Контейнеризация++

Лекторы:
Аспирант МФТИ, Шер Артём Владимирович
Аспирант МФТИ, Зингеренко Михаил Владимирович

12 ноября 2024

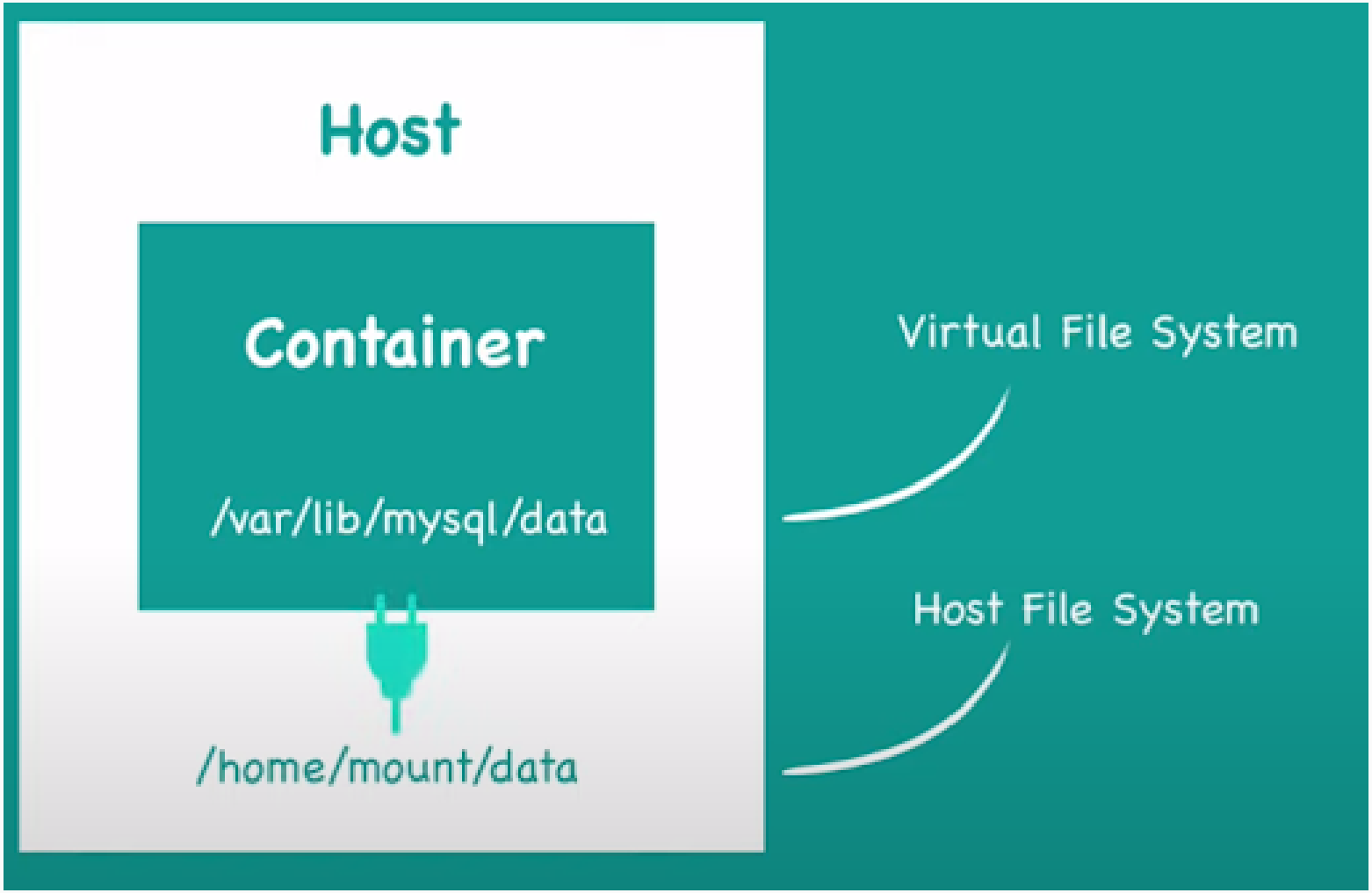
Хранение файлов

Приложения делятся на два типа:

- Stateless – не нужно хранить свое состояние
- Stateful – нужно хранить свое состояние

Docker контейнер по умолчанию хранит все данные внутри себя. При удалении данные теряются. Как сохранить данные? Docker Volumes!

