



СЕТЕВАЯ  
АКАДЕМИЯ  
ЛАНИТ



# Контейнеры

# Docker

## ПЛАН

- Введение в экосистему контейнеров на основе Docker
- Настройка рабочего окружения, подготовка и запуск Docker-контейнеров.
- Docker Compose
- DockerHub
- Сетевое взаимодействие приложений

## М - Монолит

- Крупные приложения
- Много зависимостей
- Много времени до нового релиза
- Инстансы и сервисы как домашние питомцы
- Виртуализация, как способ немного повысить эффективность

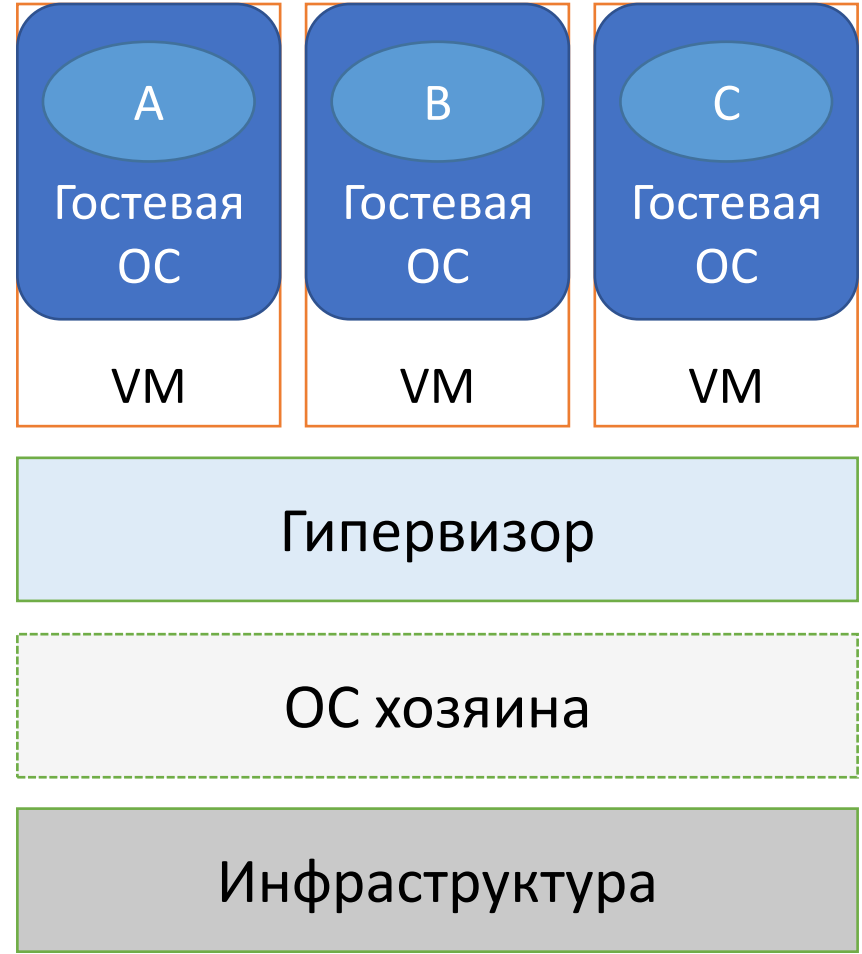
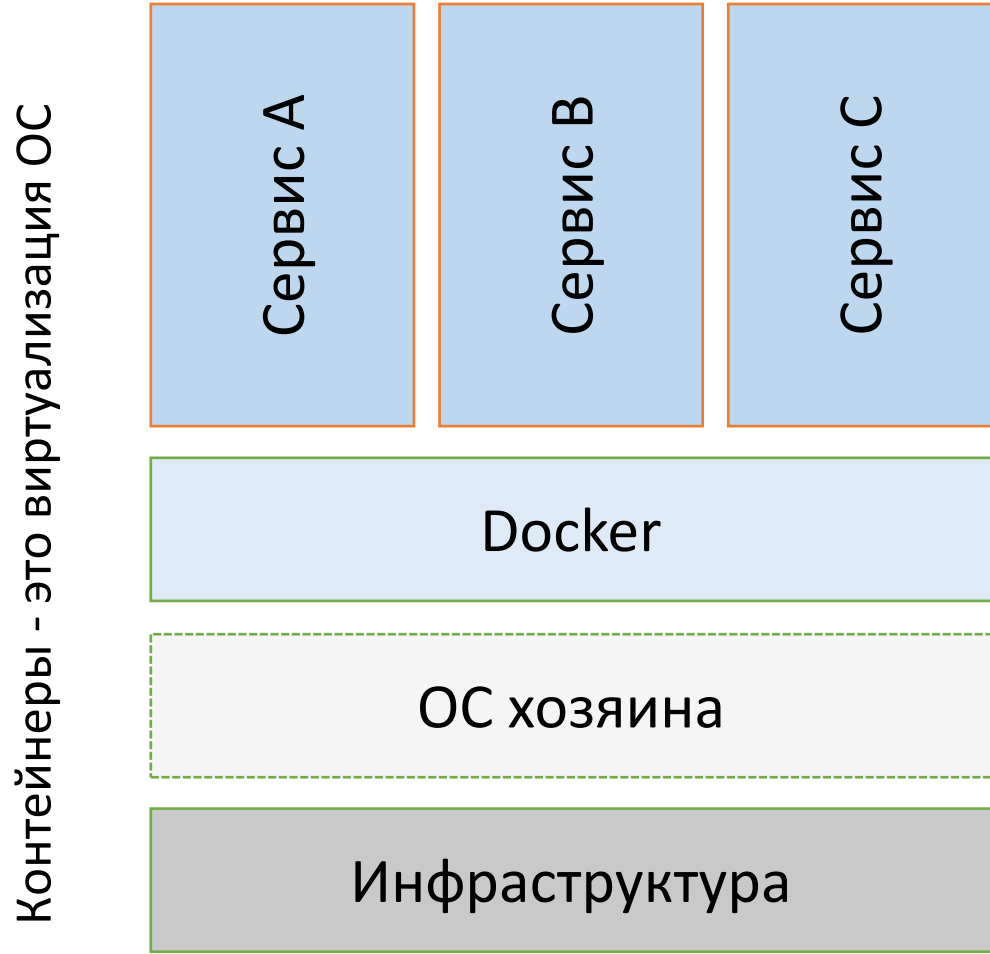
## Важное примечание об изоляции сервисов

- Изоляция сервисов – это важно!
- Повышение доступности достигается увеличением количества экземпляров/VM
- Портiruемость имеет значение, в том числе для развертывания
- Все это повышает затраты

**Как насчет  
изоляции без ОС?**

Представьте себе сервисы,  
работающие **в одной ОС,**  
но **изолированно!**

Контейнеры предлагают изоляцию,  
а не Виртуализацию



VM – это виртуализация железа

# В основе контейнера



# Контейнер

- Смена парадигмы
- Один процесс – один сервис – один контейнер
- Все зависимости – в контейнере
- Маленький образ – хороший образ
- Эфемерные инстансы (питомцы становятся стадами)

