Биоинформатика

Инфраструктура вычислений в биоинформатике

Лекция 5. Контейнеризация. Docker. Образы и работа в контейнере. Dockerfile. Сборка образа.

Важность и преимущества микросервисов

компоненты приложения

Статичный веб-сайт
nginx 1.5 + openssl + modsecurity

БД пользователей
postgresql + v8

Аналитическая БД
hadoop + OpenJDK

API-эндпоинты
Python 2.7 + Flask + celery

Web - фронтенд
Redis

фоновые обработчики/воркеры
Python 3.0 + nodejs + ...

Кэш
Redis

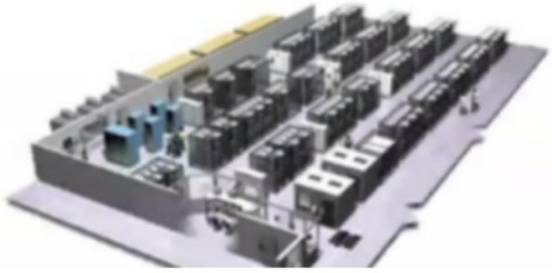
Разработка → Тестирование → Продакшн → Масштабирование

ноутбук разработчика

виртуальные машины

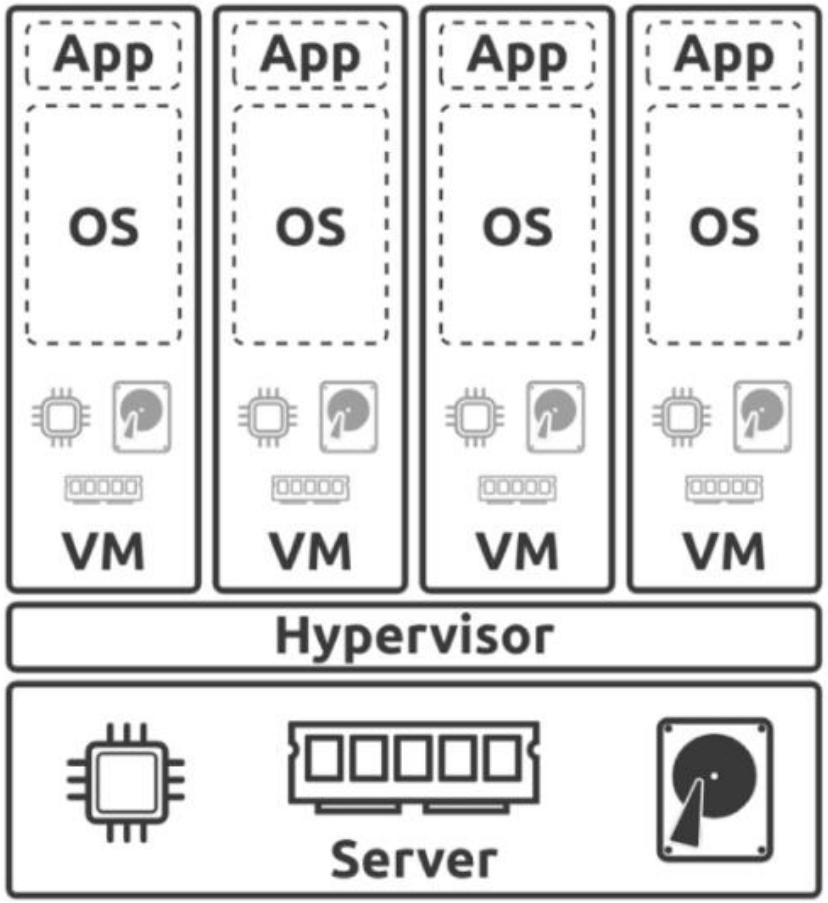
облачные сервисы

серверный кластер

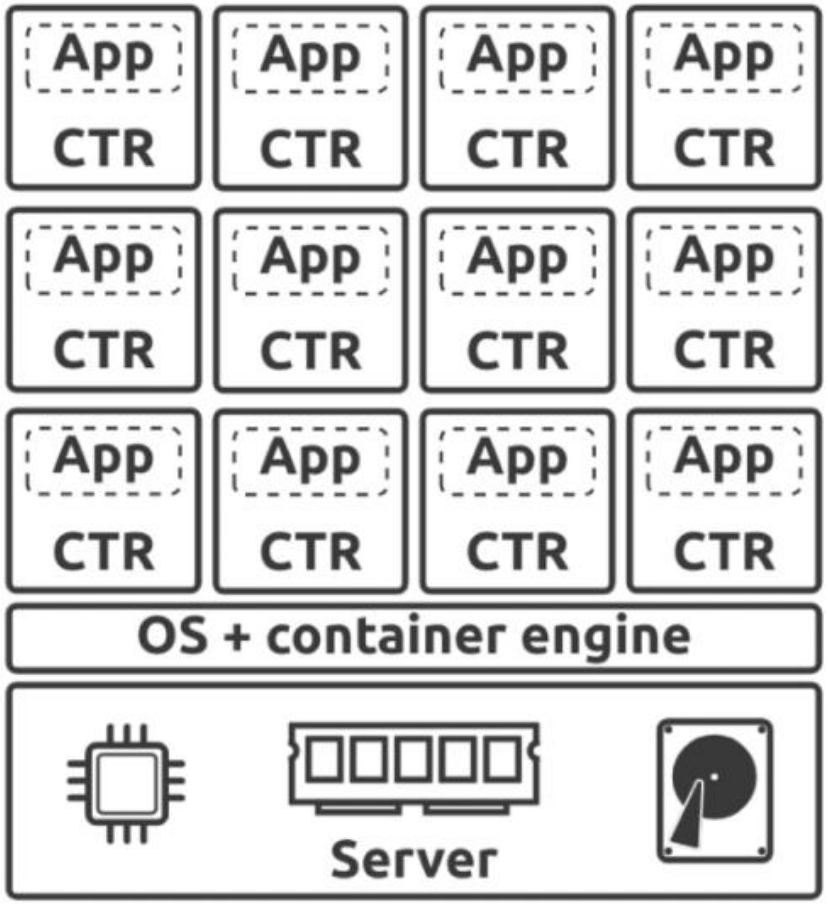


центр обработки данных

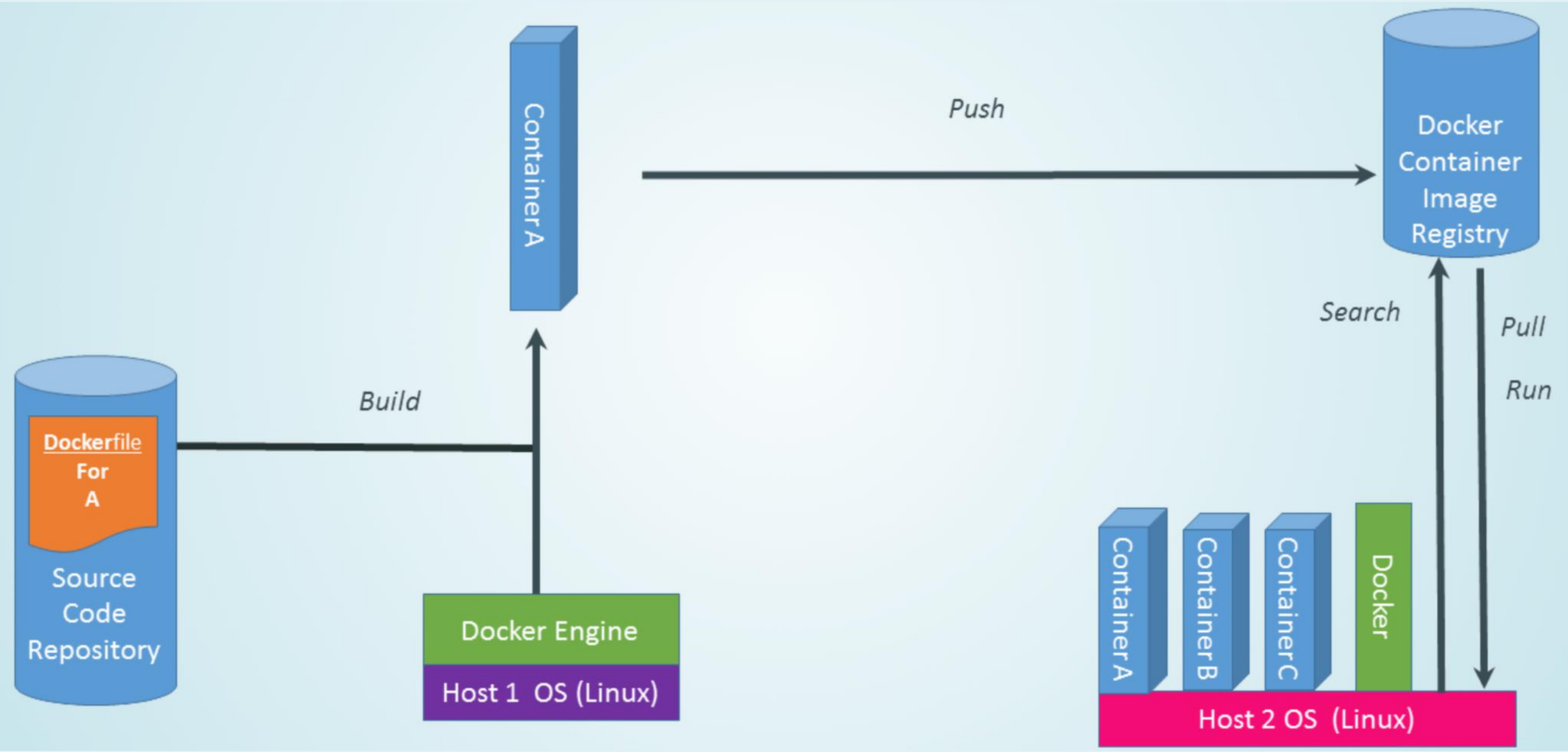
Контейнеры vs виртуальные машины



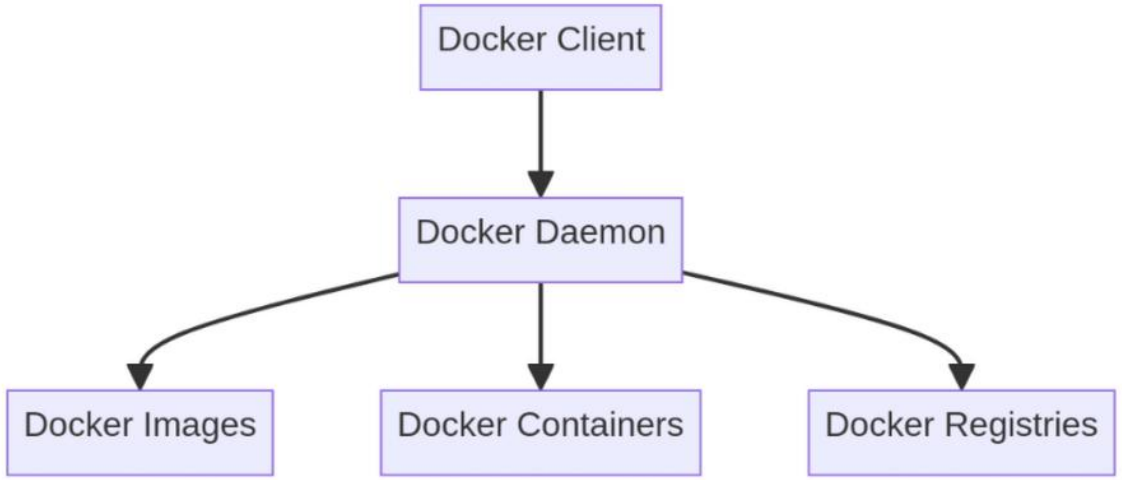
Виртуальная машина



Контейнер


Container A
Push
Docker Container Image Registry
Search
Pull
Run
Build
Dockerfile For A
Source Code Repository
Docker Engine
Host 1 OS (Linux)
Container A
Container B
Container C
Docker
Host 2 OS (Linux)

Элементы экосистемы Docker



Образ — шаблон:
(файловая система + метаданные)

Контейнер — запущенный экземпляр образа

Docker Client — интерфейс пользователя. Общается с Docker Daemon через REST API

Docker Daemon (dockerd) — фоновый процесс, управляющий всеми объектами Docker: образы, контейнеры, сети, тома.

Docker Images — образы для создания контейнеров Неизменяемые, состоят из слоёв (layers)

Docker Containers — запущенные экземпляры образов

Docker Registries — хранилища образов (Docker Hub, частные хранилища). Из них осуществляются push / pull образов.

