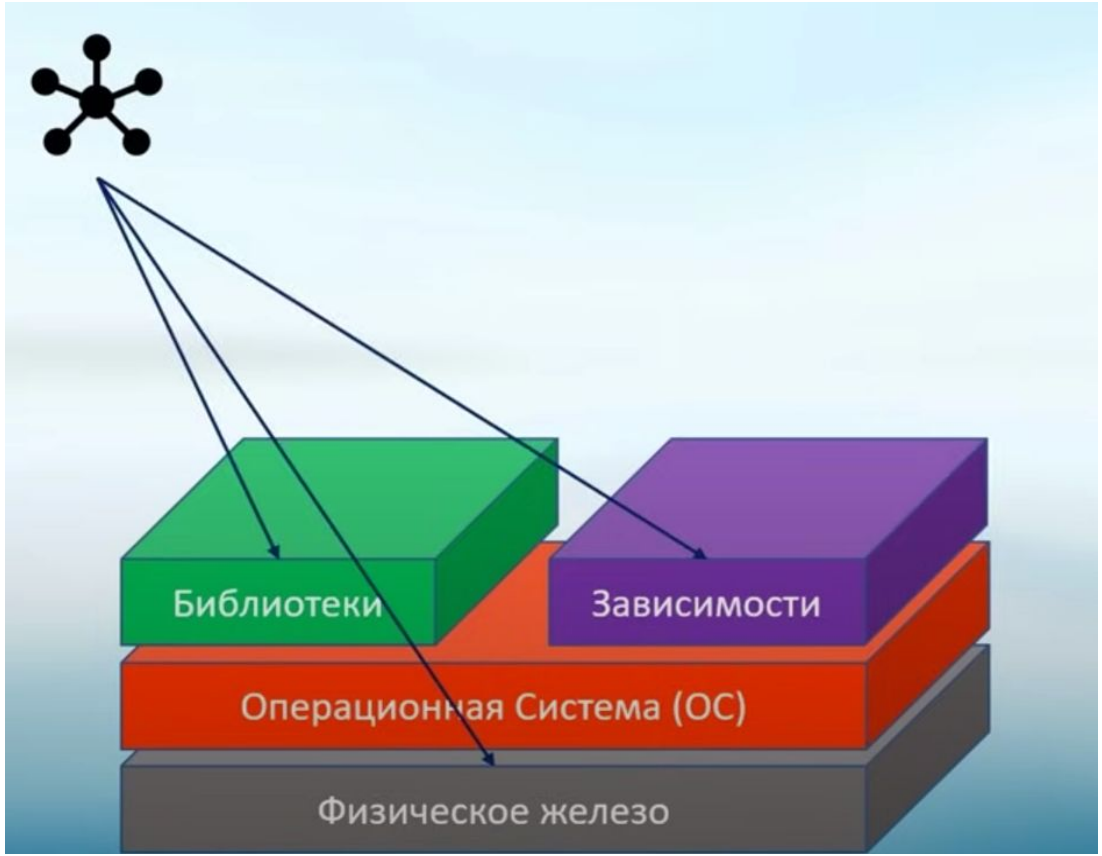
Docker

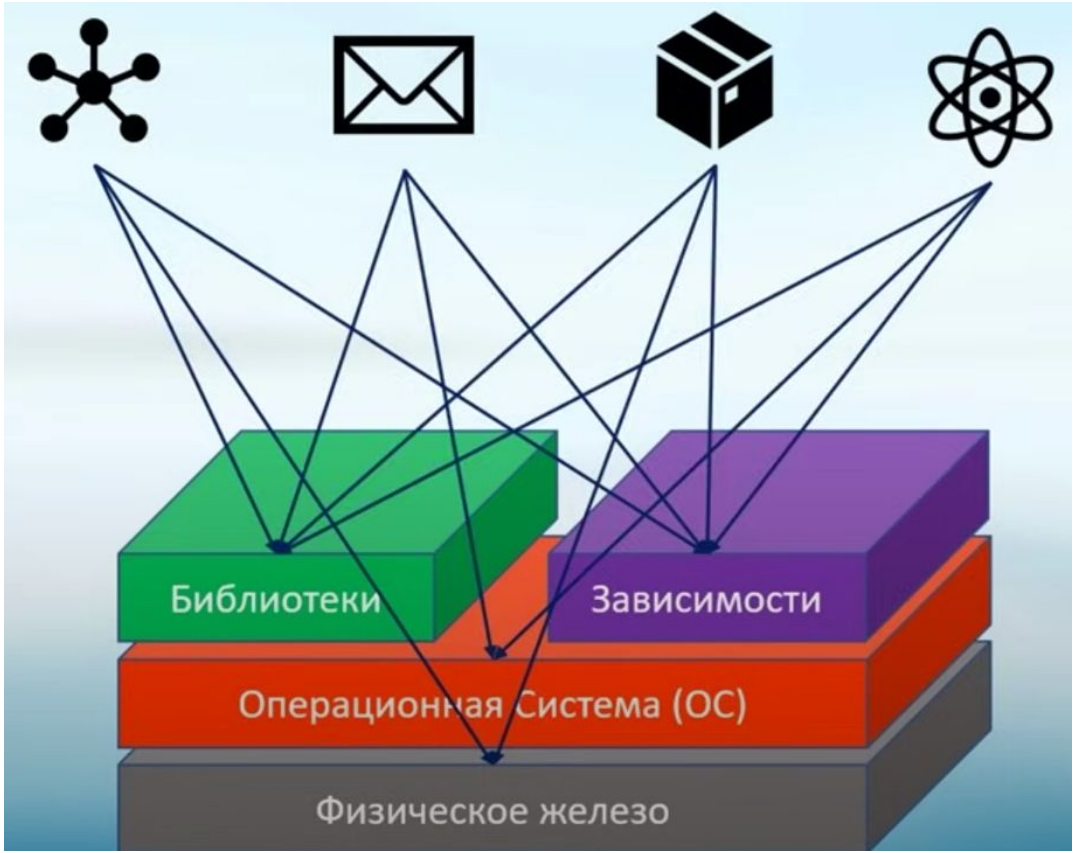
Выполнил: Гурин Артур 11-307

репозиторий
с кодом и
командами

