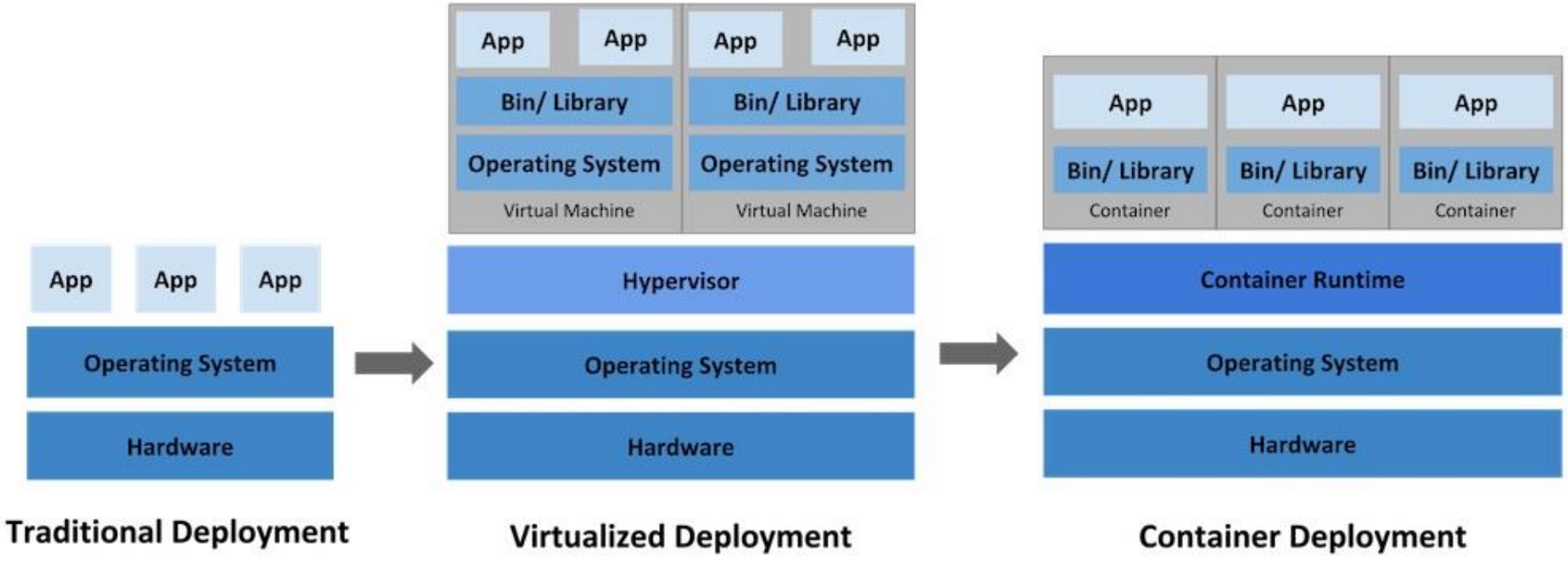
DOCKER & KUBERNETES

ПРЕДПОСЫЛКИ ВОЗНИКНОВЕНИЯ КОНТЕЙНЕРИЗАЦИИ



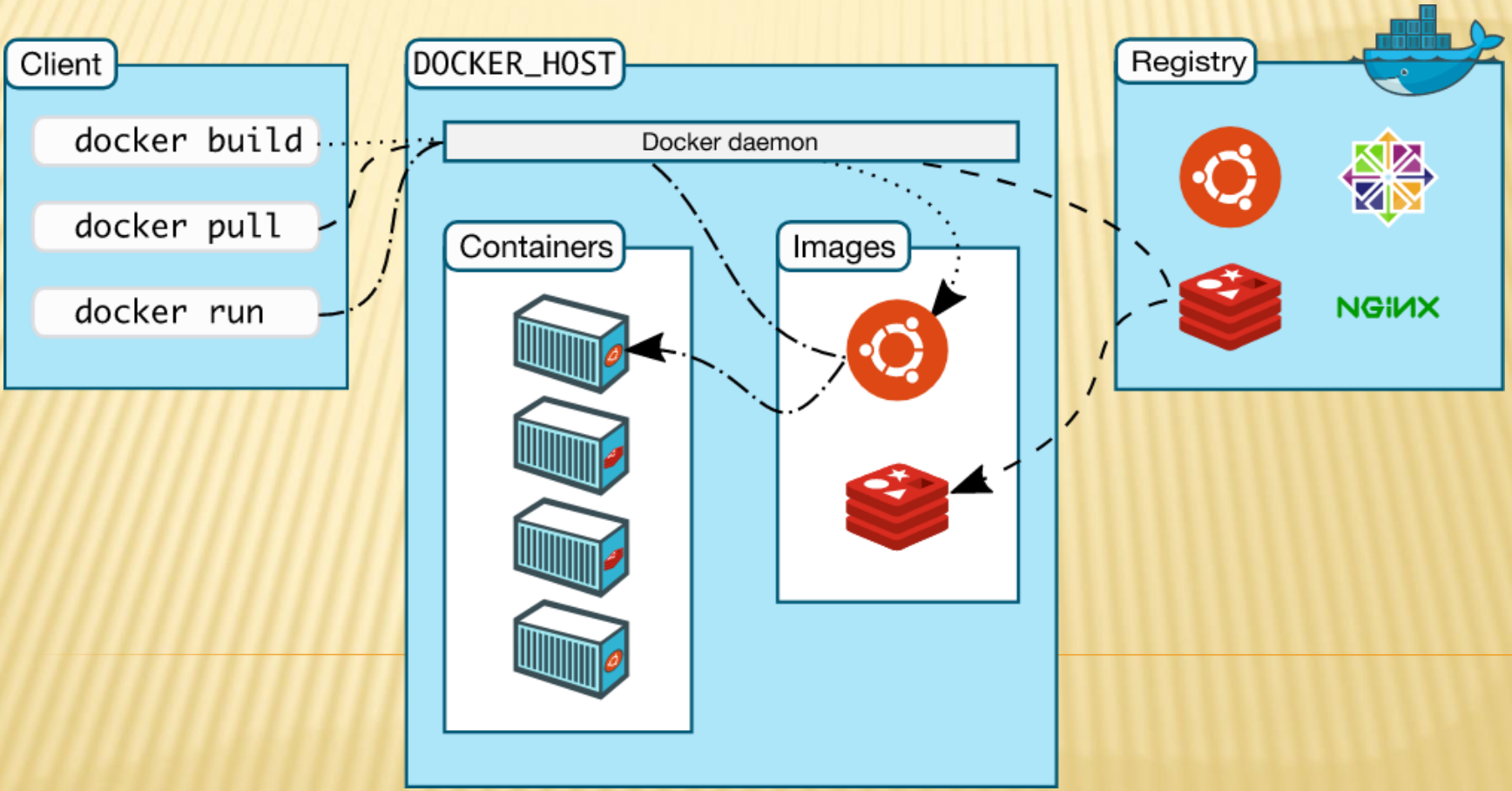
ОБЩИЕ ОПРЕДЕЛЕНИЯ

Docker – это средство виртуализации, одно из назначений которого виртуализация рабочих сред на серверах.