# Контейнер

Процесс, работающий в Директории

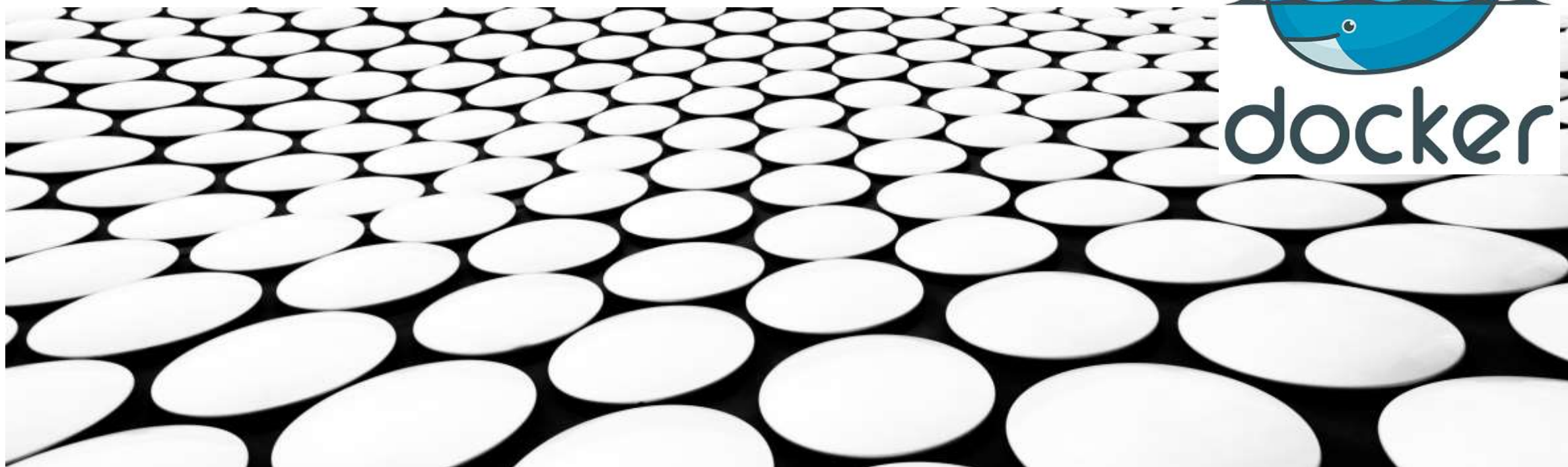
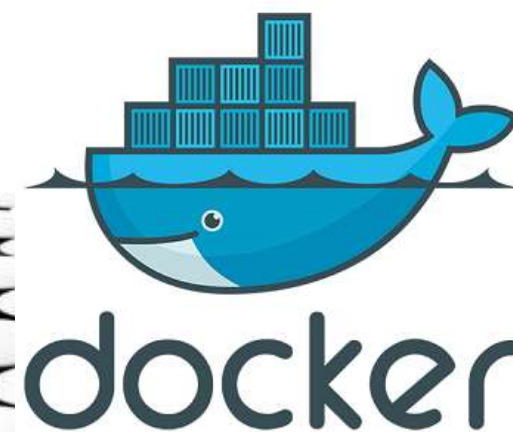
- Процесс (изолированный)
- Директория (*Namespace, cgroup*)
- Все необходимые bin/lib – там же
- Директория с IP-адресом для подключения
- Контейнеры делят ядро ОС с хозяином
  - поэтому не требуют установки собственной ОС
- Контейнер – стандартная единица ПО, которая содержит
  - Код
  - Зависимости





Кто управляет вашими контейнерами?

