Технология контейнеризации. Введение в Docker

Виртуализация

- Зачем нужна виртуализация?

Оптимизация

- По “железным мощностям” (в настоящем)
- По стоимости (в будущем)

Изоляция

- От чужих зависимостей
- От других приложений
- От сторонних пользователей

Эмуляция

- Другая ОС (Windows, Linux, Solaris…)
- Другая платформа (ARM, MIPS…)

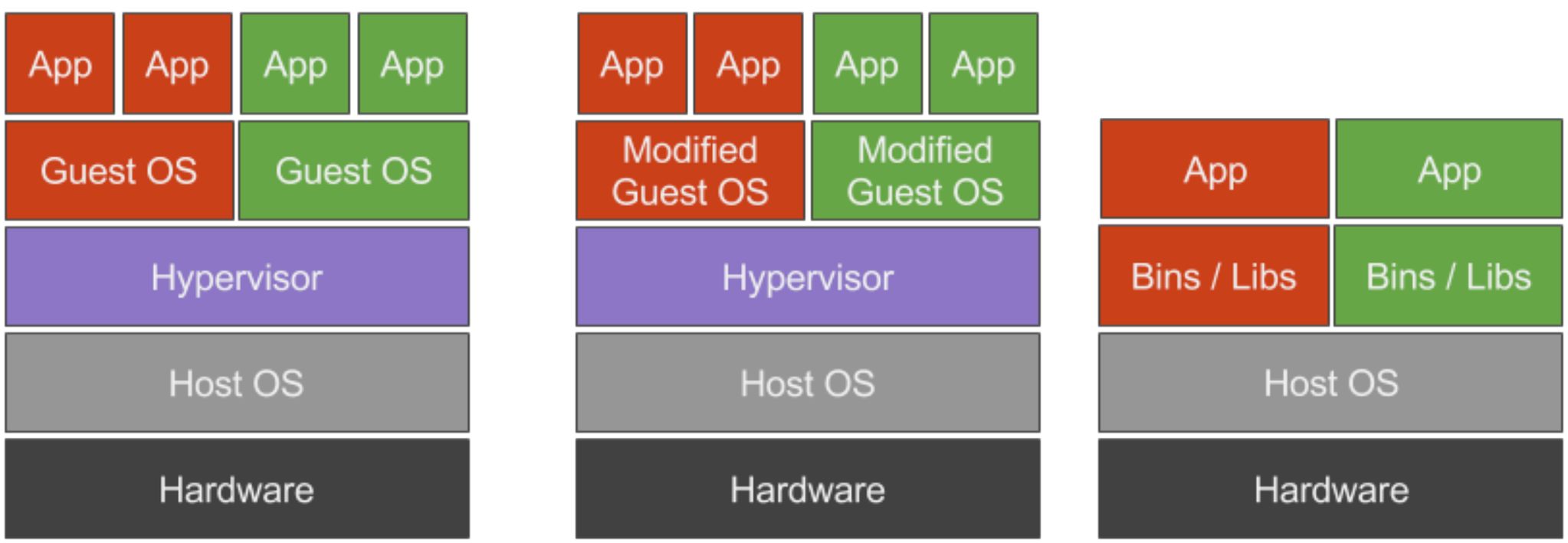
Типы виртуализации

- Аппаратная виртуализация (VMware, VirtualBox)
- Паравиртуализация (Xen)

Контейнеризация

- Контейнеризация (LXC, OpenVZ, Jail, Zones)

Типы виртуализации



Издержки виртуализации (CPU)

- Аппаратная виртуализация 5-15%
- Паравиртуализация 3-10%
- Контейнеризация 0.1-1%

Контейнеризация

- Существовала достаточно давно
- Не получила широкого распространения
- В определенных случаях заменила аппаратную виртуализацию
- Почему выстрелил именно Docker?

Docker

- Не столько про контейнеры (как технологию)
- Хотя использует контейнеризацию как основу

Docker

- Абстракция от host-системы
- Легковесное изолированное окружение
- Общие слои файловой системы
- **Компоновка и предсказуемость**
- **Простое управление зависимостями**
- **Дистрибуция и тиражируемость**