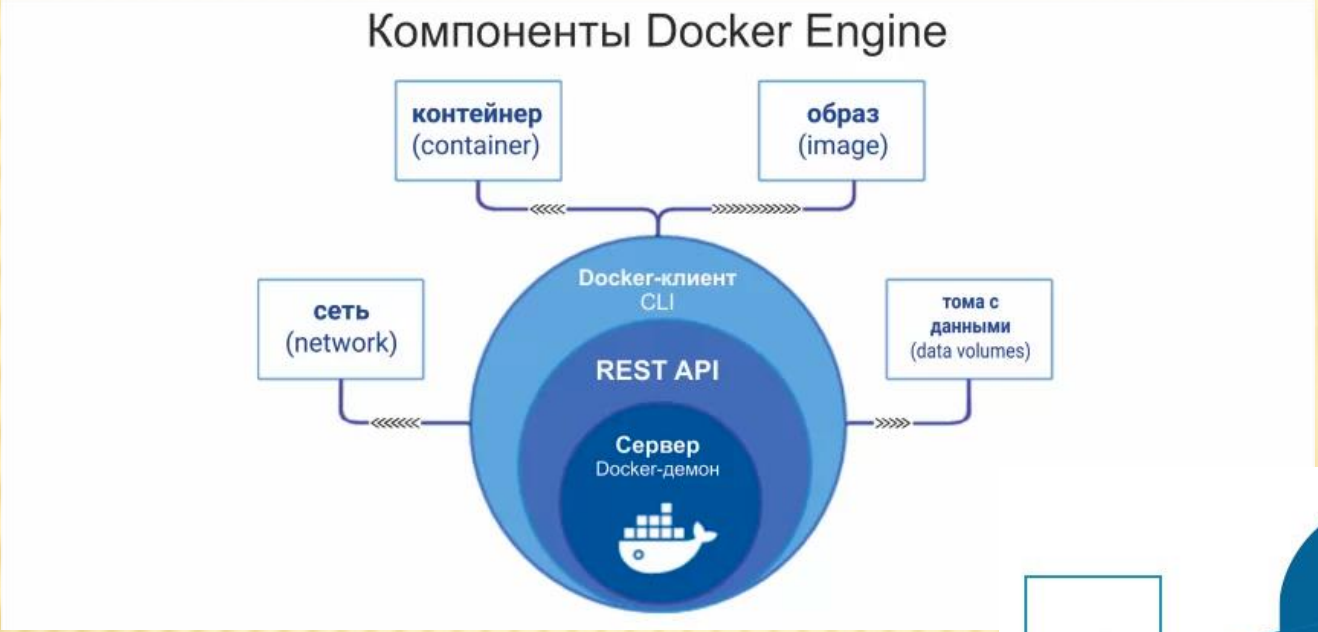
Образ — это read-only шаблон для создания Docker-контейнеров. Представляет собой исполняемый пакет, содержащий все необходимое для запуска приложения: код, среду выполнения, библиотеки, переменные окружения и файлы конфигурации.