Проблема

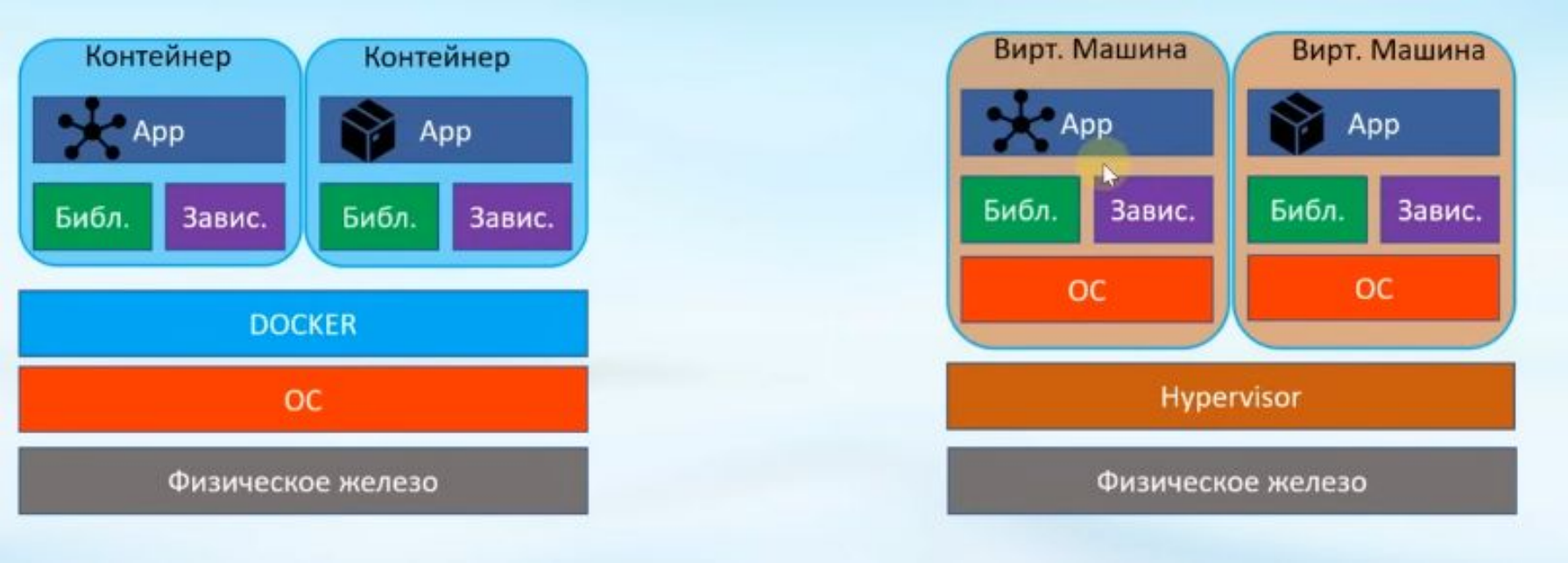

Библиотеки
Зависимости
Операционная Система (ОС)
Физическое железо

А что если много приложений?

