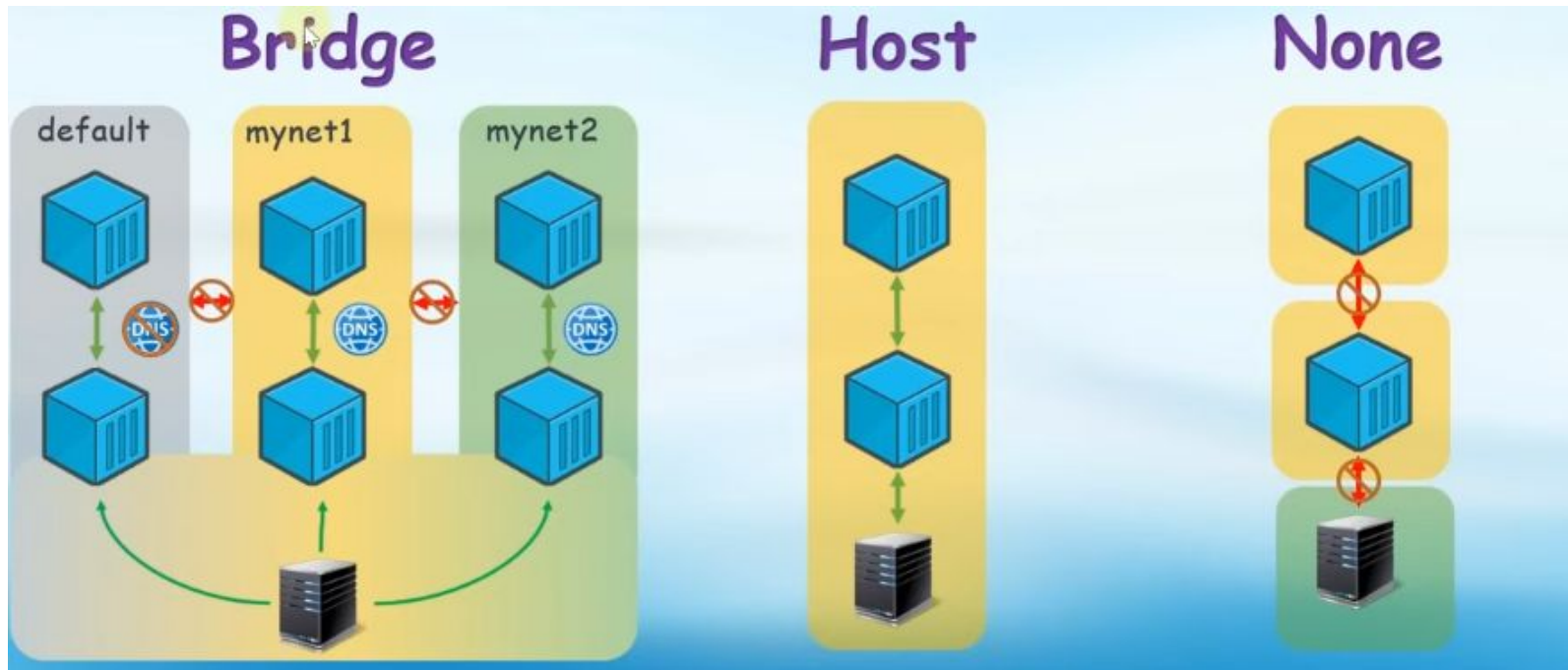
Сетевой интерфейс в Docker — это механизм, который позволяет контейнерам взаимодействовать между собой, а также с внешним миром (например с другими сетями).