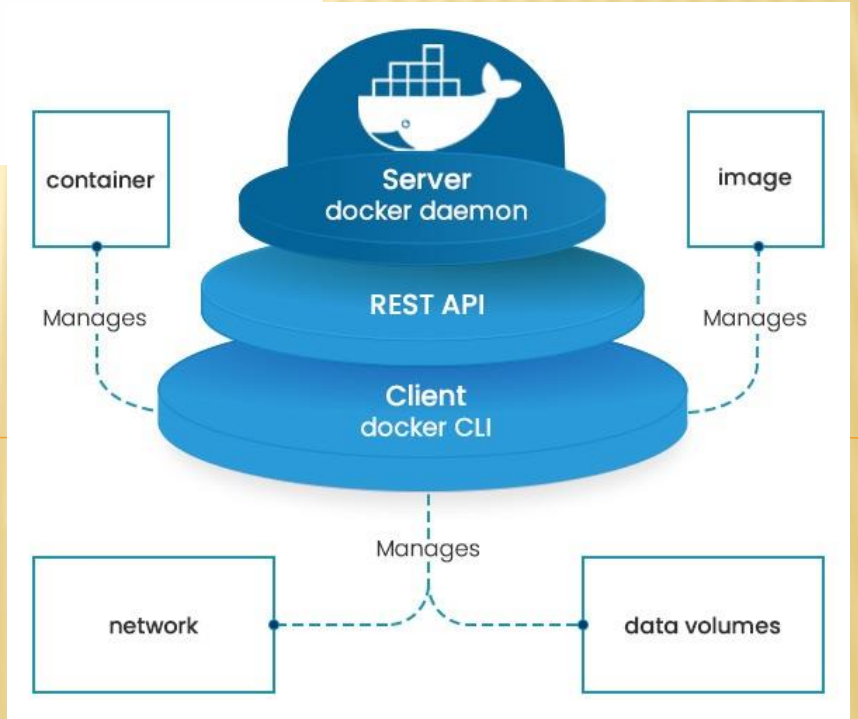
Контейнер — запущенный процесс операционной системы в изолированном окружении с подключенной файловой системой из образа. Контейнер видит свой собственный список процессов, свою собственную сеть, свою собственную файловую систему и т.д. Пока ему не укажут явно, он не может взаимодействовать с вашей основной операционной системой и всем, что в ней хранится или запущено.