# Docker

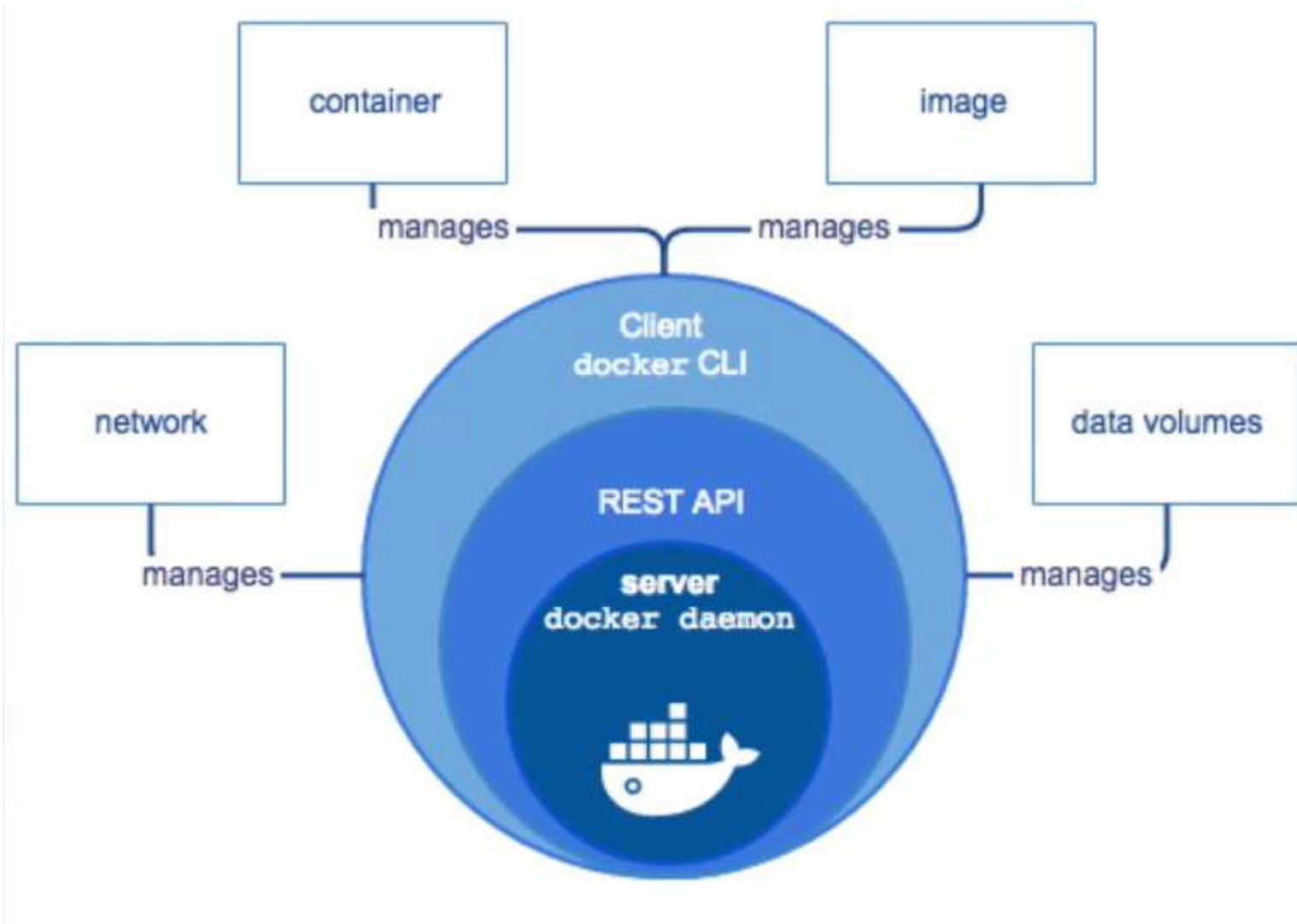


## История появления

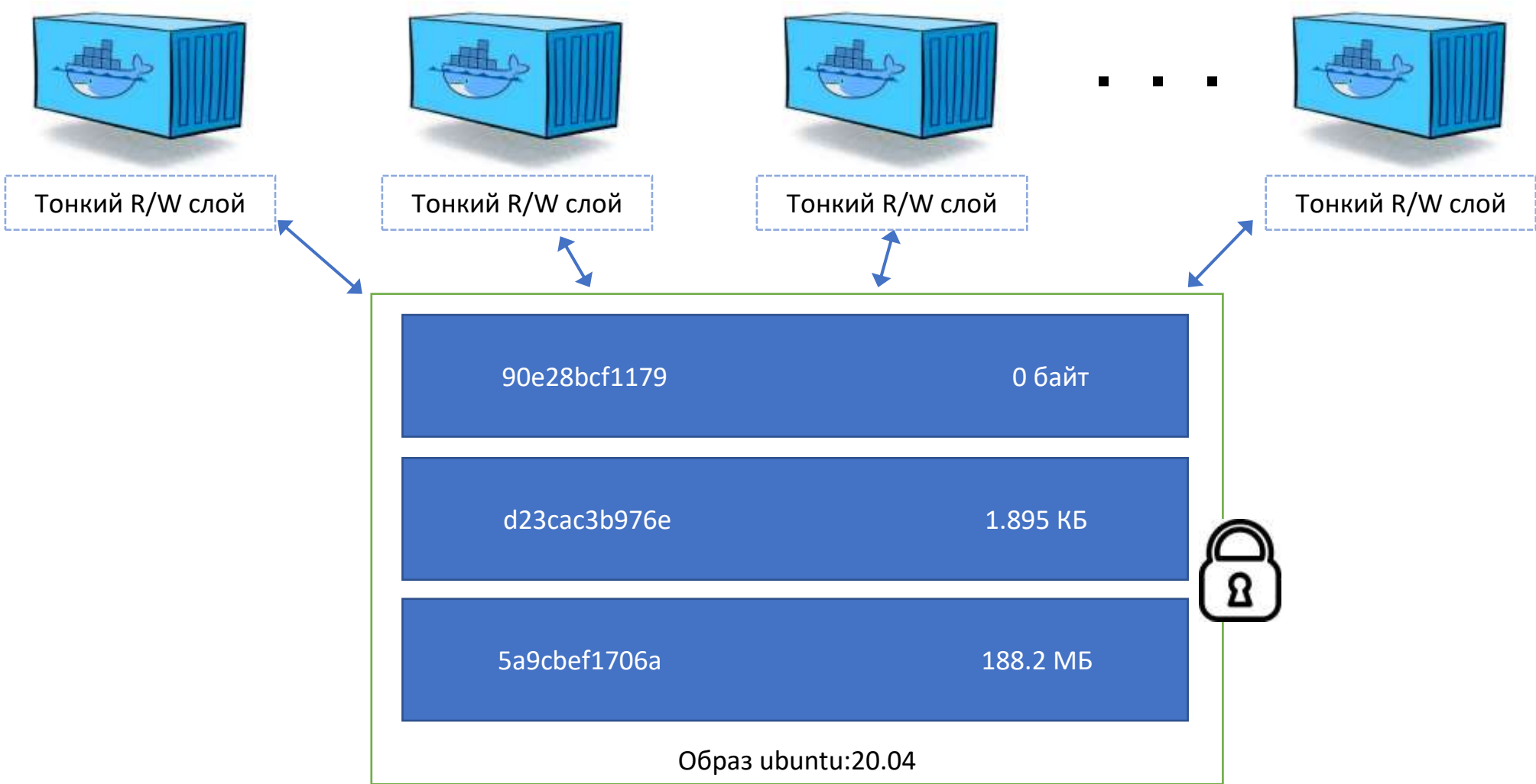


- Компания известная как DotCloud Inc
- (контейнеры в Linux)
- Создан Open Source-проект инструментария под названием Docker
- Сохраняли CarEx используя контейнеры вместо VM
- Получили деньги на развитие
- Инструментарий для управления контейнерами
- Сменили имя на Docker Inc

# Docker Engine



# Copy-on-Write



## Docker-образ

- Остановленный контейнер
  - Похоже на VM
- Состоит из нескольких слоев (read-only)
- Приложение поставляется в составе образа
- Контейнеры запускаются из образов
- Образы публикуются  
в хранилищах-репозиториях  
в составе реестра (т.к. DockerHub)

**Образы становятся  
контейнерами,  
когда запускаются  
на Docker Engine**

# Docker-контейнеры

## Стандарт упаковки приложений

- **Стандартные и воспроизводимые:** промышленный стандарт контейнеров, портируемых куда угодно
- **Легковесные:** Контейнеры используют ОС хозяина, повышая эффективность использования сервера и снижая затраты на серверы и лицензии
- **Безопасные:** лучшие условия изоляции, решение вопроса зависимостей

Под капотом



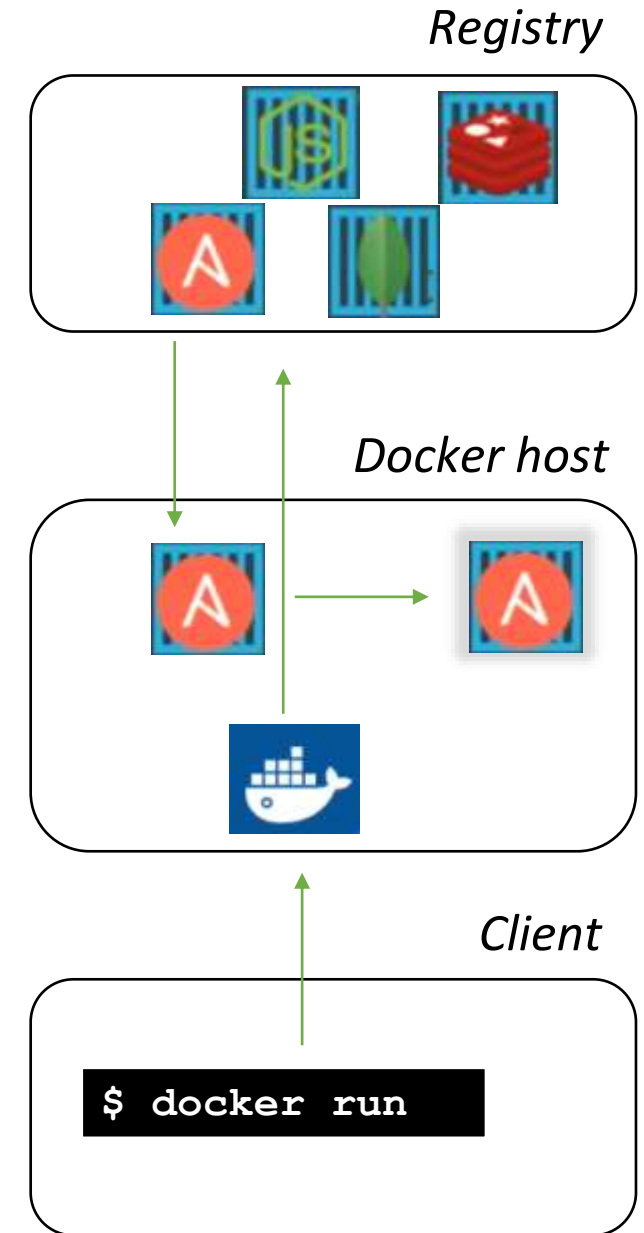
[Про containerd](#)