Хранение файлов



Хранение файлов

Типы volume в Docker:

- Host Volume
- Anonymous Volume
- Named Volume

Хранение файлов. Host Volume

Самый простой и удобный способ:

docker run -v <папка на хосте>:<папка в контейнере> ...

“Синхронизируете” папку на хосте и папку в контейнере.

Хранение файлов. Anonymous Volume

Не важно где именно хранятся, просто хотим вывести на хост:

docker run -v <папка в контейнере> ...

Применяются редко, обычно в связке с аргументом –volumes-from.

Эти тома хранятся в /var/lib/docker/volumes/<random_hash>/_data

Хранение файлов. Named Volume

Даем каждому тому свое имя (как и контейнерам):

docker run -v <имя тома>:<папка в контейнере> ...

Применяются чаще всего в проде, потому что управляете ими не вы, а Docker.

Эти тома тоже хранятся в /var/lib/docker/volumes/<random_hash>/_data

Сети

Для соединения контейнеров между собой и внешним миром Docker использует собственные виртуальные сети. К этим сетям и подсоединяются контейнеры. Чтобы посмотреть имеющиеся сети: docker network ls

- Network ID - уникальный идентификатор
- Name - человекочитаемое имя
- Driver - “тип” используемой сети
- Scope - область видимости сети

P.S. схемы взяты с канала NetworkChuck

Сети

Docker поддерживает несколько типов сетей:

- default bridge
- user-defined bridge
- host
- macvlan
- none

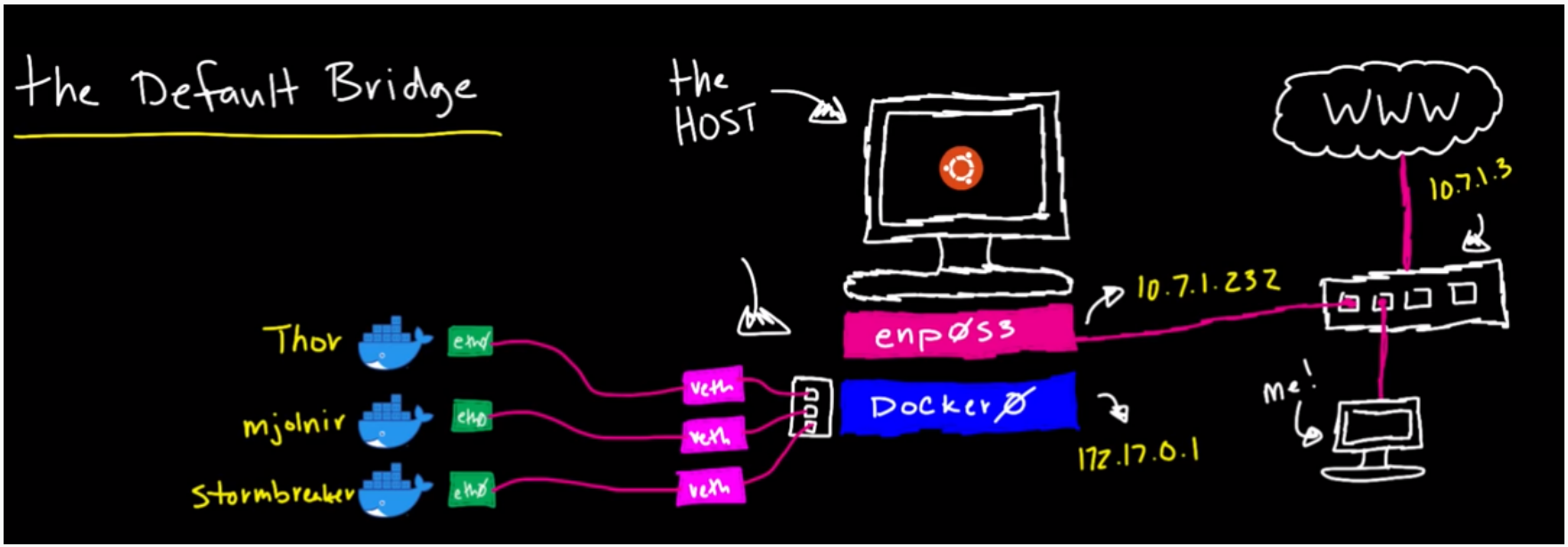
Плюс еще несколько (не рассмотрим сегодня):