АРХИТЕКТУРА DOCKER



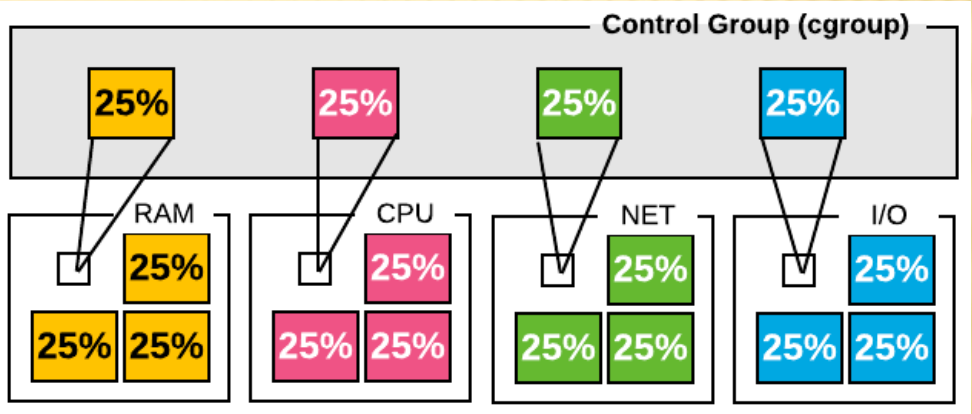
DOCKER ENGINE



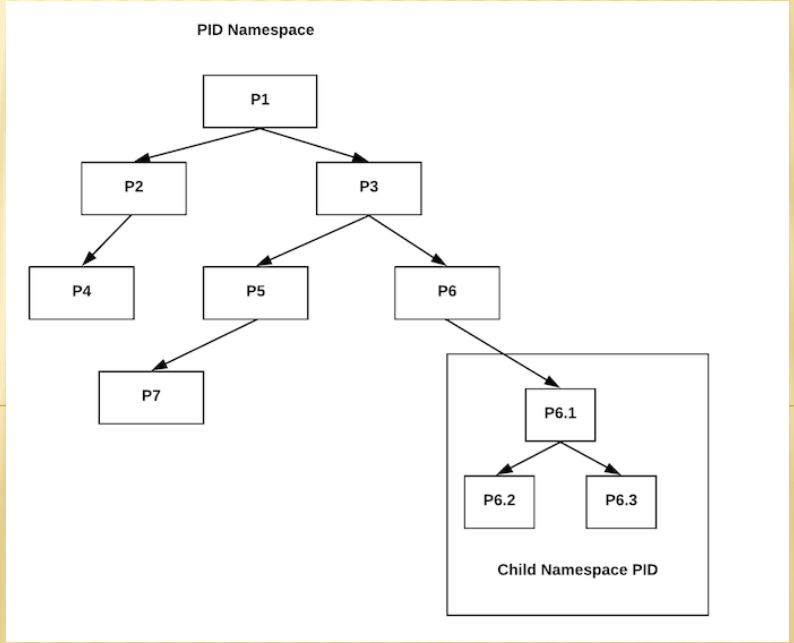


CGROUPS & NAMESPACES

cgroups – технология ядра, которая изолирует, приоритизирует и выдает квоты на использование ресурсов системы для группы процессов. С помощью этой технологии контейнеры docker получают только те ресурсы, которые были ему выделены.



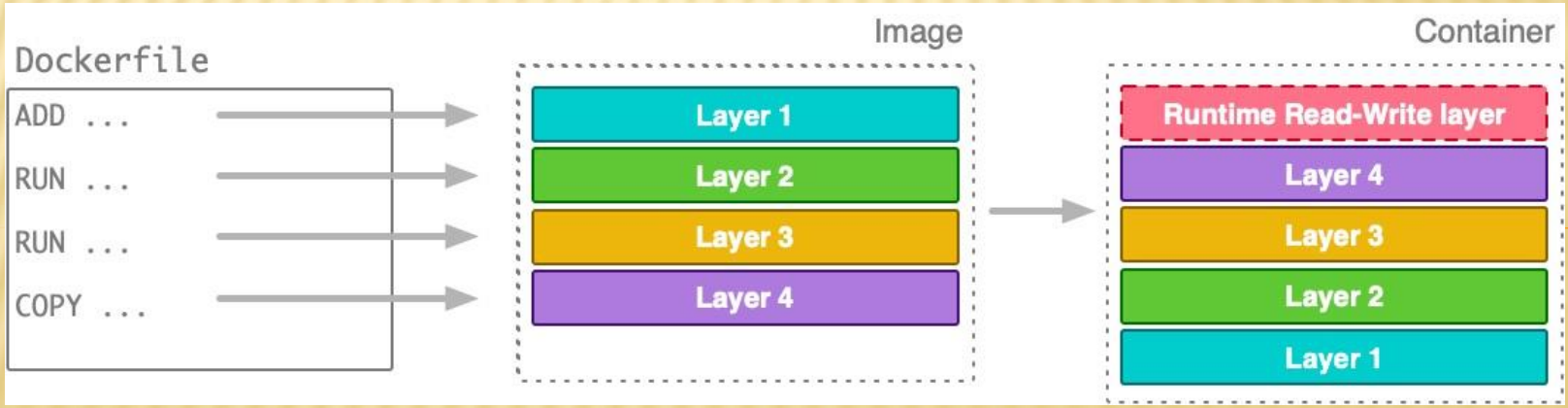
Namespaces –– это механизм ядра Linux, обеспечивающий изоляцию процессов друг от друга.



СТРУКТУРА ОБРАЗА

Образ состоит из слоев, каждый из которых представляет собой неизменяемую файловую систему, а по-простому набор файлов и директорий. Образ в целом представляет собой объединенную файловую систему (Union File System), которую можно рассматривать как результат слияния файловых систем слоев.