~\$ runc

[Про runc](#)

# Реестры Docker

- Хранилище Docker-образов
- Облачные репозитории:
  - DockerHub – по-умолчанию
  - GCR (Google Container Registry)
  - Amazon ECR
- Локальные или inhouse-репозитории
  - Nexus 3+
  - Jfrog Artifactory
  - DTR (Docker trusted Registry)





# Команды Docker

<https://docs.docker.com/engine/reference/commandline/cli/>

- `docker search` => Поиск образа в реестре
- `docker pull` => Скачать образ из реестра
- `docker build` => Собрать образ
- `docker images` => Список локальных образов
- `docker run` => Создать новый контейнер
- `docker ps` => Список запущенных контейнеров
- `docker ps -a` => Состояние всех контейнеров
- `docker exec` => Выполнить команду в контейнере
- `docker start/stop/restart` => Работа с контейнером
- `docker rm/rmi` => Удалить контейнер / образ
- `docker inspect` => Свойства контейнера / образа

# Контейнер и данные

- Данные не сохраняются, когда контейнер уничтожается
  - Как вытащить данные для обработки другим процессом?
- Записываемый слой контейнера тесно связан с машиной-ХОЗЯИНОМ
  - Как переместить данные в другое место?
- Есть два варианта хранения файлов:
  - Тома (Volumes)
    - Управляется Docker (в Linux это `/var/lib/docker/volumes/`)
  - Монтирование (Bind Mounts)
    - Хранится в любом месте, доступном с хоста

# Сборка образа и Dockerfile

```
1. FROM ubuntu:latest
2. ENV DEBIAN_FRONTEND=noninteractive
3. RUN apt update
4. CMD ["/usr/sbin
5. EXPOSE 80
6. WORKDIR /var/www
7. ADD nano.tar.gz
```



Dockerfile



docker build



Docker Image

Слой 1



Слой 2



Слой 3



Слой 4



Слой 5



Слой 6



Слой 7



*repo / image\_name : tag*

# Dockerfile - Инструкции

<https://docs.docker.com/engine/reference/builder/>

- FROM => Базовый образ
- LABELS => Добавить метаданные к образу
- RUN => Выполнить команды в новом слое (+commit)
- ADD/COPY => Добавить файлы и папки к образу
- CMD => Выполнить команды во время docker run
- ENTRYPOINT => Сделать контейнер запускаемым, задав точку входа
- VOLUME => Создать точку монтирования
- EXPOSE => Указать порт, который контейнер будет слушать

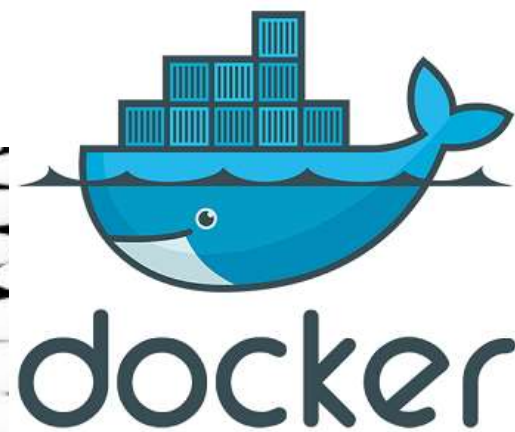
# Dockerfile - Инструкции

<https://docs.docker.com/engine/reference/builder/>

- ENV           => Задать переменную окружения
- USER           => Задать имя пользователя (UID)
- WORKDIR       => Задать рабочий каталог
- ARG           => Определить переменную, которую пользователь сможет передать во время сборки
- ONBUILD       => Добавить к образу триггер – инструкцию, которая будет выполнена, если на базе данного образа собирают другой.

Хорошие практики

# Тюнинг



## Best practice

- Тюнинг Dockerfile
- Размер образа
- Отключение кешей
- .dockerignore
- Установка и удаление в одном шаге
- Порядок выполнения шагов
- Multi-stage сборка
- Логи stdout/stderr

# export|import|load|save

```
$ docker --help | grep -E "(export|import|load|save)"
```

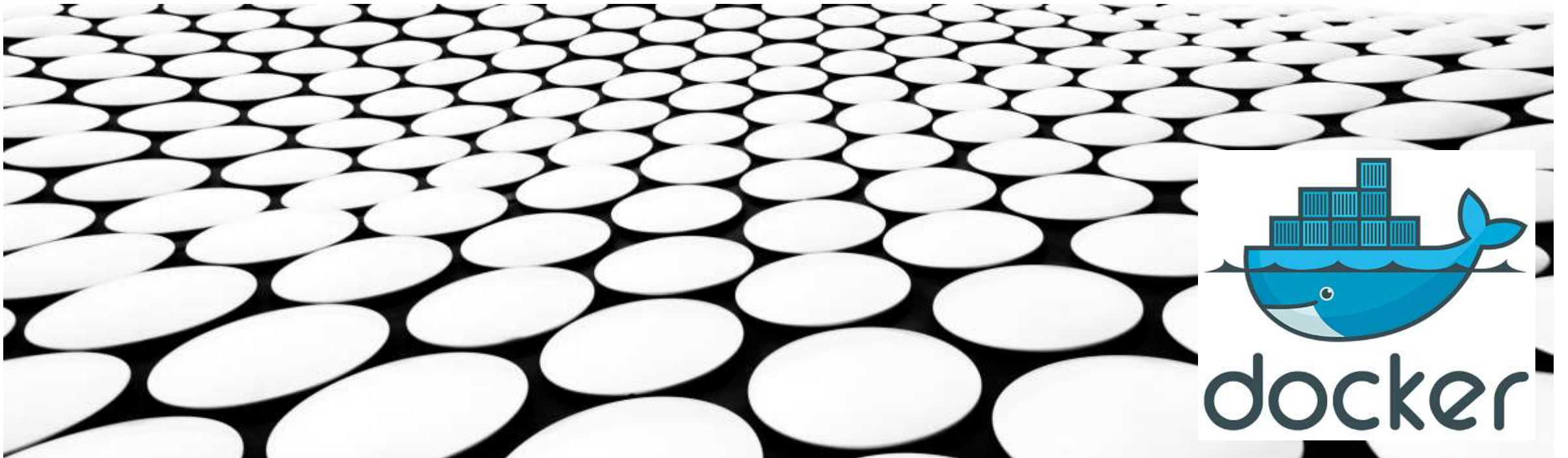
export	Export a container's filesystem as a tar archive
import	Import the contents from a tarball to create a filesystem image
load	Load an image from a tar archive or STDIN
save	Save one or more images to a tar archive (streamed to STDOUT by default)