- macvlan (802.1q)
- ipvlan L2 & L3
- overlay

Сети. default bridge

Тип сети по-умолчанию. Представляет собой “мост” между контейнерами и внешней сеть - почти как роутер в вашей комнате, только виртуальный и для контейнеров. Посмотрим на имеющиеся сетевые интерфейсы: **ip a** Теперь создадим простой контейнер и еще раз посмотрим на интерфейсы. **docker run -dit –name thor nginx**

Сети. default bridge



Сети. default bridge

bridge - сеть, используемая по-умолчанию при создании контейнеров. Посмотрим на нее подробнее с помощью команды:
docker inspect bridge
Видим еще и IP адреса наших контейнеров, сеть занимается и менеджментом IP адресов внутри нее (DHCP). Контейнеры внутри одной сети могут общаться друг с другом. Но снаружи контейнеры недоступны (без проброса портов):
-p <порт на хосте>:<порт в контейнере>

Сети. User-defined bridge

Можем сделать свою собственную сеть:

docker network create asgard

Посмотрим еще раз на интерфейсы: **ip a**

И на сети Docker: **docker network ls**

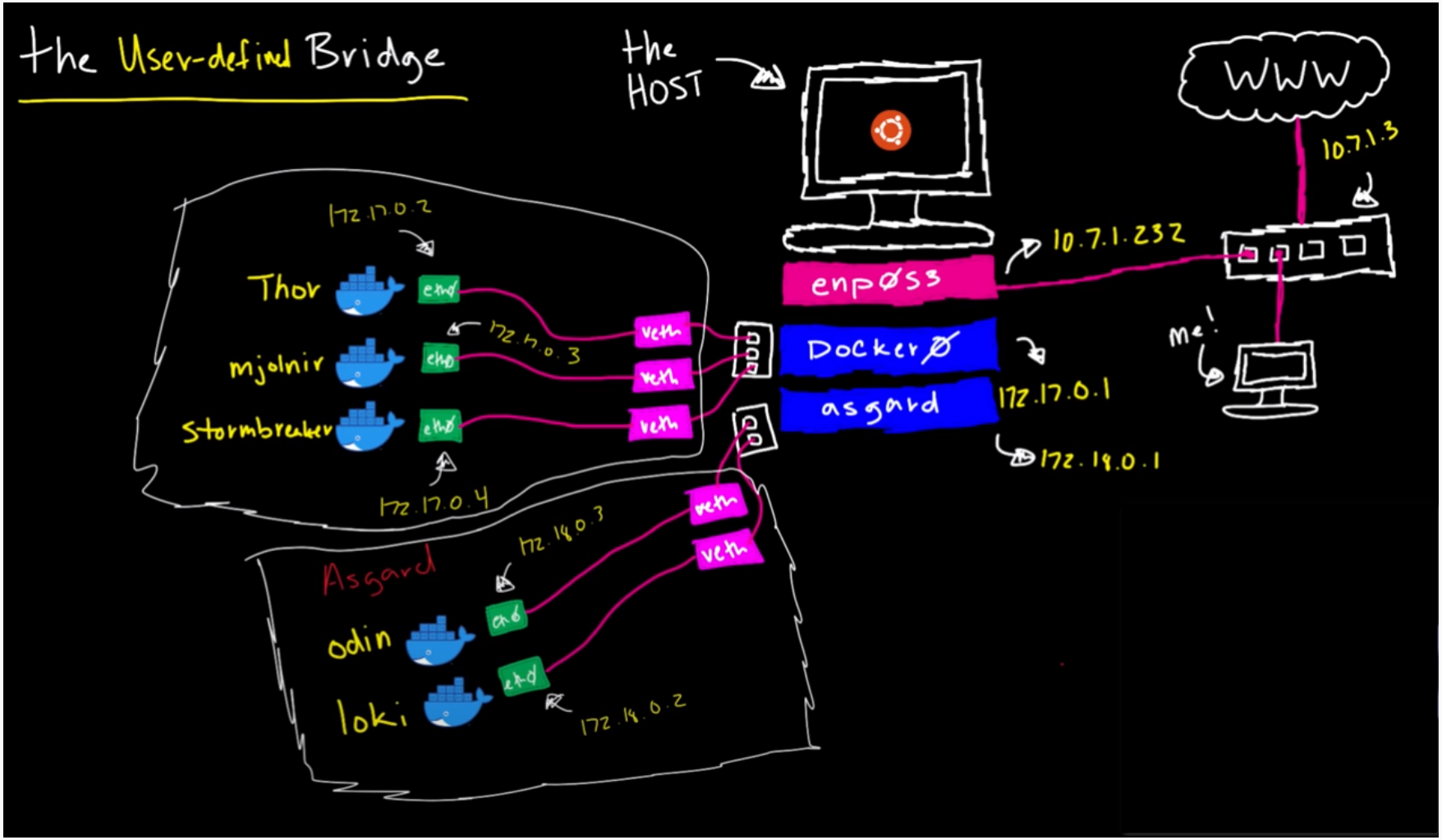
В этой сети одиноко, добавим контейнер:

docker run -dit –network asgard –name loki busybox

Зачем нужны собственные bridge-сети:

изоляция!