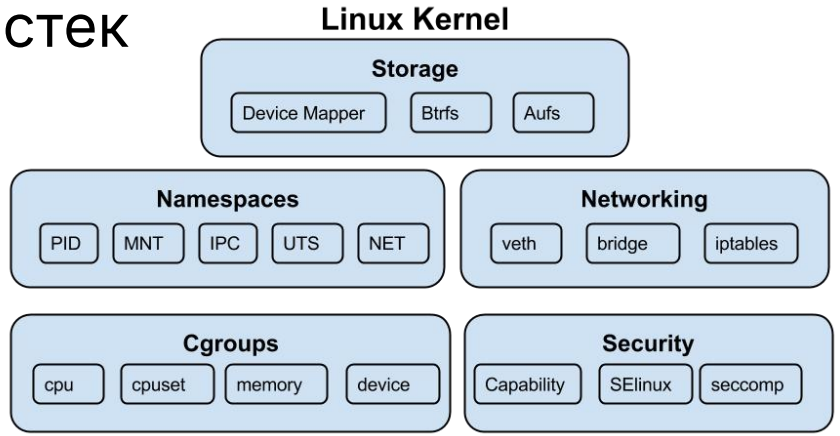
Пространства имен namespaces

Docker использует “namespaces” системы Linux для изоляции ресурсов:

- ***“PID namespaces”***: изоляция идентификаторов процессов — процессы внутри контейнера видят только свои PID

- ***“Mount namespaces”***: изоляция точек монтирования файловой системы — каждый контейнер имеет свою собственную иерархию файлов

- ***“Net namespaces”***: изоляция сетевых интерфейсов, IP-адресов, таблиц маршрутизации — контейнер имеет свой сетевой стек



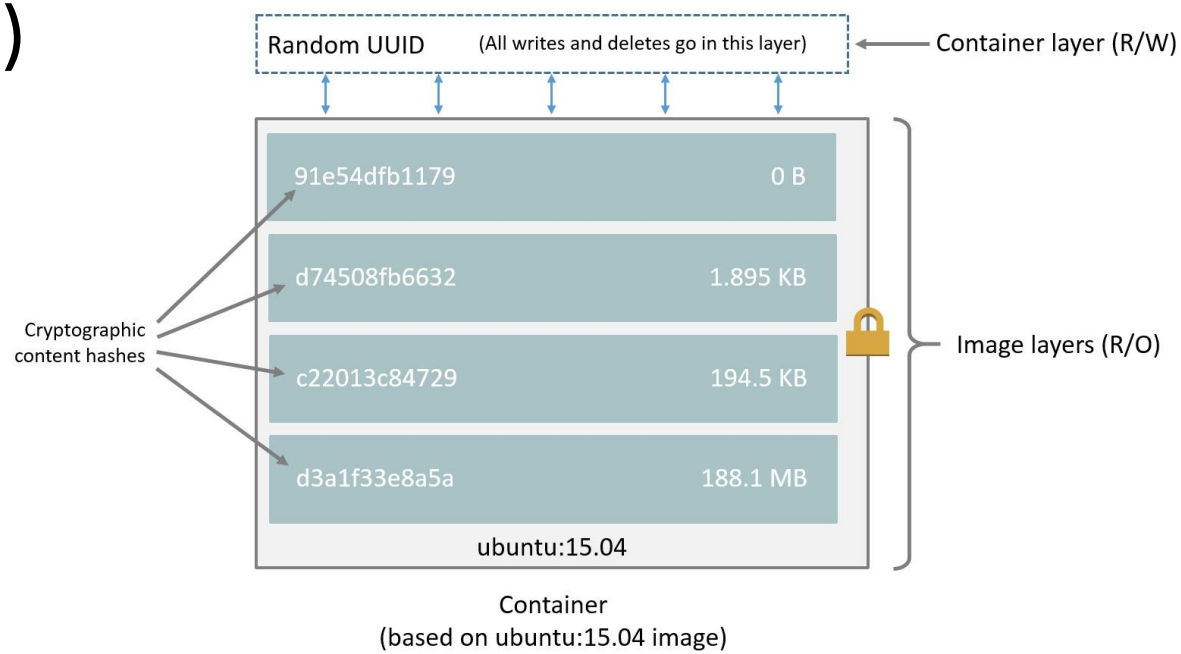
Архитектура Docker

При загрузке Linux запускается с одним процессом - PID 1 (обычно `systemd` или `init`), который затем разветвляется и создаёт все последующие процессы. Когда создаётся контейнер, он получает собственное **пространство имён PID** (PID namespace). Это означает, что:

- Процессы внутри контейнера выглядят так, будто они начинаются с **PID 1**
- На самом деле эти процессы управляются системой хоста и имеют свои собственные уникальные номера PID.