# Развертывание проектов

Docker-compose

# Развертывание проектов



# Docker и Docker Compose

```
docker run ergotoro/my_docker_app
```

```
docker run mongodb
```

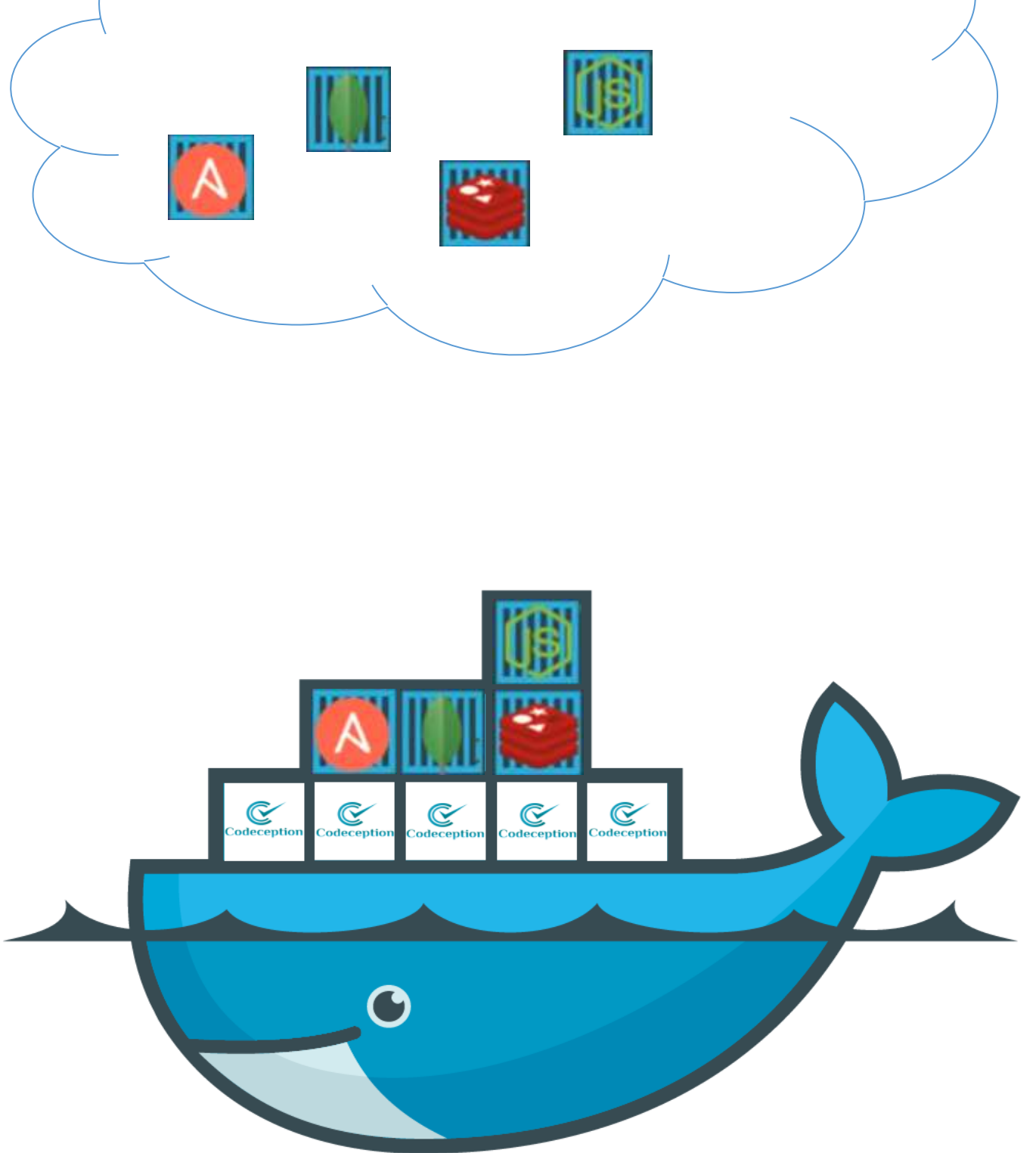
```
docker run redis:alpine
```

```
docker run ansible
```

## **docker-compose.yml**

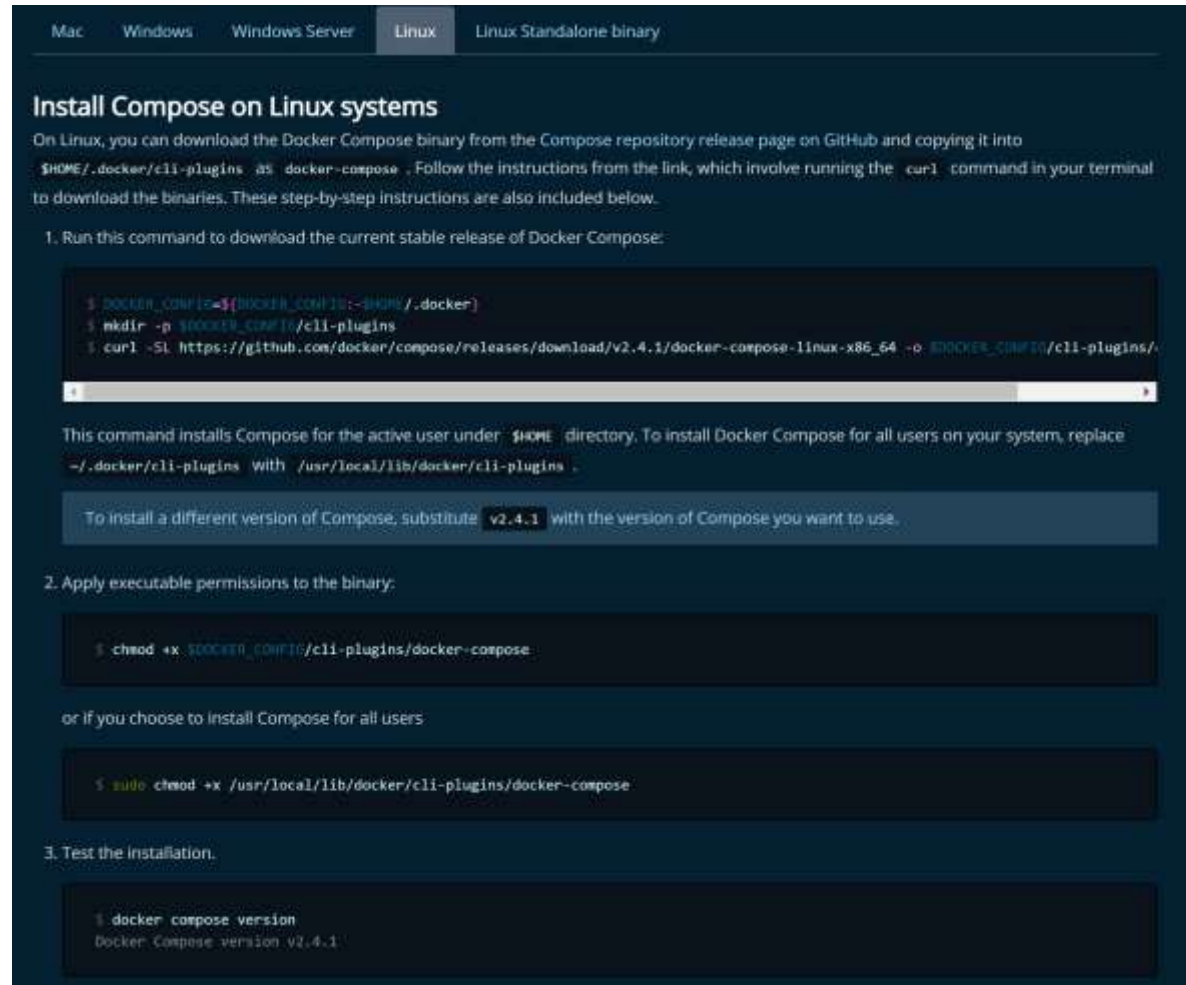
```
services:  
  web:  
    image: "ergotoro/my_docker_app"  
  database:  
    image: "mongodb"  
  messaging:  
    image: "redis:alpine"  
  orchestration:  
    image: "ansible"
```

```
docker-compose up
```



# Docker Compose: Установка

<https://docs.docker.com/compose/install/>



The screenshot shows the 'Linux' tab selected in the Docker documentation. The page title is 'Install Compose on Linux systems'. It provides instructions for downloading the Docker Compose binary from the GitHub repository release page. The instructions include a list of terminal commands to create a directory, download the binary, and apply executable permissions. A note indicates that the version 'v2.4.1' can be substituted with the desired version. The final step is to test the installation by running 'docker compose version', which returns 'Docker Compose version v2.4.1'.