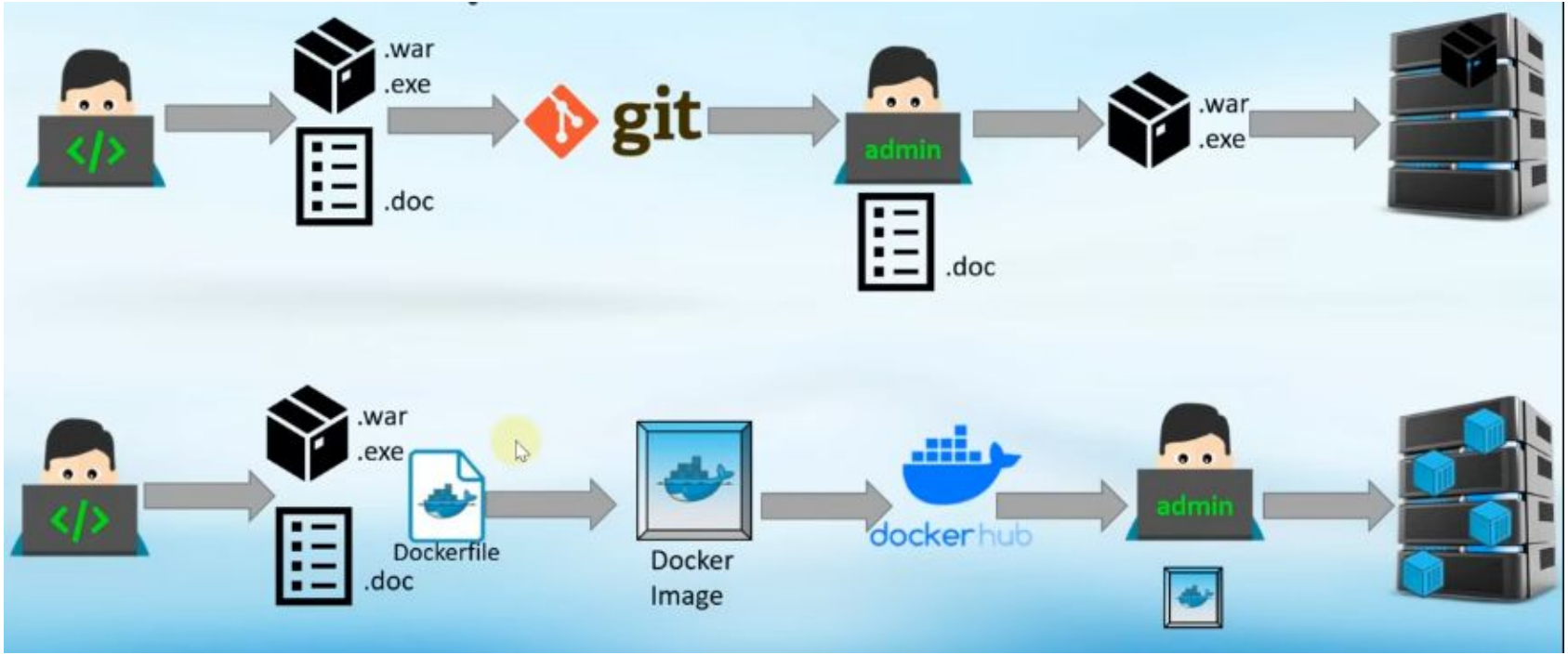


А как с докером?

Докер VS Виртуальная машина



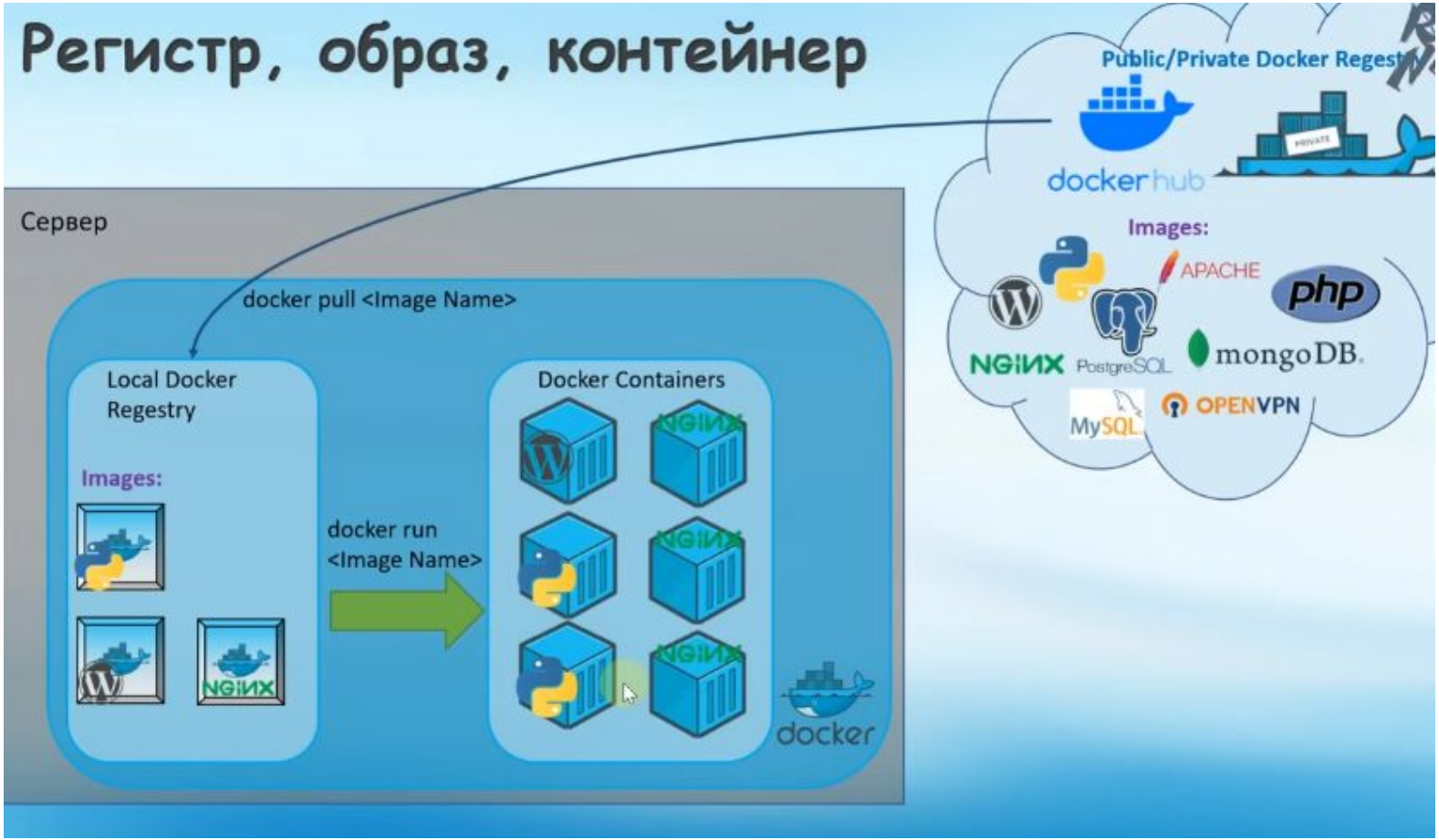
В чем еще плюс?