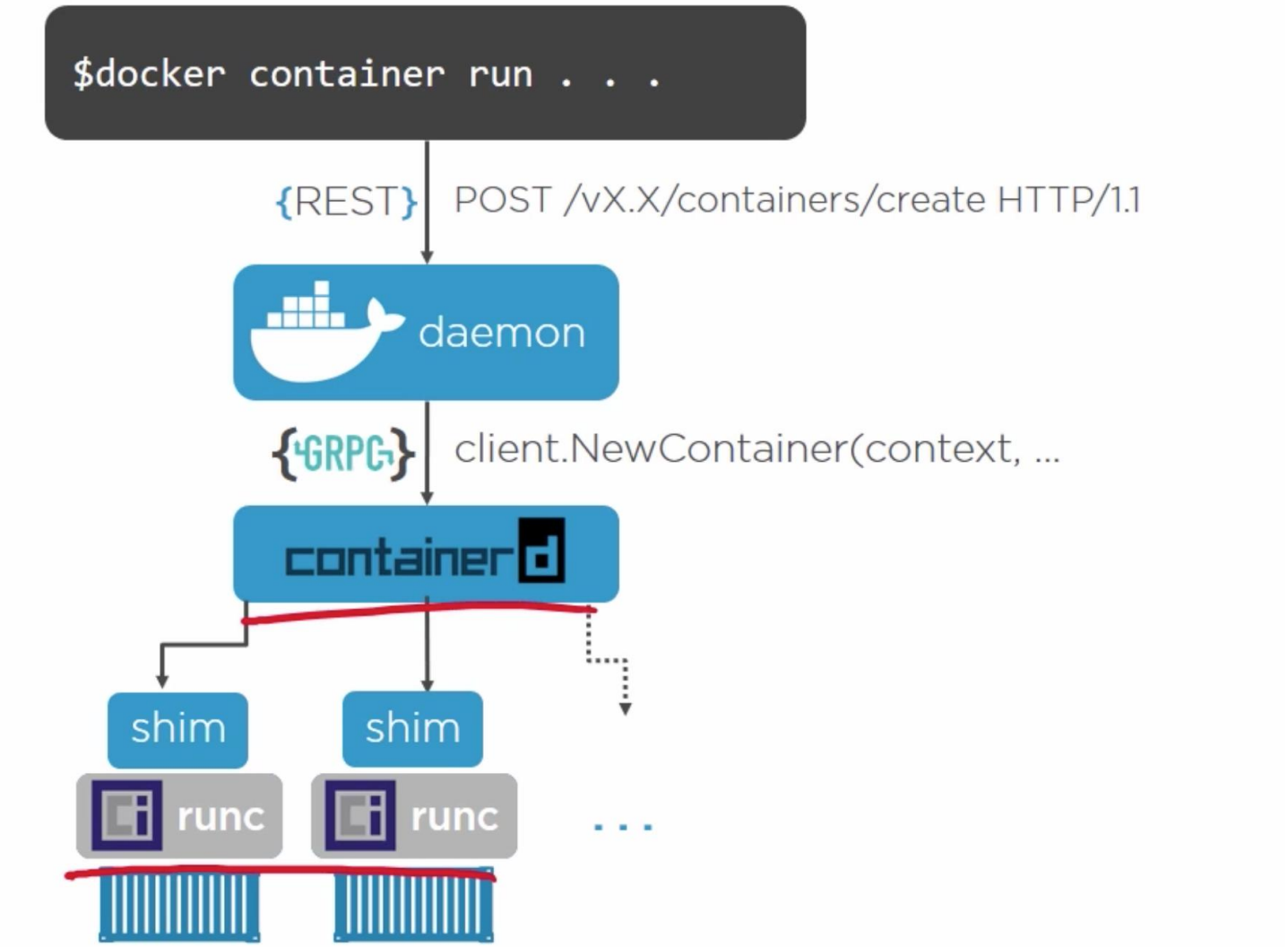
Образы Docker (Docker Images)