Mac Windows Windows Server Linux Linux Standalone binary

## Install Compose on Linux systems

On Linux, you can download the Docker Compose binary from the Compose repository release page on GitHub and copying it into `$HOME/.docker/cli-plugins` as `docker-compose`. Follow the instructions from the link, which involve running the `curl` command in your terminal to download the binaries. These step-by-step instructions are also included below.

1. Run this command to download the current stable release of Docker Compose:

```
$ DOCKER_CONFIG=${DOCKER_CONFIG:-$HOME/.docker}
$ mkdir -p $DOCKER_CONFIG/cli-plugins
$ curl -SL https://github.com/docker/compose/releases/download/v2.4.1/docker-compose-linux-x86_64 -o $DOCKER_CONFIG/cli-plugins/docker-compose
```

This command installs Compose for the active user under `$HOME` directory. To install Docker Compose for all users on your system, replace `~/.docker/cli-plugins` with `/usr/local/lib/docker/cli-plugins`.

To install a different version of Compose, substitute `v2.4.1` with the version of Compose you want to use.

2. Apply executable permissions to the binary:

```
$ chmod +x $DOCKER_CONFIG/cli-plugins/docker-compose
```

or if you choose to install Compose for all users

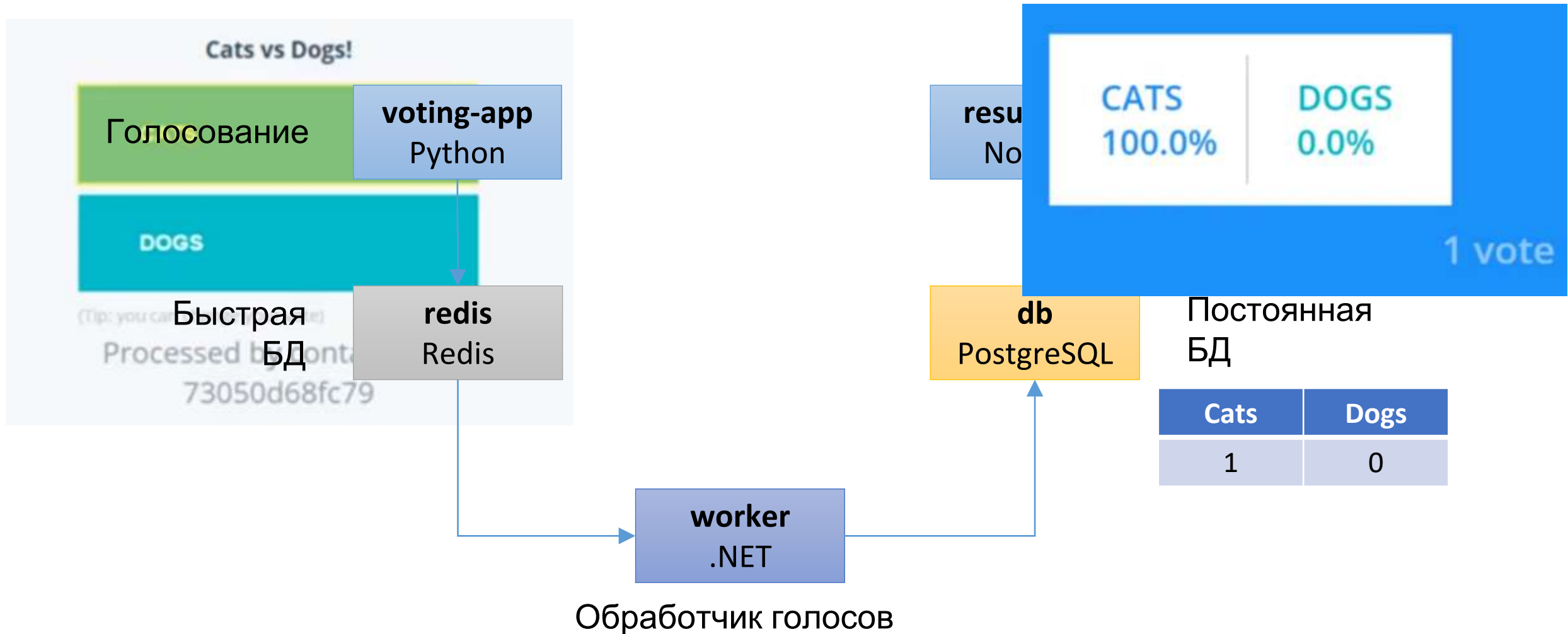
```
$ sudo chmod +x /usr/local/lib/docker/cli-plugins/docker-compose
```