# Типы сетей

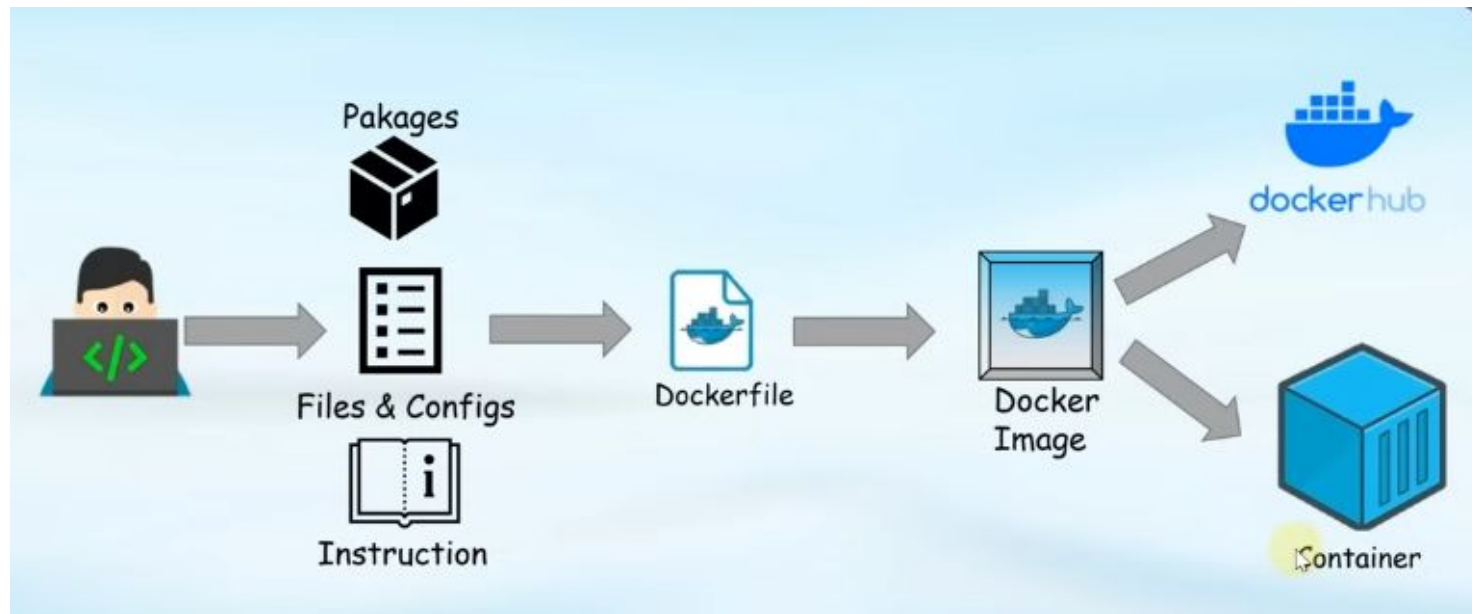
1. **Bridge** - тип сети по умолчанию, то есть при запуске контейнера по умолчанию, он попадает именно в данный тип сети. Они имеют доступ наружу (к интернету) и локально. Чтобы подключиться к контейнеру, который работает в сети bridge, нужно как бы создать мостик, **как до этого указывали порты -p 80:80.**
2. **Host.** Контейнеры, создаваемые в данном типе сети, получают айпи адрес именно хоста. Они также могут выходить в сеть  
**`docker run <image name> --network=host`**
3. **None.** **`docker run <image name> --network=none`** К нему ни локально, ни по интернету. Можем просто посмотреть логи и просто запустить программы



**docker network create --driver  
bridge NAME  
docker run --net NAME <image  
name>**

# Создание своего image

Чтобы создать свой docker **image (образ)** нам надо написать **dockerfile** для своего приложения. В нем (в dockerfile) хранится вся информация о пакетах, конфигурациях, и т.д. для приложения



# Докер файл

- **Dockerfile** — это текстовый файл с инструкциями, которые Docker использует для создания образа (image) приложения. В нем указываются команды для установки зависимостей, копирования файлов, выполнения команд и настройки среды.

## Описание:

- Рабочие директории
- Файлы необходимые для работы
- Команды для работы с файлами
- Указание переменных
- Порты (только информационный контекст)
- Описание команд при запуске контейнера

# Как его написать?

`docker build .` - создание image

`docker tag <image id> <image name>:<tag>`

`docker image inspect <image name>` - посмотреть все команды в image

`docker build -t <image name> .` - создастся image с именем

## Builder main commands

command	description
<code>FROM image scratch</code>	base image for the build
<code>MAINTAINER email</code>	name of the mainainer (metadata)
<code>COPY path dst</code>	copy <i>path</i> from the context into the container at location <i>dst</i>
<code>ADD src dst</code>	same as <code>COPY</code> but untar archives and accepts http urls
<code>RUN args...</code>	run an arbitrary command inside the container
<code>USER name</code>	set the default username
<code>WORKDIR path</code>	set the default working directory
<code>CMD args...</code>	set the default command
<code>ENV name value</code>	set an environment variable