- **Образ** — неизменяемый шаблон
- Состоит из слоёв (**layers**), каждый слой — изменение файловой системы
- Образы неизменяемы: изменения в контейнере → новый образ (через **`commit`** или новый **`Dockerfile`**)



Архитектура Docker

- **runc**
- пространства имен (**namespaces**)
- **Cgroups**



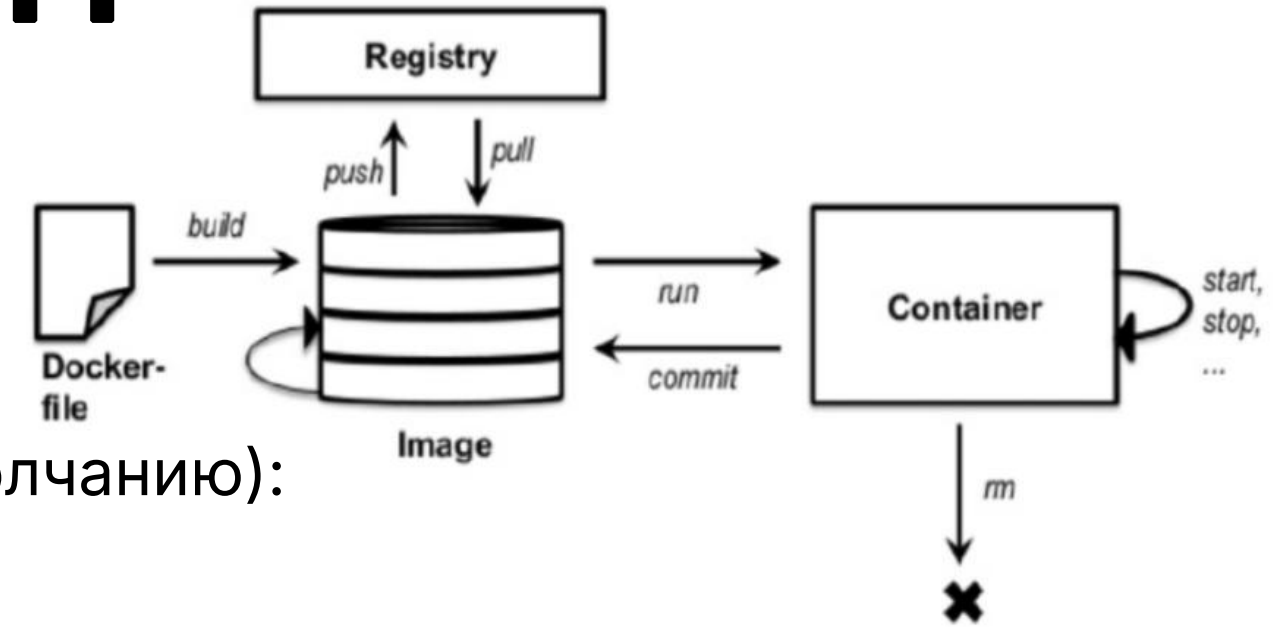
Ресурсы и запуск контейнера

По умолчанию контейнер может использовать ВСЕ ресурсы хоста, один "жадный" контейнер может "съесть" **всю CPU, память, I/O**

Docker использует контрольные группы Linux (**`cgroups`**) для ограничения аппаратных ресурсов каждого контейнера

```
docker run --cpus=0.5 ubuntu
docker run --memory=100m ubuntu
```

Базовые команды



Загрузка образа (latest-версия по умолчанию):

```
docker pull ubuntu
```

Просмотр локальных образов:

`docker image ls` ИЛИ `docker images`

Запуск контейнера:

```
docker run -d -p 5000:5000 --name mycontainer ubuntu
```

Остановка контейнера:

`docker stop mycontainer` ИЛИ `docker stop <container_id>`

Базовые команды

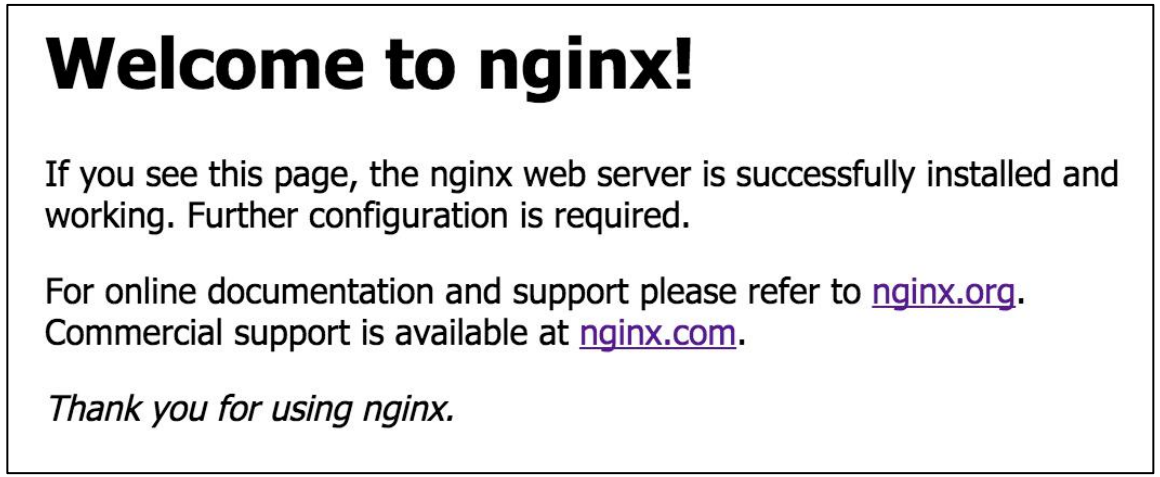
```
root@example:~# mkdir myapp && cd myapp
root@example:~/myapp# docker pull mysql
Using default tag: latest
latest: Pulling from library/mysql
500d7b2546c4: Pull complete
fc5138e88017: Pull complete
b534c7c08c95: Pull complete
5525b1bd2d5d: Pull complete
9effc86d91a3: Pull complete
a1bcea418c7c: Pull complete
fc3e1c37f699: Pull complete
30e3c68e682c: Pull complete
50786f9db9d5: Pull complete
4ea0fa0ace0c: Pull complete
Digest: sha256:439bfb4044dc59ade76c4e5c4065c02e5ba4d4007db32c40ac58d55c03069916
Status: Downloaded newer image for mysql:latest
docker.io/library/mysql:latest
root@example:~/myapp# docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
mysql        latest    2c5440daffa8   4 weeks ago   921MB
```

Базовые команды

```
docker run -d -p 5000:80 --name my-nginx nginx
```

- `-d` — detached (в фоне)
- `-p 5000:5000` — проброс порта хост:контейнер
- `--name` — удобное имя
- `nginx` — официальный образ nginx

После запуска http://localhost:5000 в браузере



Базовые команды

- Список запущенных контейнеров
  `docker ps`
- Список ВСЕХ контейнеров (включая остановленные)
  `docker ps -a`

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                      PORTS               NAMES
355b1893e282        alpine              "/bin/sh"                11 minutes ago      Exited (0) 11 minutes ago                       test-con1
a972eee241f7        redis               "docker-entrypoint.s…"   11 minutes ago      Exited (1) 11 minutes ago                       flamboyant_nobel
e837fbb4217c        alpine              "/bin/sh"                11 minutes ago      Exited (0) 11 minutes ago                       vigilant_jang
3648ebf8ea8e        alpine              "/bin/sh"                11 minutes ago      Exited (0) 11 minutes ago                       silly_booth
c6940736a75b        ubuntu              "sleep 3600"             11 minutes ago      Up 11 minutes                                   quirky_leakey
$ docker ps --all
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                      PORTS               NAMES
355b1893e282        alpine              "/bin/sh"                11 minutes ago      Exited (0) 11 minutes ago                       test-con1
a972eee241f7        redis               "docker-entrypoint.s…"   11 minutes ago      Exited (1) 11 minutes ago                       flamboyant_nobel
e837fbb4217c        alpine              "/bin/sh"                11 minutes ago      Exited (0) 11 minutes ago                       vigilant_jang
3648ebf8ea8e        alpine              "/bin/sh"                11 minutes ago      Exited (0) 11 minutes ago                       silly_booth
c6940736a75b        ubuntu              "sleep 3600"             11 minutes ago      Up 11 minutes                                   quirky_leakey
```

Базовые команды

- Остановка запущенного контейнера (отправляет `SIGTERM`, ждёт 10 секунд, потом `SIGKILL`)
  ```
  docker stop hello-world
  ```
- Если контейнер не останавливается (мгновенная остановка `SIGKILL`)
  ```
  docker kill hello-world
  ```
- Удаление остановленного контейнера (удаляет контейнер полностью, освобождает место)
  ```
  docker rm hello-world
  ```
- Удаление + остановка в одной команде
  ```
  docker rm -f hello-world
  ```

docker info