Сети. User-defined bridge



Docker Image подробнее

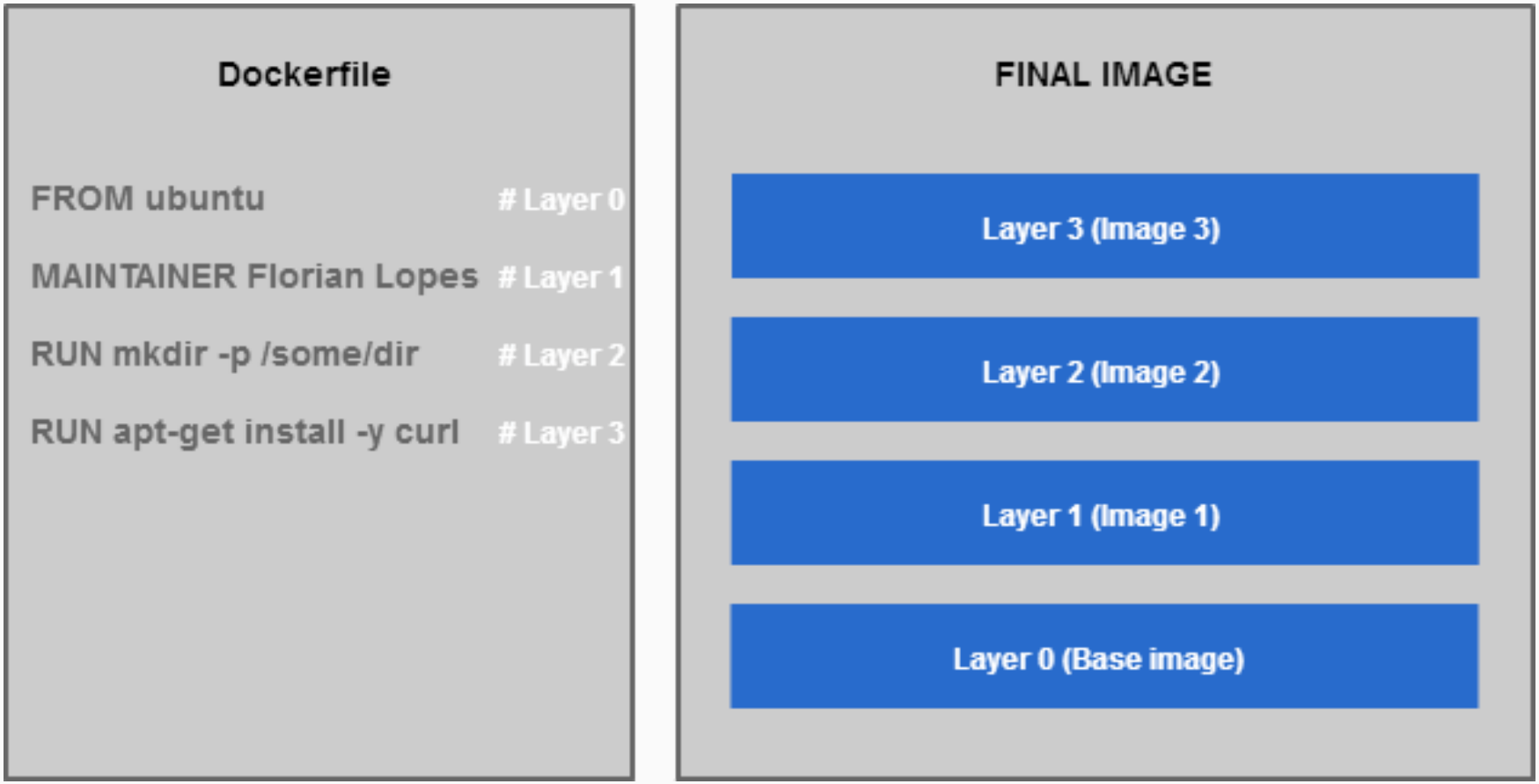
Заметим, что мы создаем образы не с нуля, а используя базовый образ:

- Ubuntu, Debian,...
- python, node, ...

Часто они являются общими для нескольких наших образов. В таком случае получаем дупликацию – занимаем лишнее место.