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



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

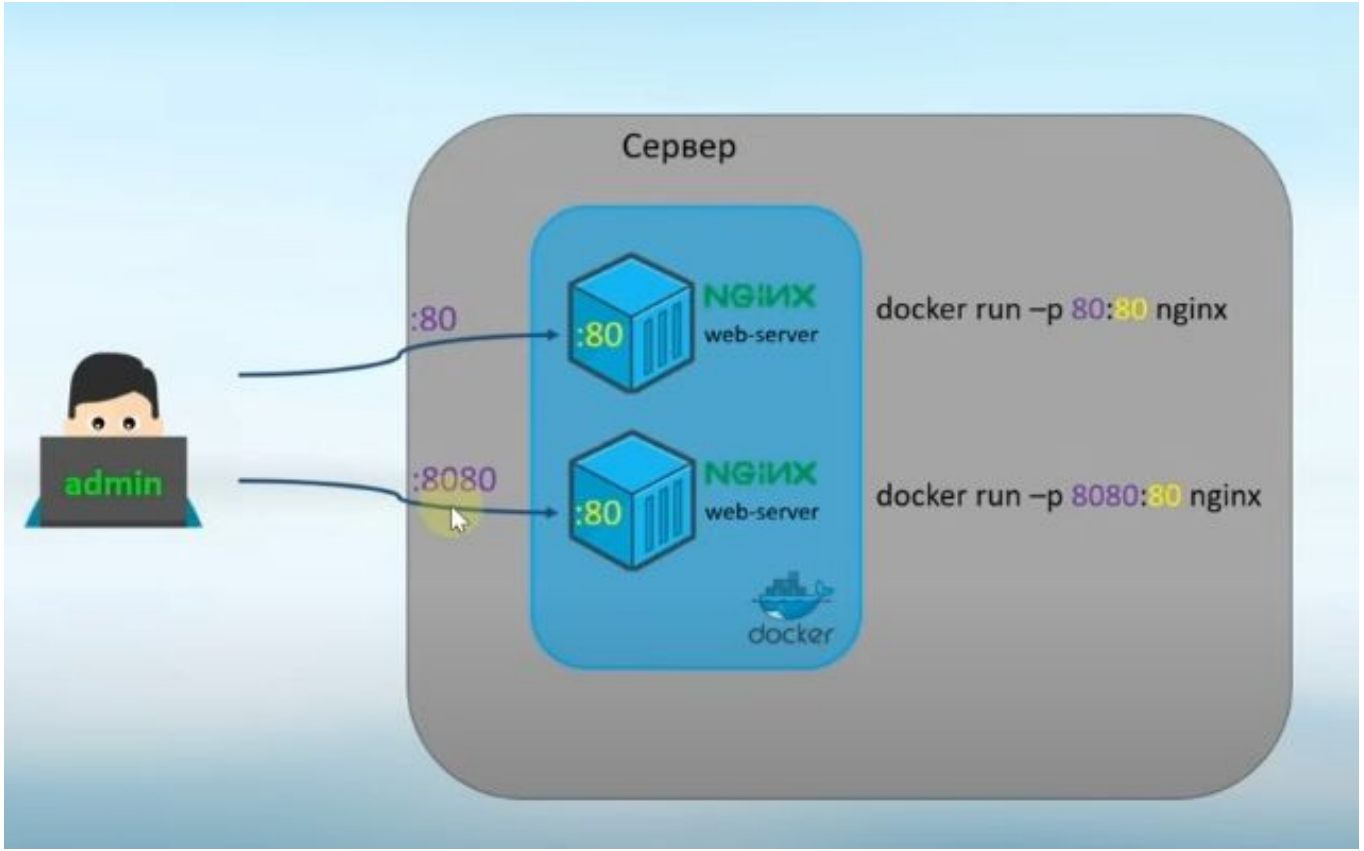
Попробуем Ubuntu

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 (как будто есть мост)



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:

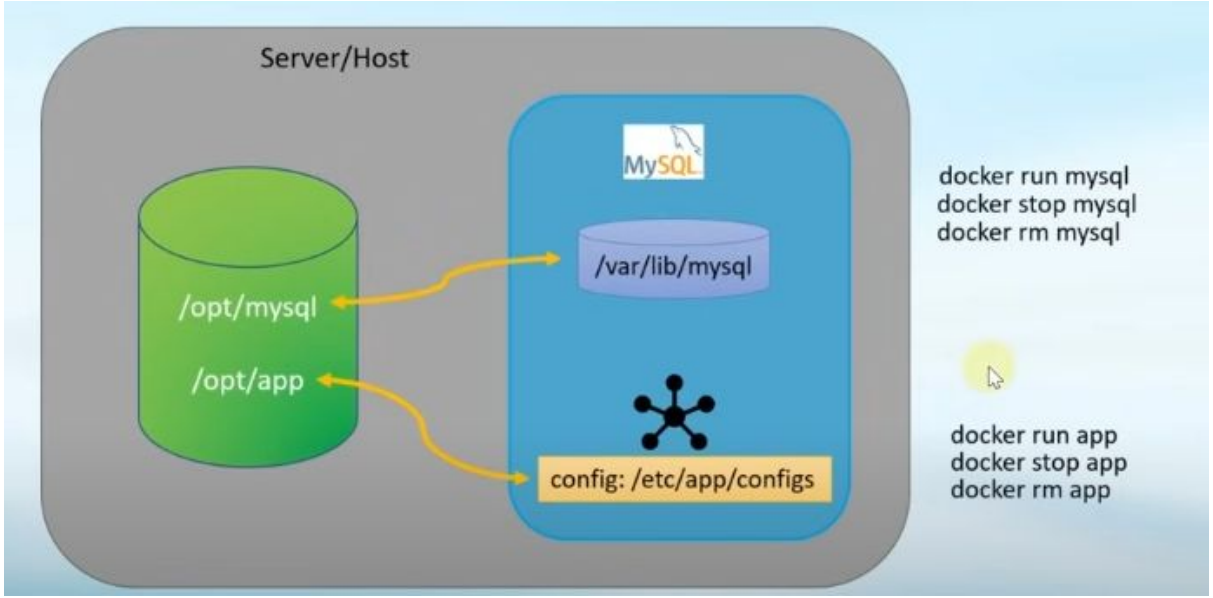
docker exec -it <container id> /bin/bash
и ввести env

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

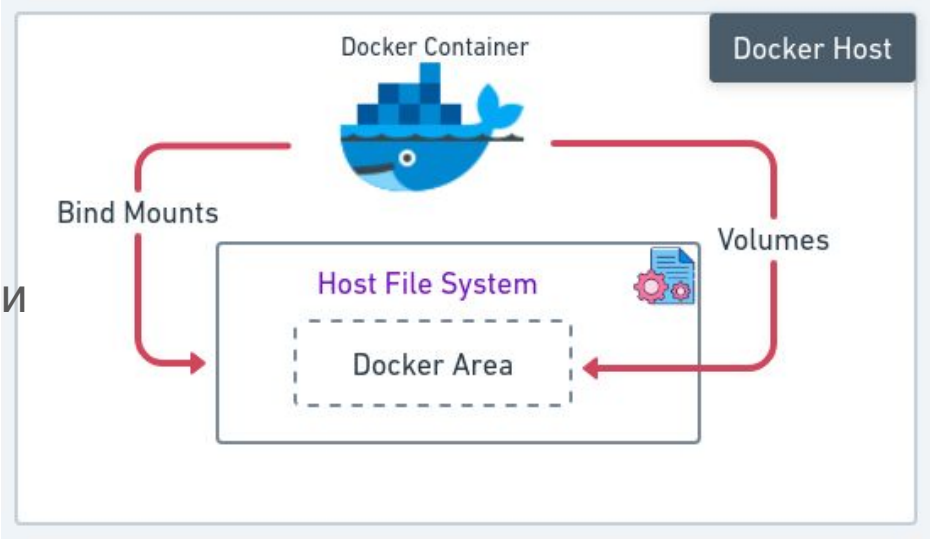
После команды kill удаляются все переменные окружения вместе с контейнером, но что если нам нужно где то постоянно хранить переменные из контейнера. Для этого используются Docker **Volumes и Mounting**



Docker Volume

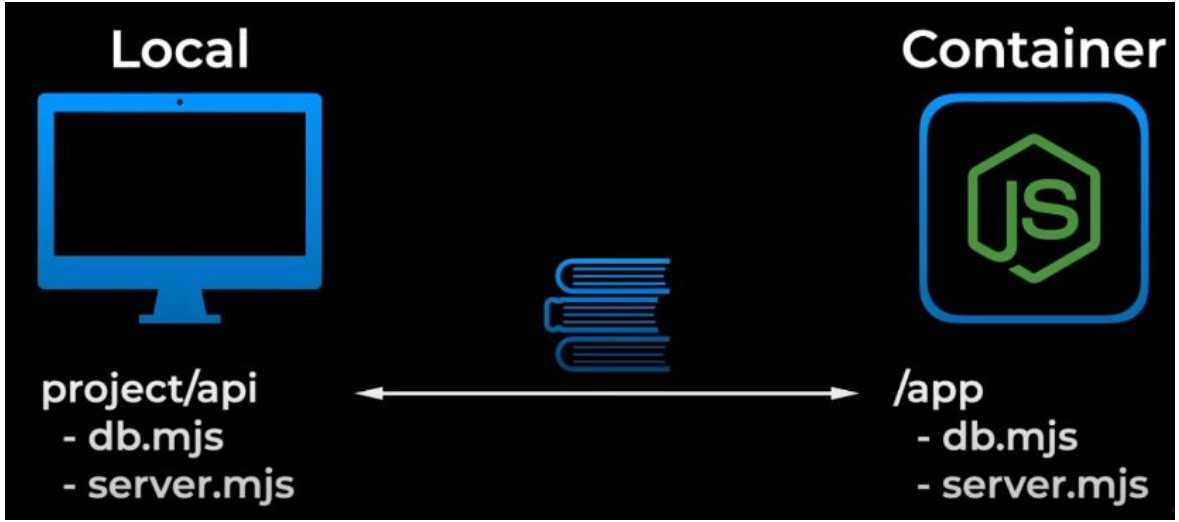
В Docker **том** или **volume** — это механизм для хранения и управления данными, которые используются внутри контейнеров. Том можно рассматривать как отдельную файловую систему, которая существует вне контейнера и может быть подключена к одному или нескольким контейнерам.