```
Server Version: 28.0.4
Storage Driver: overlay2
 Backing Filesystem: extfs
 Supports d_type: true
 Using metacopy: false
 Native Overlay Diff: true
 userxattr: false
Logging Driver: json-file
Cgroup Driver: systemd
Cgroup Version: 2
Plugins:
 Volume: local
 Network: bridge host ipvlan macvlan null overla
 Log: awslogs fluentd gcplogs gelf journald json
Swarm: inactive
Runtimes: io.containerd.runc.v2 nvidia runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 05044ec0a9a75232cad458027ca8
```

Работа внутри контейнера — docker exec

Два режима:

- Интерактивный (`-it`)
- Одноразовый (без `-it`)

```
docker exec -it webserver sh
docker exec -it webserver bash
docker exec whoami
```

docker inspect

Возвращает информацию (в формате JSON) о различных объектах Docker: контейнерах, образах, сетях, томах

```
docker inspect
docker container inspect
docker image inspect
docker network inspect
docker volume inspect
```

Содержимое контейнера

Контейнеры в DockerHub оптимизированы - содержат минимальный набор утилит.

Например, команда **nano** скорее всего не работает.

Установить нужные пакеты можно в Dockerfile при создании образа.

Dockerfile

Dockerfile — это текстовый файл с инструкциями для сборки Docker-образа

Он описывает, как именно должен выглядеть контейнер: от какой базы начинать, какие пакеты установить, какие файлы скопировать и что запускать при старте контейнера.

Dockerfile

```
FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /src
COPY ["WebApplication/WebApplication.csproj", "WebApplication/"]
RUN dotnet restore "WebApplication/WebApplication.csproj"
COPY . .
WORKDIR "/src/WebApplication"
RUN dotnet build "WebApplication.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "WebApplication.csproj" -c Release -o /app/publish

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

Что написать в Докерфайле для создания простого образа?

```
FROM python:3.11-slim
WORKDIR /app
COPY . .
RUN pip install flask
EXPOSE 8000
CMD ["python", "app.py"]
```

базовый образ

установка рабочей директории

копирование файлов/директорий

команды, выполняемые **во время сборки** (установка библиотек, создание файлов ..)

открытые **порты**

команда по умолчанию при запуске контейнера

Dockerfile – примеры

```
1  ## Используем официальный образ Ubuntu 22.04 как базовый
2  FROM ubuntu:22.04
3
4  ## Устанавливаем рабочую директорию внутри контейнера
5  WORKDIR /app
6
7  ## Копируем  скрипт hello-world.sh из текущей папки в контейнер
8  COPY hello-world.sh .
9
10 ## Делаем скрипт исполняемым
11 RUN chmod +x hello.sh
12
13 ## Указываем команду, которая будет выполняться при запуске контейнера
14 CMD ["./hello.sh"]
```

Сборка образа

```
docker build -t hello-world:1.0 .
```

Формат: имя:тег (если тег не указан → :latest). Точка = текущая директория

Запуск контейнера: `docker run --rm hello-world:1.0`

При выполнении **`docker build`**:

- Docker читает Dockerfile сверху вниз
- Выполняет каждую инструкцию и создаёт с ней слой
- Кэширует слои (если ничего не изменилось, то использует кэш)

Образы

Образы неизменяемы: изменения в контейнере → новый образ (через **`commit`** или новый **`Dockerfile`**).

`docker commit` — создаёт новый образ на основе изменений в запущенном (или остановленном).

Например:

Запускаем в интерактивном режиме:

```
docker run -it --name my-ubuntu ubuntu:24.04 bash
```

Внутри контейнера:

```
apt update && apt install -y nano htop
exit
```

На хосте:

```
docker commit my-ubuntu my-ubuntu-with-tools:1.0
docker images
```