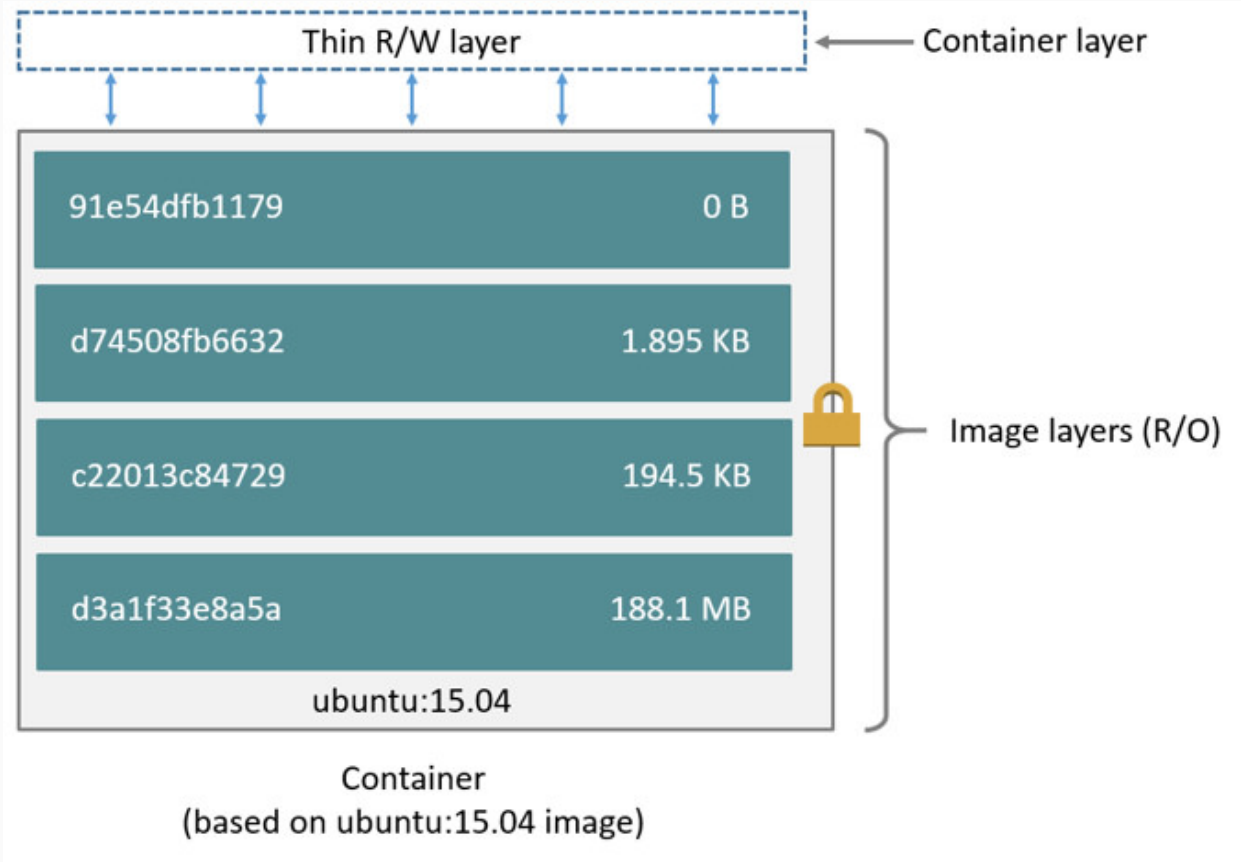
Как можно избавиться от этого? Слоеная архитектура! Каждый слой лишь содержит изменения файлов (diff) - подход Copy on Write