3. Test the installation.

```
$ docker compose version
Docker Compose version v2.4.1
```

# Пример приложения – Voting App

<https://github.com/dockeramples/example-voting-app>



# Пример приложения – Voting App

```
docker run -d --name=redis redis
```

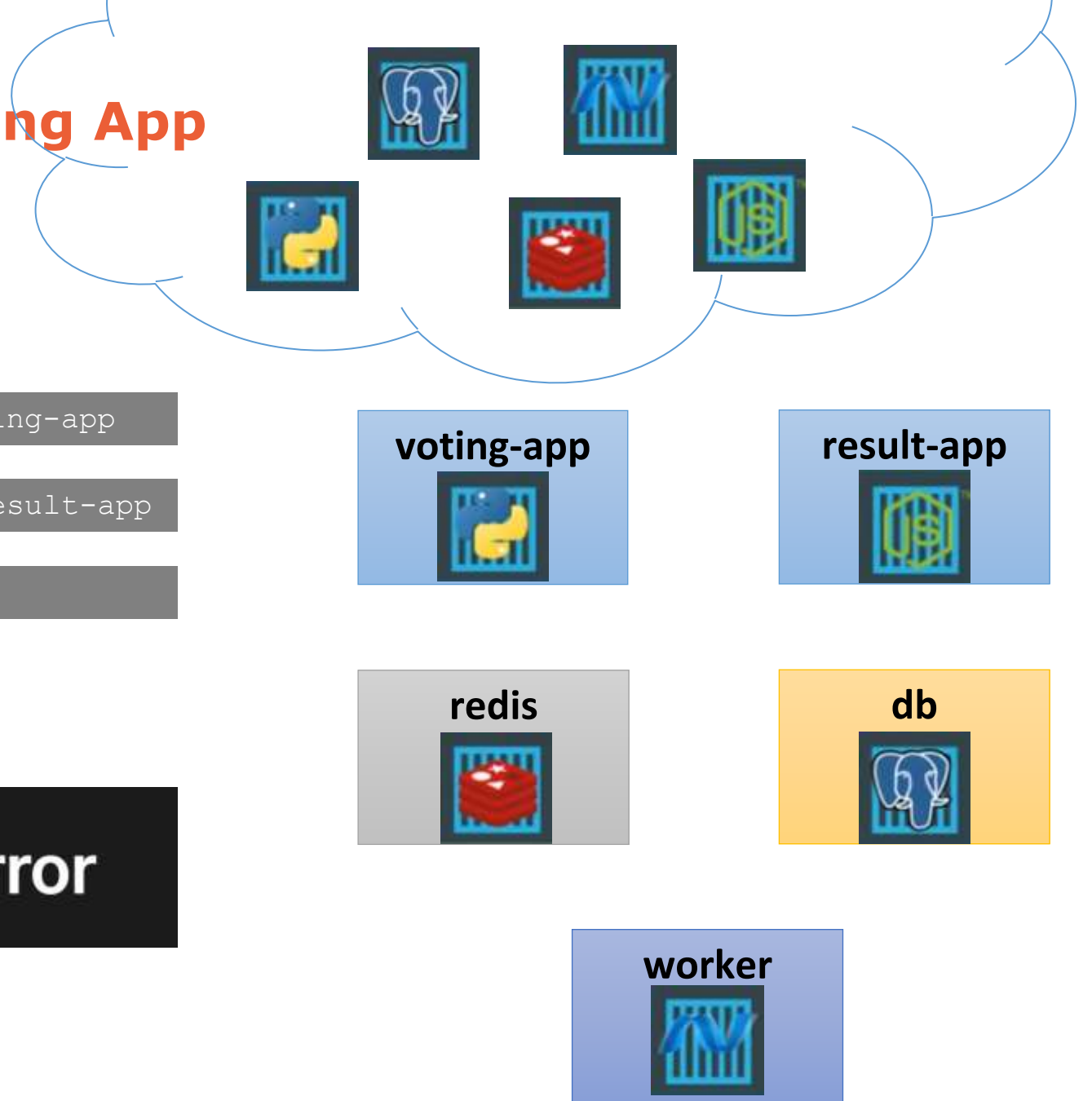
```
docker run -d --name=db postgres:9.4
```

```
docker run -d --name=vote -p 5000:80 voting-app
```

```
docker run -d --name=result -p 5000:80 result-app
```

```
docker run -d --name=worker worker
```

**500 Internal Server Error**



# docker run --link

```
docker run -d --name=redis redis
```

```
docker run -d --name=db postgres:9.4
```

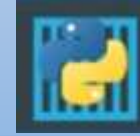
```
docker run -d --name=vote -p 5000:80 --link redis:redis voting-app
```

```
docker run -d --name=result -p 5000:80 result-app
```

```
docker run -d --name=worker worker
```

```
19 def get_redis():
20     if not hasattr(g, 'redis'):
21         g.redis = Redis(host="redis", db=0, socket_timeout=5)
22     return g.redis
```

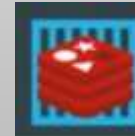
voting-app



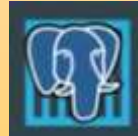
result-app



redis



db



worker



# docker run --link

```
docker run -d --name=redis redis
```

```
docker run -d --name=db postgres:9.4
```

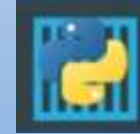
```
docker run -d --name=vote -p 5000:80 --link redis:redis voting-app
```

```
docker run -d --name=result -p 5001:80 --link db:db result-app
```

```
docker run -d --name=worker worker
```

```
26 var pool = new pg.Pool({
27   connectionString: 'postgres://postgres:postgres@db/postgres'
28 });
29
30 async.retry(
31   {times: 1000, interval: 1000},
32   function(callback) {
33     pool.connect(function(err, client, done) {
```

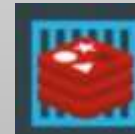
voting-app



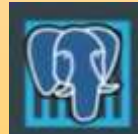
result-app



redis



db



worker





# docker run --link

```
docker run -d --name=redis redis
```

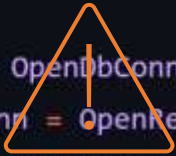
```
docker run -d --name=db postgres:9.4
```

```
docker run -d --name=vote -p 5000:80 --link redis:redis voting-app
```

```
docker run -d --name=result -p 5001:80 --link db:db result-app
```

```
docker run -d --name=worker --link redis:redis --link db:db worker
```

```
try
{
    var pgsql = OpenDbConnection("Server=db;Username=postgres;Password=postgres;");
    var redisConn = OpenRedisConnection("redis");
    var redis = redisConn.GetDatabase();
}
```



Работает, но устарело

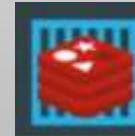
voting-app



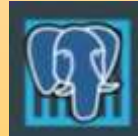
result-app



redis



db



worker



# docker-compose.yml

```
docker run -d --name=redis redis
```

```
docker run -d --name=db postgres:9.4
```

```
docker run -d --name=vote -p 5000:80 --link redis:redis voting-app
```

```
docker run -d --name=result -p 5001:80 --link db:db result-app
```

```
docker run -d --name=worker --link redis:redis --link db:db worker
```

```
docker-compose up
```

## docker-compose.yml

```
redis:
  image: "redis"
db:
  image: "postgres:9.4"
vote:
  image: "voting-app"
  ports:
    - 5000:80
  links:
    - redis
result:
  image: "result-app"
  ports:
    - 5001:80
  links:
    - db
worker:
  image: "worker"
  links:
    - redis
    - db
```

# Сеть в Docker

- Сети: **bridge, none, host**
- По-умолчанию: **docker0** (bridge)
- Сети изолированы и auto service discovery отсутствует
- Тип сети overlay – для multi-host network
  - Требуется синхронизация (Etcd, Zookeeper, ...)
- Работа с сетями:

```
docker network ls
```

```
docker network create -driver bridge my_own_network
```