Dockerfile

- Базовый образ (image)
- Описания образа
- Команды
- Рабочие директории
- Файлы
- Работа с файлами
- Указание переменных
- Порты
- Описание команд при запуске контейнера

```
FROM ubuntu:22.04
LABEL autor=RomNero
RUN apt-get update
RUN apt-get install nginx -y
WORKDIR /var/www/html/
COPY files2/index.html .
COPY files2/script.sh /opt/script.sh
RUN chmod +x /opt/script.sh
ENV OWNER="RomNero"
ENV TYPE=demo
EXPOSE 80
ENTRYPOINT ["echo"]
CMD ["Hello my FIRST Docker"]
```

# Пример dockerfile для asp.net приложения

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
USER $APP_UID
WORKDIR /app
EXPOSE 8080
EXPOSE 8081
```

```
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
COPY ["API/API.csproj", "API/"]
RUN dotnet restore "API/API.csproj"
COPY . .
WORKDIR "/src/API"
RUN dotnet build "API.csproj" -c $BUILD_CONFIGURATION -o /app/build
```

```
FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "API.csproj" -c $BUILD_CONFIGURATION -o /app/publish /p:UseAppHost=false
```

```
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "API.dll"]
```

# Docker Compose

- Используется для управления одним или несколькими контейнерами
- Инструкции по запуску контейнера(ов)
- Автоматизация запуска контейнера
- Описывается в .yaml файле

```
docker run -d nginx
```

```
docker run -p 80:80 -p 443:443 -d nginx:stable
```

```
docker run --name mynginx -p 80:80 -p 443:443 -d nginx:stable
```

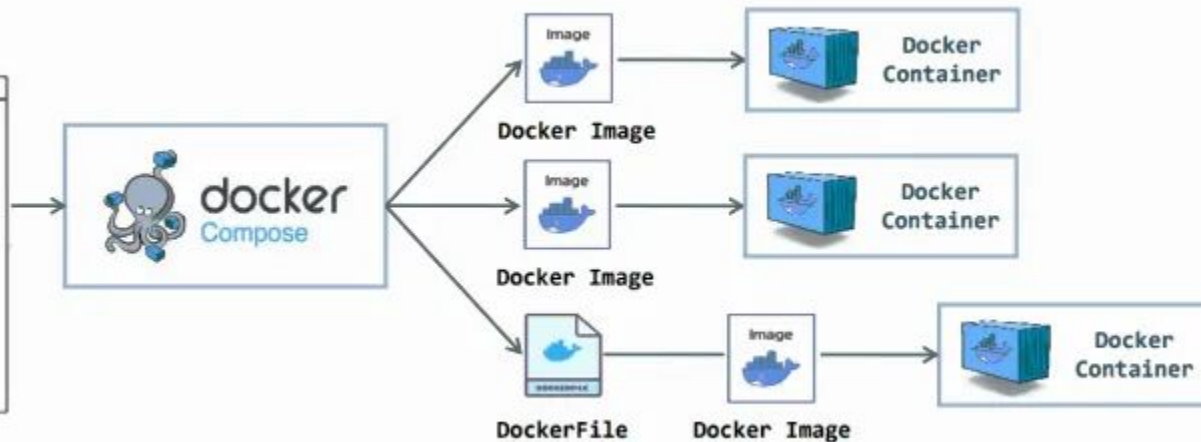
```
docker run --name mynginx -v /opt/web/html:/var/www/html -v /opt/web/pics:/var/www/pictures -p 80:80 -p 443:443 -d  
nginx:stable
```

```
docker run --name mynginx -v /opt/web/html:/var/www/html -v /opt/web/pics:/var/www/pictures -e
```

```
NGINX_HOST=web.romnero.de -e NGINX_PORT=80 -p 80:80 -p 443:443 -d nginx:stable
```

## docker-compose.yml

```
version: "3.7"
services:
  db:
    image: mysql:5.0.19
    restart: always
    environment:
      - MYSQL_DATABASE=example
      - MYSQL_ROOT_PASSWORD=password
  app:
    build: app
    restart: always
  web:
    build: web
    restart: always
    ports:
      - 80:80
```



✓ DockerAspNetExample · 1 project

✓ Solution Items

docker-compose.yml

✓ API

> Dependencies

> Properties

API API.http

appsettings.json

appsettings.Development.json

> Dockerfile

Program.cs

```
services:
  api:
    image: api
    build:
      context: .
      dockerfile: API/Dockerfile
```