Docker Image подробнее



Docker Image подробнее

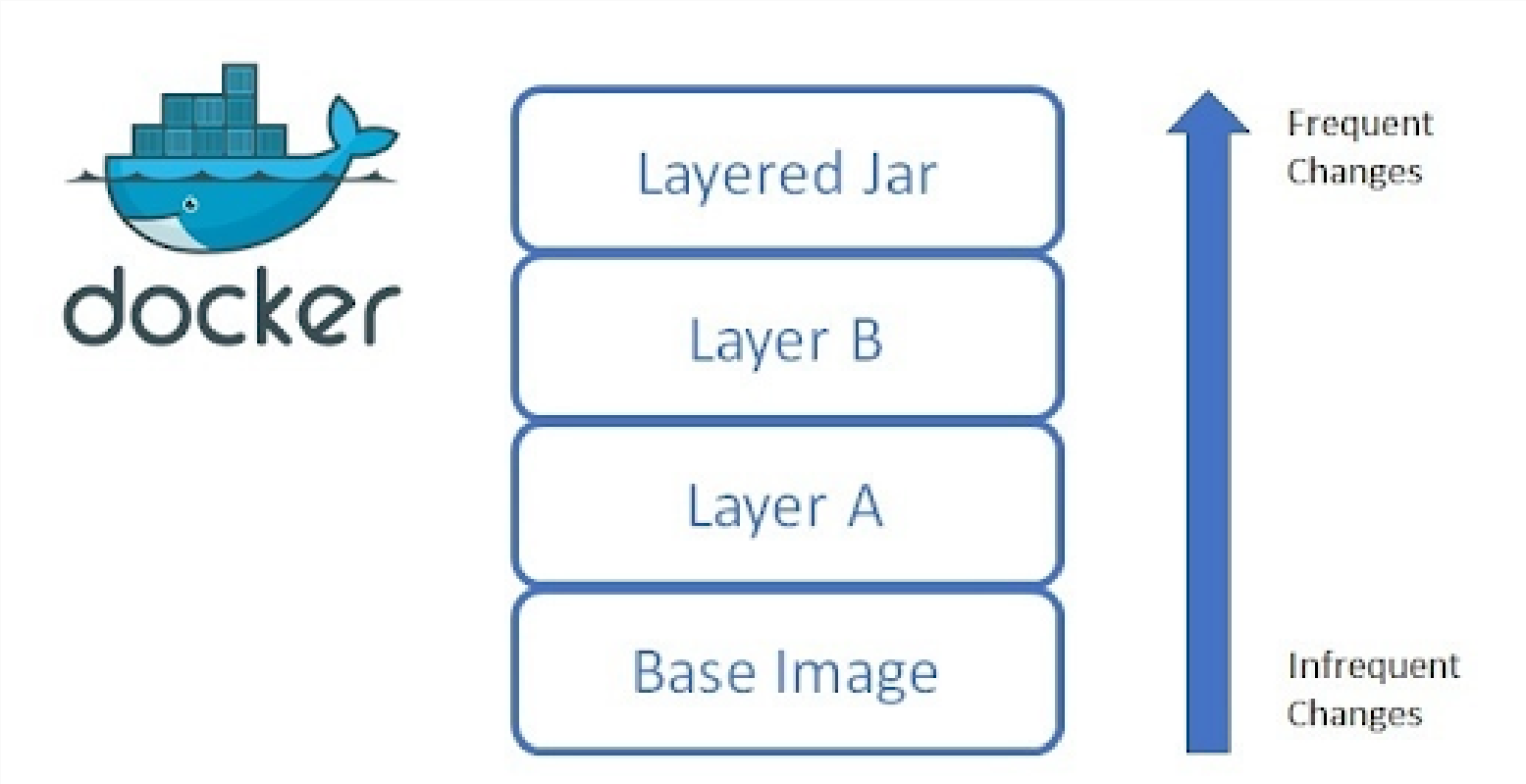
Как же мы переходим к контейнерам от образов? Добавляем еще один слой!



Docker Image подробнее

Сборка образа тоже происходит послойно. При изменении одного из слоев пересобираются только этот слой и идущие после него.

Поэтому часто меняющиеся слои (исходный код приложений) стоит по возможности размещать позднее.



Docker Image подробнее

Практический совет - минимизируйте количество слоев. Меньше слоев - меньше время на сборку - меньше время развертывания - меньше места занимает. Вместо

- RUN apt-get install -y nginx
- RUN apt-get install -y cockpit
- RUN service nginx restart

лучше

RUN apt-get install -y nginx cockpit && service nginx restart

Docker Compose

Команды для создания одного контейнера - длинные. Что делать, когда нужно развернуть инфраструктуру из нескольких (десятков) контейнеров? Нужен инструмент автоматизации! Docker Compose!

Давайте хранить информацию о контейнерах, сетях и томах в одном файле, а затем по этой “инструкции” разворачивать их.

Создаем файл docker-compose.yaml, указываем что хотим развернуть и бам:

docker compose up

Документация

Docker Compose

Структура файла docker-compose.yml

- services: определение сервисов, их образы, порты, volume и зависимости.
- networks: настройка сети для взаимодействия контейнеров.
- volumes: подключение volume для хранения данных.

```yaml
version: '3'
services:
  web:
    image: nginx
    ports:
      - "80:80"
  db:
    image: postgres
    environment:
      POSTGRES_PASSWORD: example
```

До следующей лекции!