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.



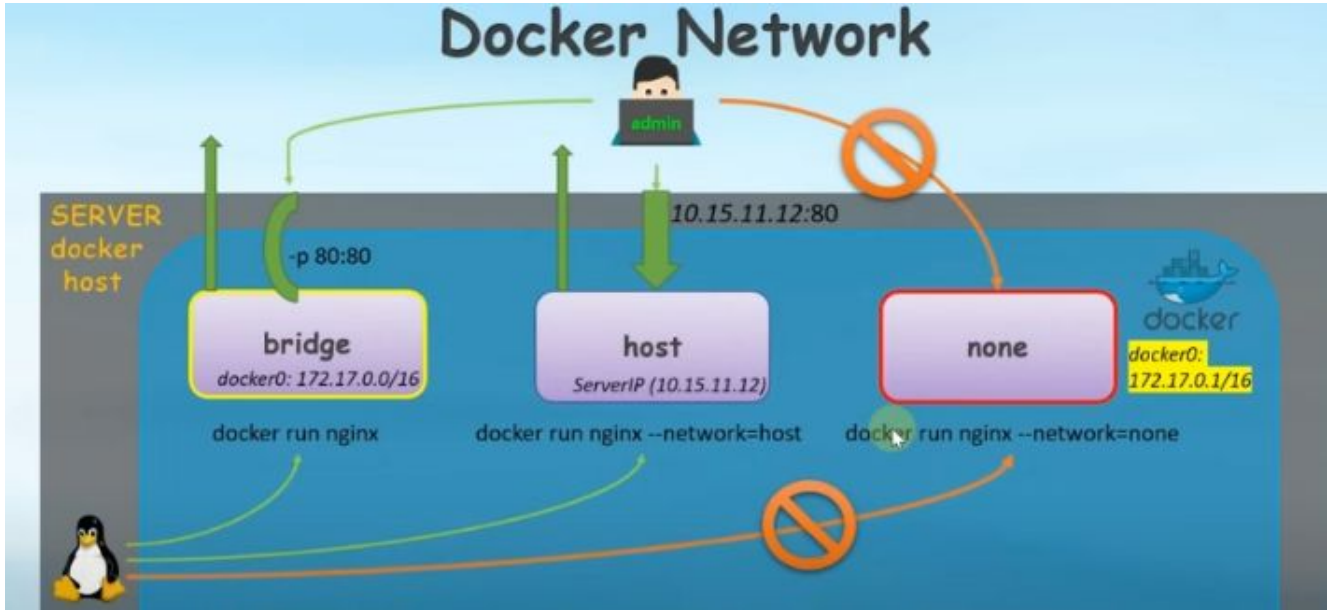
Volumes и Bind Mount

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).



почитать

Какие могут быть кейсы в теории…?