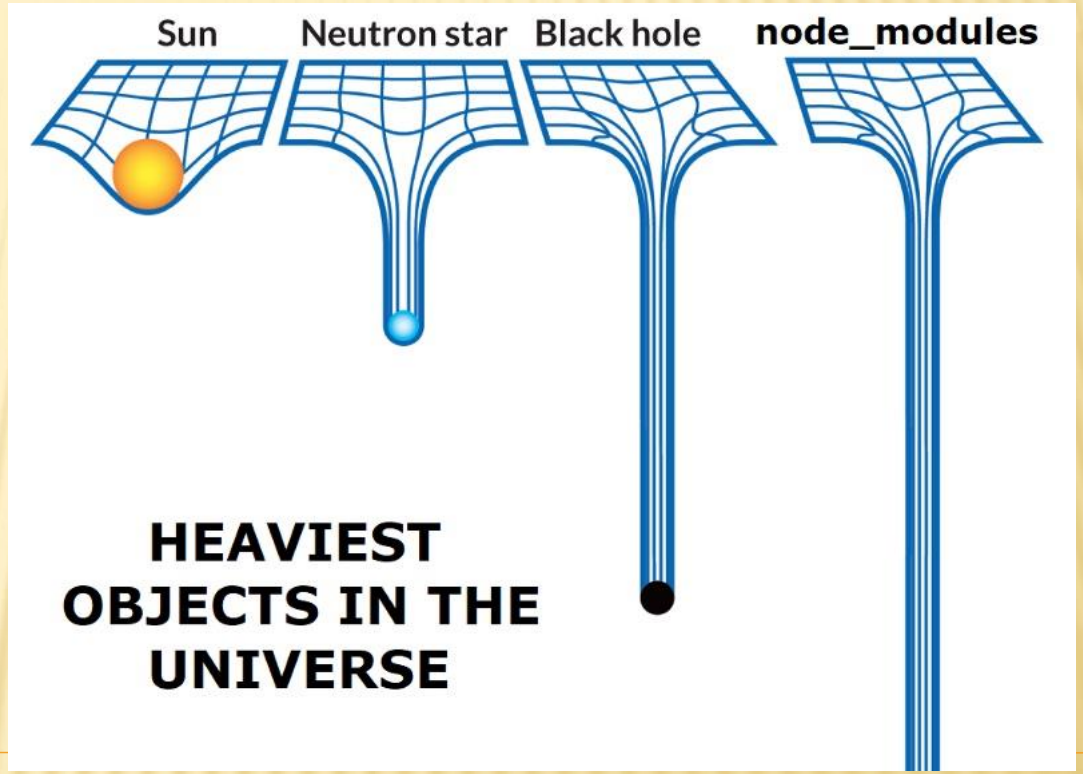
Каждый следующий слой добавляет или удаляет какие-то файлы из предыдущих слоев. В данном контексте "удаляет" можно рассматривать как "затирает", т.е. файл в нижележащем слое остается, но его не будет видно в объединенной файловой системе.



ОПТИМИЗАЦИЯ РАЗМЕРА ОБРАЗА

Каждая дополнительная инструкция в Dockerfile будет только увеличивать общий размер образа. Соответственно, чтобы уменьшить результирующий размер образа:

- нужно объединять однотипные команды;
- использовать максимально компактный базовый образ, например на базе Alpine Linux;
- использовать multistage build, чтобы не тащить в результирующий образ лишнее.



ОСНОВНЫЕ КОМАНДЫ DOCKER

```
# запустить контейнер postgres:13 на порту 5432, данные сохранять в локальный volume с именем postgres-data и создать пользователя test:test и базу example
$ docker run \
    --name postgres \
    -p 5432:5432 \
    -e POSTGRES_USER=test \
    -e POSTGRES_PASSWORD=test \
    -e POSTGRES_DB=services \
    -v postgres-data:/var/lib/postgresql/data \
    postgres:13

# сборка образа frontend в папке example/frontend
$ docker build examples/frontend -t my-frontend:v1.0

# вывод всех образов
$ docker images

# вывод всех volumes
$ docker volume ls

# просмотр информации о конкретном volume
$ docker volume inspect postgres-data

# вывод логов контейнера
$ docker logs postgres
```

ОСНОВНЫЕ КОМАНДЫ DOCKER

```
# просмотр всех сведение о контейнере
$ docker inspect postgres

# получение внутреннего ip-адреса контейнера
$ docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' postgres

# получение пути к папке с логами контейнера
$ docker inspect --format='{{.LogPath}}' postgres

# заход внутрь образа
$ docker exec --interactive --tty postgres /bin/bash

# остановка, старт и рестарт контейнера
$ docker start/restart/stop postgres

# информация о потребляемых ресурсах docker
$ docker stats

# удаление контейнера postgres
$ docker rm postgres

# показать запущенные процессы в контейнере
$ docker top postgres
```