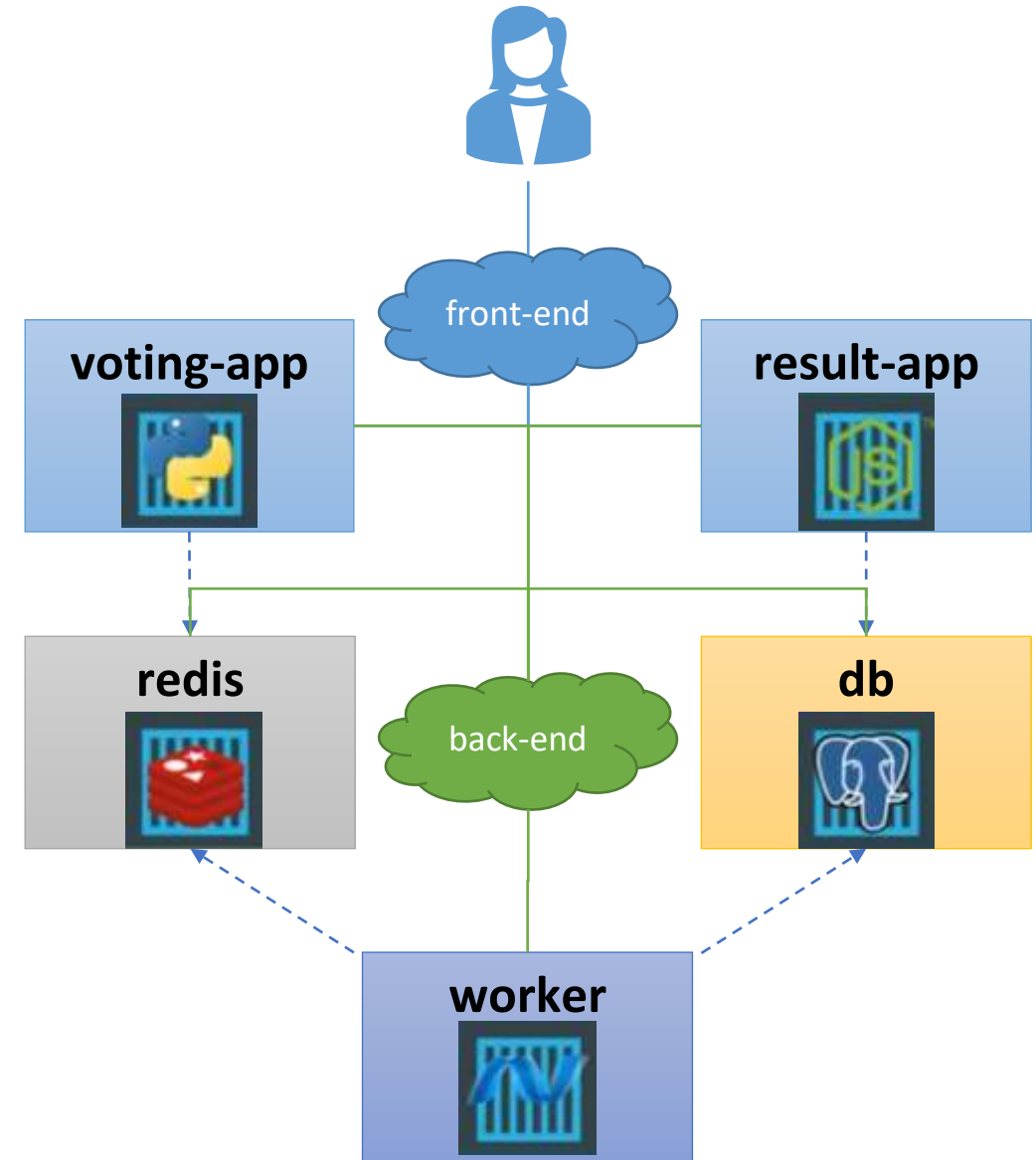
```
docker run --net=my_own_network ...
```

```
docker network inspect my_own_network
```

# Docker Compose: Сеть

**docker-compose.yml**

```
version: 2
services:
  redis:
    image: "redis"
    networks:
      - back-end
  db:
    image: "postgres:9.4"
    networks:
      - back-end
  vote:
    image: "voting-app"
    networks:
      - back-end
      - front-end
  result:
    image: "result"
    networks:
      - back-end
      - front-end
networks:
  back-end:
  front-end:
```



# Сеть в Docker Compose

<https://docs.docker.com/compose/compose-file/>

- По-умолчанию: одна сеть на проект (директорию)
  - `--project-name / COMPOSE_PROJECT_NAME`
- Другие сети – опция `external`

```
networks:  
  default:  
    name: an-existing-network  
    external: true
```

- Опции:

```
networks:  
  front-end:  
    driver: bridge  
    driver_opts:  
      com.docker.network.enable_ipv6: "true"  
    ipam:  
      driver: default  
      config:  
        - subnet: 172.16.238.0/24  
          gateway: 172.16.238.1  
        - subnet: "2001:3984:3989::/64"  
          gateway: "2001:3984:3989::1"
```

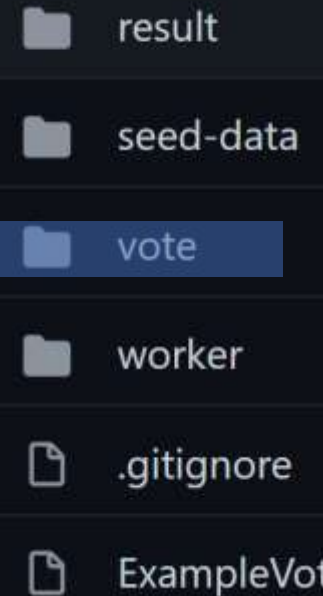
# Docker Compose: Build

## `docker-compose.yml`

```
redis:
  image: "redis"
db:
  image: "postgres:9.4"
vote:
  image: "voting-app"
  ports:
    - 5000:80
  links:
    - redis
result:
  image: "result-app"
  ports:
    - 5001:80
  links:
    - db
worker:
  image: "worker"
  links:
    - redis
    - db
```

## `docker-compose.yml`

```
redis:
  image: "redis"
db:
  image: "postgres:9.4"
vote:
  build: ./vote
  ports:
    - 5000:80
  links:
    - redis
result:
  build: ./result
  ports:
    - 5001:80
  links:
    - db
worker:
  build: ./worker
  links:
    - redis
    - db
```



A file explorer window showing the project structure. It contains a list of files and folders: 'result' (folder), 'seed-data' (folder), 'vote' (folder), 'worker' (folder), '.gitignore' (file), and 'ExampleVotingAp' (file). The 'vote' folder is highlighted with a blue selection bar.

- result
- seed-data
- vote
- worker
- .gitignore
- ExampleVotingAp

# Docker Compose: Версии

## `docker-compose.yml`

```
version: 1
services:
  redis:
    image: "redis"
  db:
    image: "postgres:9.4"
  vote:
    image: "voting-app"
    ports:
      - 5000:80
    links:
      - redis
```

**Версия 1.x**

## `docker-compose.yml`

```
version: 2
services:
  redis:
    image: "redis"
  db:
    image: "postgres:9.4"
  vote:
    image: "voting-app"
    ports:
      - 5000:80
    links:
      - redis
    depends_on:
      - redis
```

**Версия 2.x**

## `docker-compose.yml`

```
version: 3
services:
  redis:
    image: "redis"
  db:
    image: "postgres:9.4"
  vote:
    image: "voting-app"
    ports:
      - 5000:80
    links:
      - redis
```

**Версия 3.x**

## На практике

- Похожие, но разные:
  - `docker compose up / up -d`
  - `docker compose start / stop`
  - `docker compose down / down -v`
  - COPY vs ADD
  - ENTRYPOINT vs CMD
  - ENV vs ARG
- Проброс портов (8080:8080)
  - Лучше: 127.0.0.1:8080:8080
- Размер раздела /var



# Логи

[Configure logging drivers \(docker.com\)](https://docs.docker.com/logging/drivers/)

- Logging driver
  - по-умолчанию - JSON-файл
- Основные подходы:
  - Пассивный – отдаем на откуп приложению
  - Пишем в syslog
  - Отдаем во внешнюю систему (например, Splunk)
  - ELK или аналоги



СЕТЕВАЯ  
АКАДЕМИЯ  
ЛАНИТ

**Больше, чем обучение!**  
**[www.academy.ru](http://www.academy.ru)**