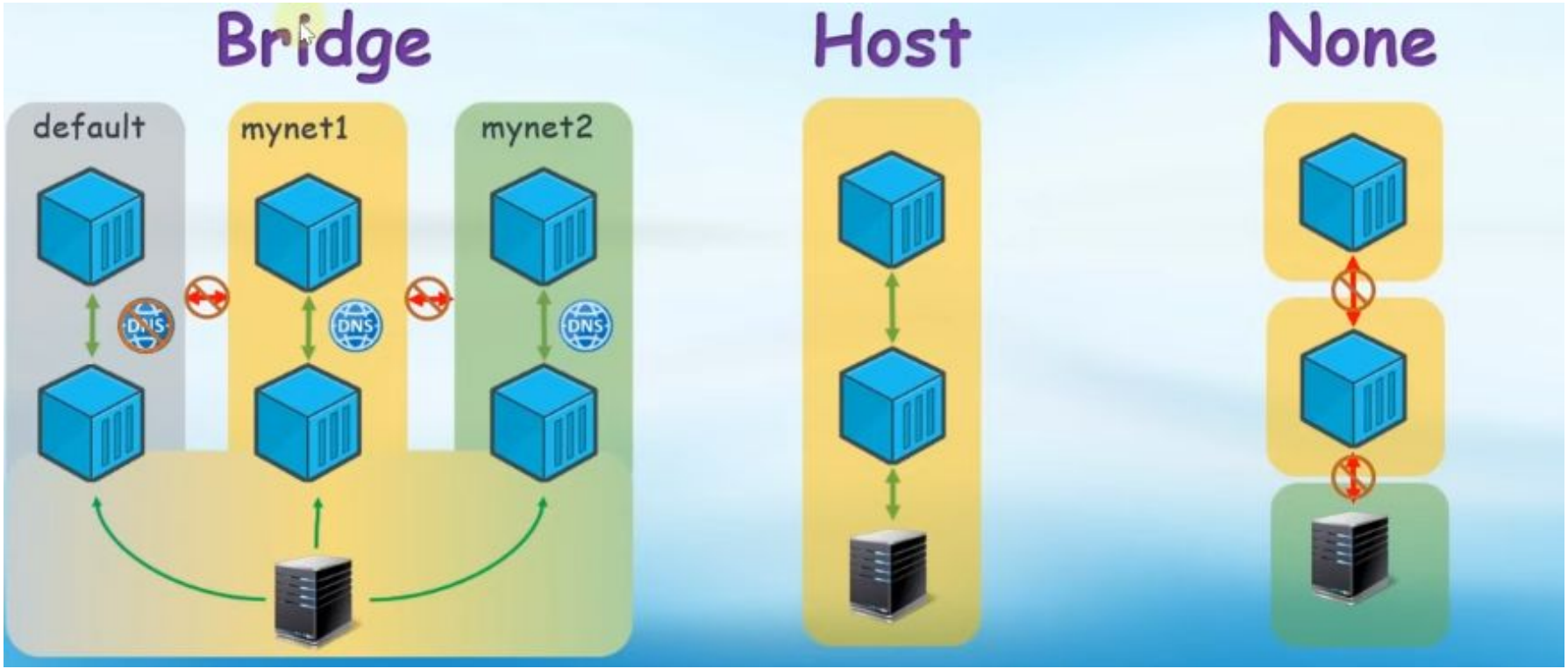
1) Иногда, контейнер должен функционировать в полностью изолированной среде.
2) Что если приложение состоит из нескольких контейнеров, которые должны между собой как то общаться?
3) Что если контейнер должен вести себя так, как будто он работает прямо на хосте?
4) Что если контейнер должен использовать все доступные сетевые ресурсы хоста и взаимодействовать с приложениями, которые также зависят от хостовых сетевых интерфейсов?

В общем, нужна какая-то гибкость…

Типы сетей

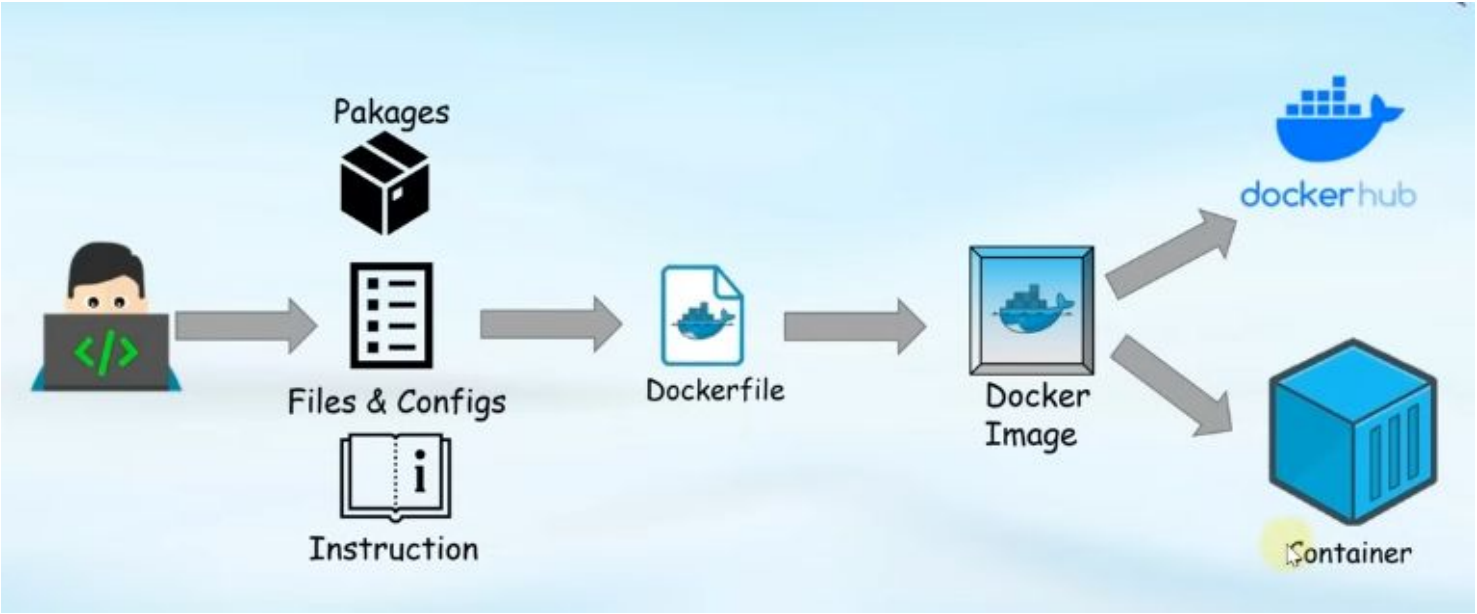
1. **Bridge** - тип сети по умолчанию, то есть при запуске контейнера по умолчанию, он попадает именно в данный тип сети. Они имеют доступ к интернету и локально. Чтобы подключиться к контейнеру, который работает в сети bridge, нужно как бы *создать мостик* (как до этого указывали порты -p 80:80)
2.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