DOCKER FILE

```
# Указывает с какого образа брать сборку.
FROM ubuntu:20.04

# Указывает от какого пользователя и группы запускаются команды.
USER ronin:staff

# Информирует что образ слушает порт 8080 по протоколу tcp.
EXPOSE 8080/tcp

# Указывает из какой директории выполнять дальнейшие инструкции.
# Если указан относительный путь, он будет применяться относительно предыдущих инструкций WORKDIR.
WORKDIR application

# Определяет перменную, оторую пользователь может передать при запуске `docker build --build-arg <varname>=<value>`.
ARG PROFILE=docker

# Копирует файлы с host-машины в образ, доступны wildcards, --chmod user:user.
# ADD умеет распаковывать архивы, но в документации советуют использовать COPY, где магия ADD не требуется.
ADD https://github.com/Netflix/eureka/archive/refs/tags/v1.10.17.zip /app/eureka
COPY build/libs/order-service.jar /app/order-service.jar
```

DOCKER FILE

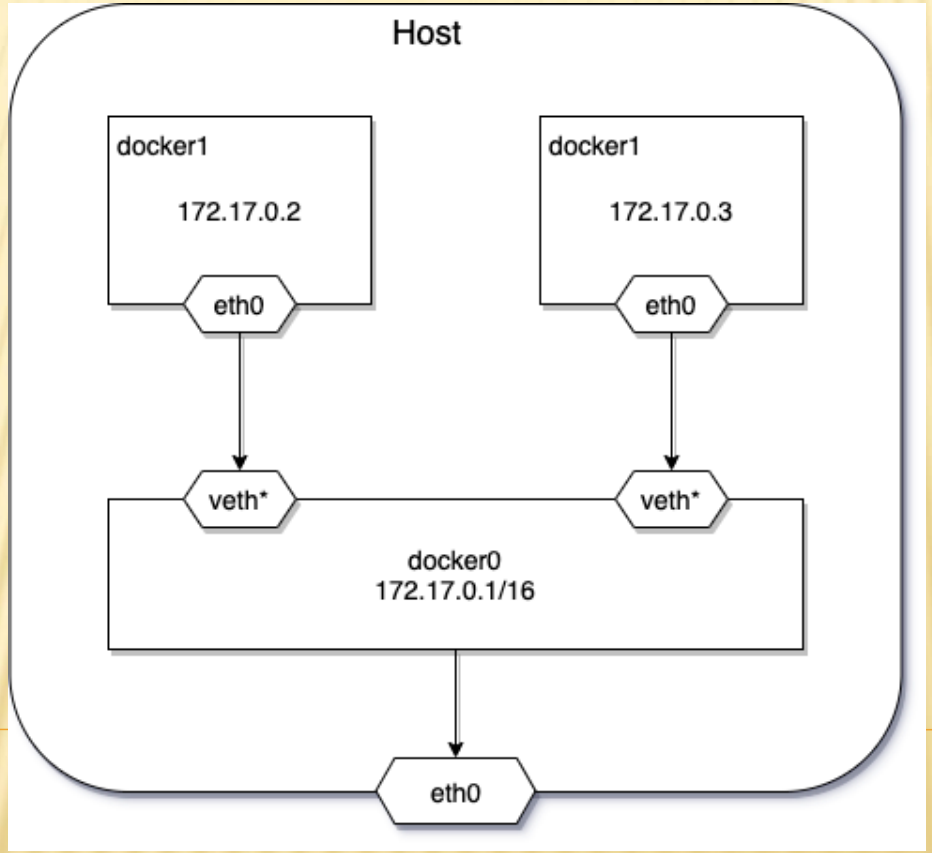
```
# Задает переменные окружения. Переменная окружения будет определена для запущенного контейнера
ENV SPRING_PROFILES_ACTIVE=$PROFILE

# выполняет каждую команду в новом слое поверх текущего слоя.
RUN apt-get update && \
  apt-get install openjdk-11-jdk -y

# Задает дефолтную команду при старте. Одна команда на контейнер и может быть перезаписана при старте контейнера.
CMD ["bash", "-c", "echo $HOME"]

# Позволяет описывать контейнер как исполняемый.
ENTRYPOINT ["java", "org.springframework.boot.loader.JarLauncher"]
```