почитать подробнее (есть еще и другие сети)


Bridge
default
mynet1
mynet2
DNS
DNS
DNS
Host
None

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

почитать

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

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"]
```

Какая еще появляется проблема с контейнерами?

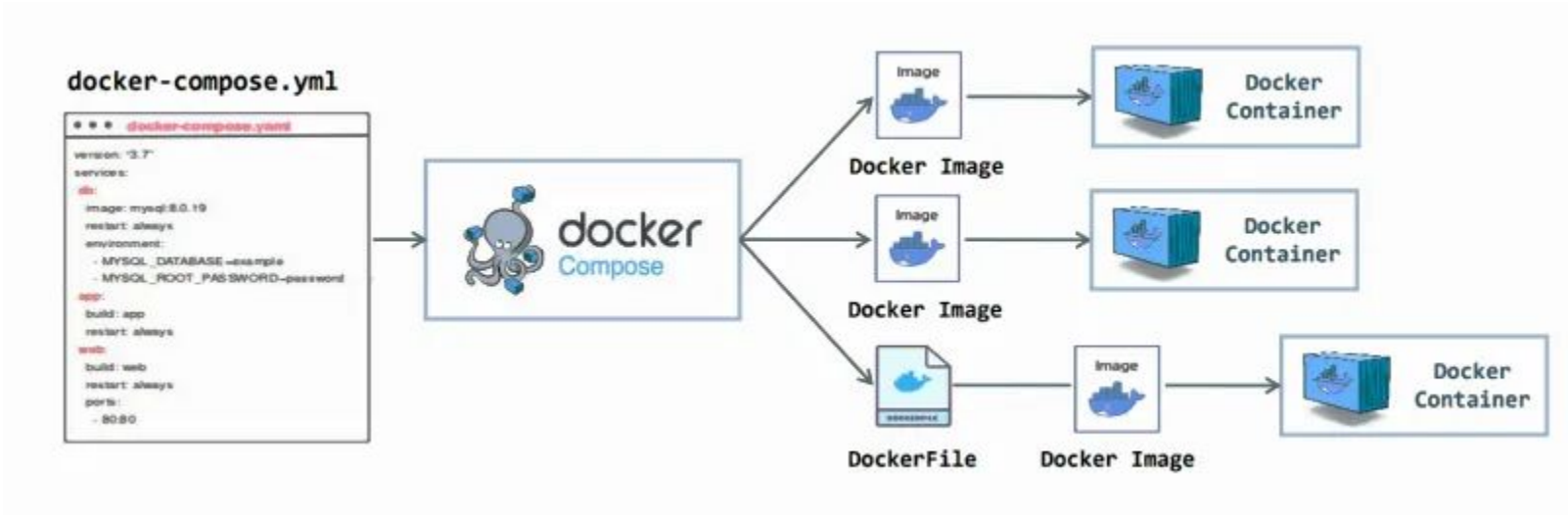
Когда твое приложение разделяется на несколько частей (например, отдельные сервисы для базы данных, API, кэширования и т.д.), лучше использовать несколько контейнеров. **Почему?**

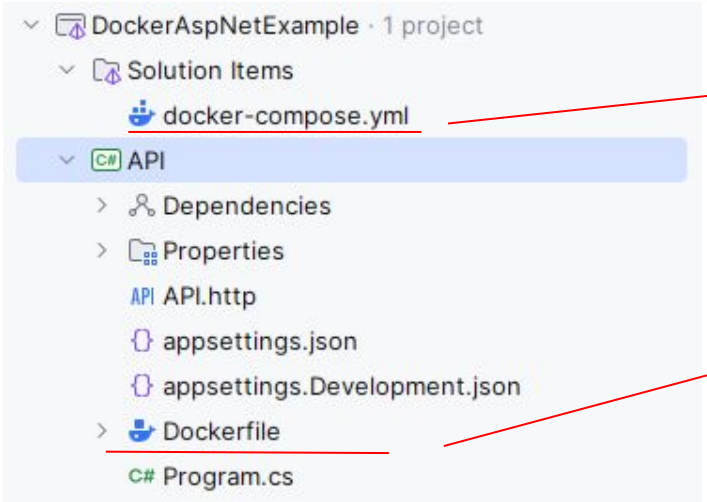
- Когда контейнеры разделены, каждый из них может работать в своей ограниченной среде, что повышает безопасность.
- Каждый контейнер выполняет свою задачу (вспоминаем про Single Responsibility).
- Если приложение стало требовать большего количества ресурсов, ты сможешь легче масштабировать контейнеры, если они разделены по отдельности.
- Если все делать в одном контейнере, будет сложнее отлавливать ошибки и поддерживать работу приложения.

Docker Compose

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

почитать






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