СЕТЕВОЕ ВЗАИМОДЕЙСТВИЕ



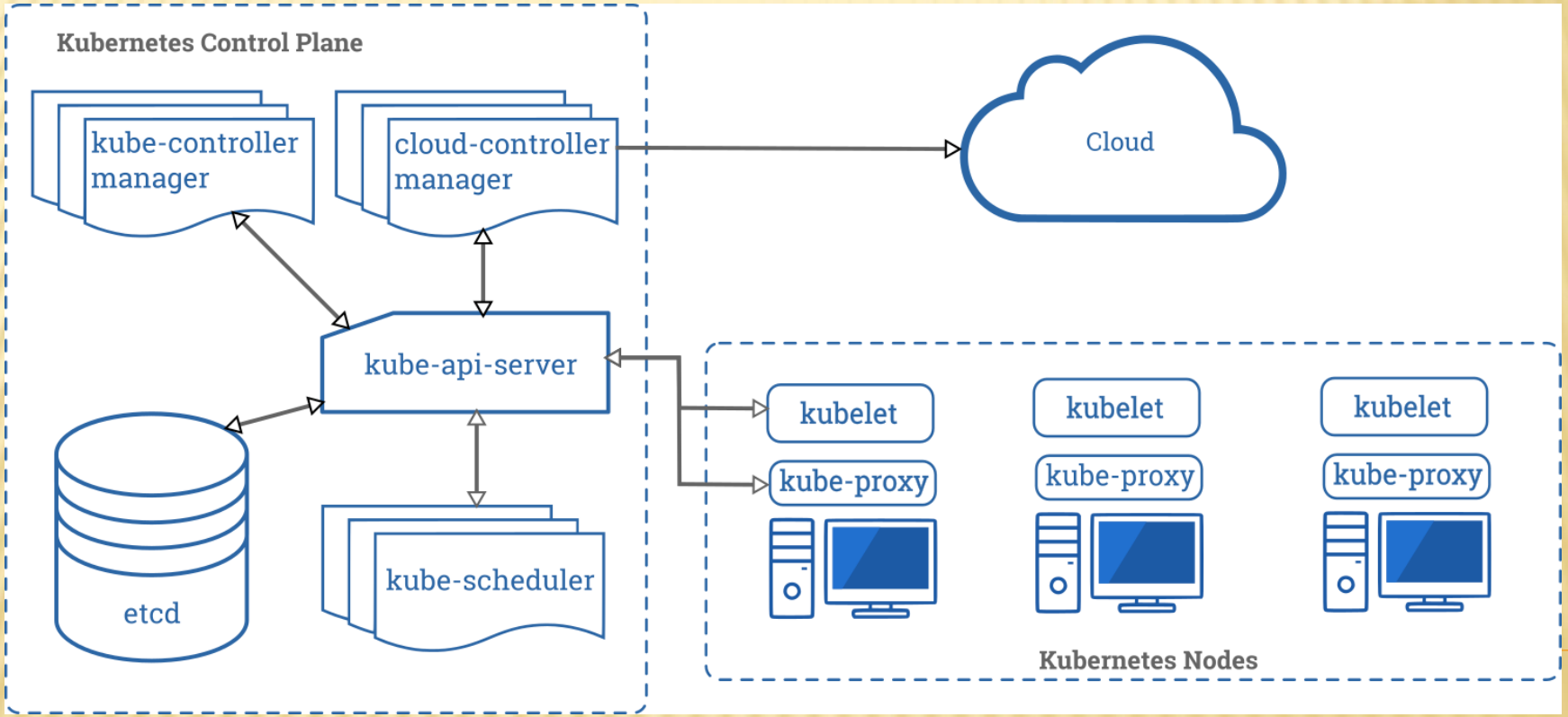
KUBERNETES

Kubernetes — это портативная расширяемая платформа с открытым исходным кодом для управления контейнеризованными рабочими нагрузками и сервисами, которая облегчает как декларативную настройку, так и автоматизацию.

Kubernetes предоставляет:

- Мониторинг сервисов и распределение нагрузки.
- Автоматическое развертывание и откаты.
- Автоматическое распределение нагрузки.
- Контроль состояния развертывания.
- Управление конфиденциальной информацией и конфигурацией.

СТРУКТУРА КЛАСТЕРА



ОСНОВНЫЕ ОБЪЕКТЫ

- ***Pod*** – минимальная сущность для развертывания в кластере. Каждый Pod предназначен для запуска одного (обычно) экземпляра конкретного приложения.
- ***Service*** – абстракция, которая определяет логический набор Pod'ов и политику доступа к ним, как сетевой сервис.
- ***Volume*** – персистентное хранилище данных внутри кластера. могут использоваться как volume для конфигурирования приложения.
- ***Namespace*** – это виртуальные кластеры размещенные поверх физического.
- ***Secrets*** используются для хранения конфиденциальной информации.

ОСНОВНЫЕ ОБЪЕКТЫ

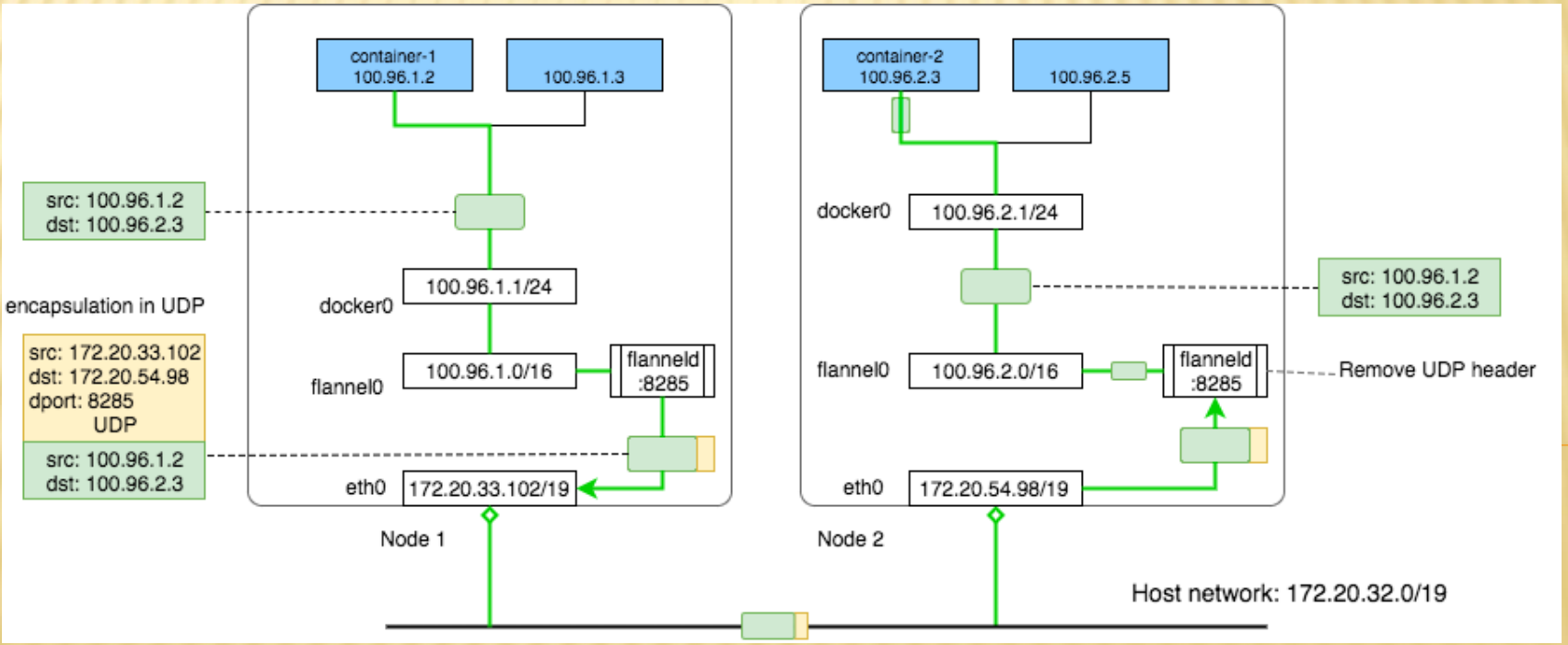
- ***Deployment*** обеспечивает декларативные обновления для Pods и ReplicaSets.
- ***DaemonSet*** гарантирует, что определенный Pod будет запущен на всех нодах.
- ***StatefulSet*** используется для управления приложениями с сохранением состояния.
- ***ReplicaSet*** гарантирует, что определенное количество экземпляров Pod будет запущено в кластере в любой момент времени.

ОСНОВНЫЕ ОБЪЕКТЫ

- ***Labels*** используются для маркирования объектов кластера, а так же для выбора этих объектов.
- ***ConfigMaps*** – абстракция над файлами конфигурации, позволяет разделять настройки приложения и сами контейнеры, избавляя от необходимости упаковывать конфиги в docker-образ.
- ***Annotations*** используются для добавления собственных метаданных к объектам.

OVERLAY NETWORK

В основе сетевого устройства Kubernetes — у каждого пода свой уникальный IP. IP пода делится между всеми его контейнерами и является доступным (маршрутизируемым) для всех остальных подов. На каждой машине есть сетевой интерфейс eth0, внутри пода тоже есть eth0, на host-машине они подключены к интерфейсу vethxxx. Эти интерфейсы общаются с eth0 через ethernet bridge интерфейс cni0 (docker использует аналогичный docker0).
Взаимодействие между узлами реализуется либо посредством ARP-запросов (L2), либо с помощью таблицы роутинга (ip-маршрутизация, L3). Для более гибкой маршрутизации строятся overlay-сети. Overlay-сеть выглядит как единая сеть между нодами.



КОНВЕРТИРОВАНИЕ СУЩЕСТВУЮЩИХ DOCKER ФАЙЛОВ В МАНИФЕСТЫ K8S

```
$ kompose convert --controller deployment --out k8s/ --with-kompose-annotation=false
WARN Service "simple-backend" won't be created because 'ports' is not specified
INFO Kubernetes file "k8s/simple-frontend-service.yaml" created
INFO Kubernetes file "k8s/simple-backend-deployment.yaml" created
INFO Kubernetes file "k8s/simple-frontend-deployment.yaml" created
```