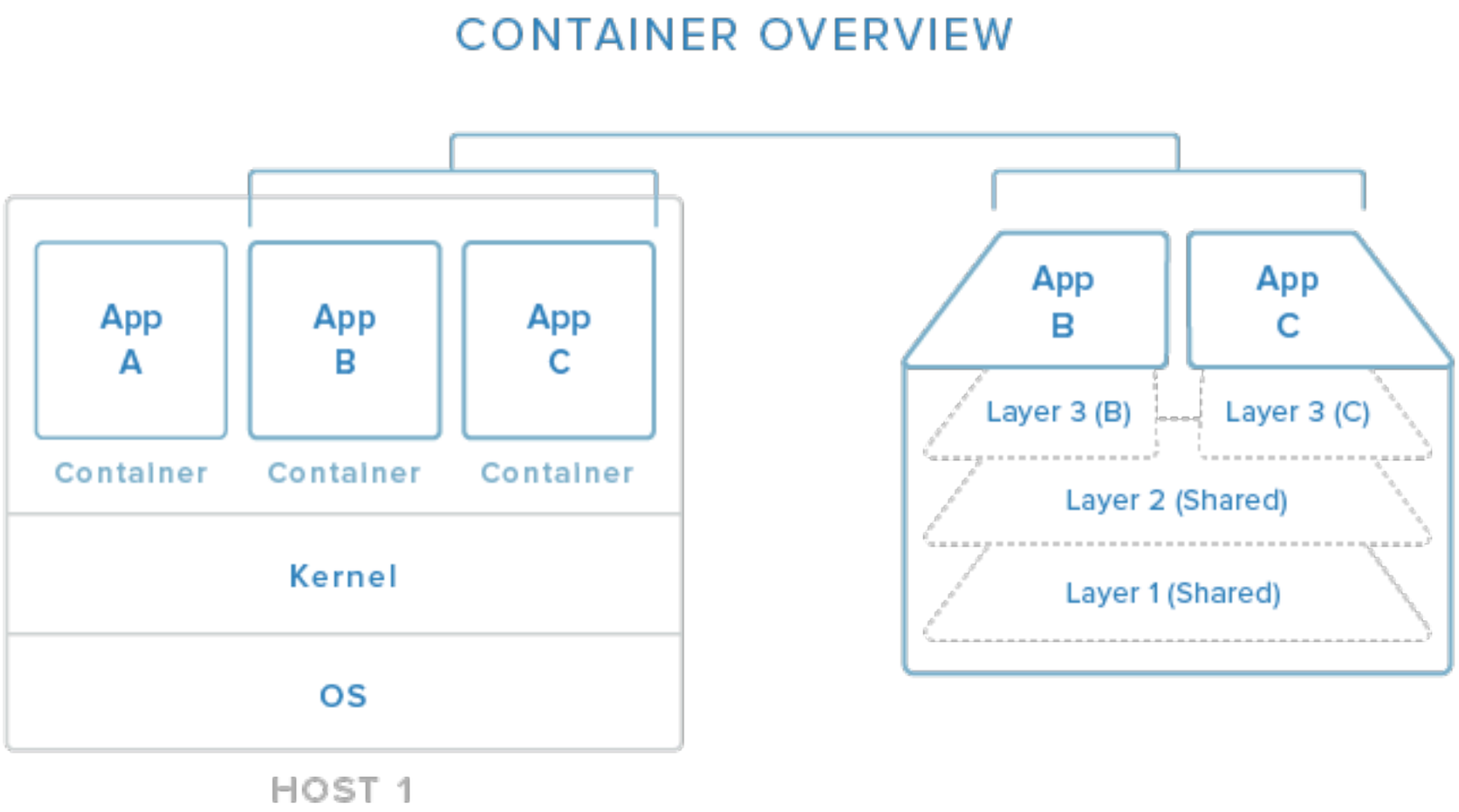
Docker это про стандарты

- Стандартизация описания окружения, сборки, деплоймента
- Стандартизированная дистрибуция
- 100% консистентная среда приложения
- Воспроизводимость (DEV->QA->Production)

Docker

- Контейнер - приложение и его зависимости упакованные в стандартизированное, изолированное, легковесное окружение.

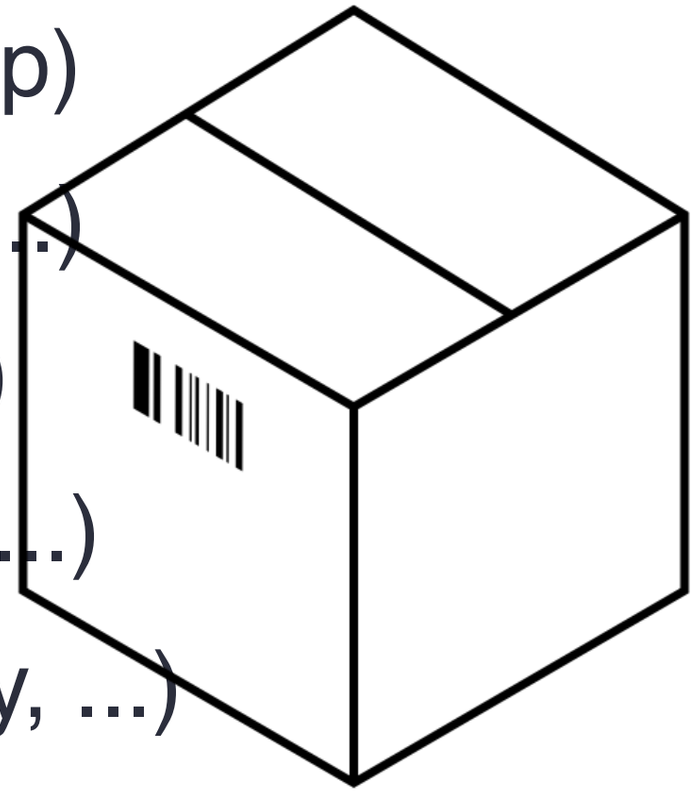
Docker


CONTAINER OVERVIEW
App A
App B
App C
Container
Container
Container
Kernel
OS
HOST 1
App B
App C
Layer 3 (B)
Layer 3 (C)
Layer 2 (Shared)
Layer 1 (Shared)

Стандартизация контейнеров

Docker:

- App (your Java/Ruby/Go/... app)
- Libraries (libxml, wkhtmltopdf, ...)
- Services (postgresql, redis, ...)
- Tooling (sbt, ant, gems, eggs, ...)
- Frameworks&runtime (jre, ruby, ...)
- OS packages (libc6, tar, ps, bash, ...)



Docker

- Как он работает?
- Daemon
- Client
- Registry

Docker daemon

- Предоставляет API
- Управляет Docker-объектами
- Общается с другими docker daemon'ами

Docker daemon

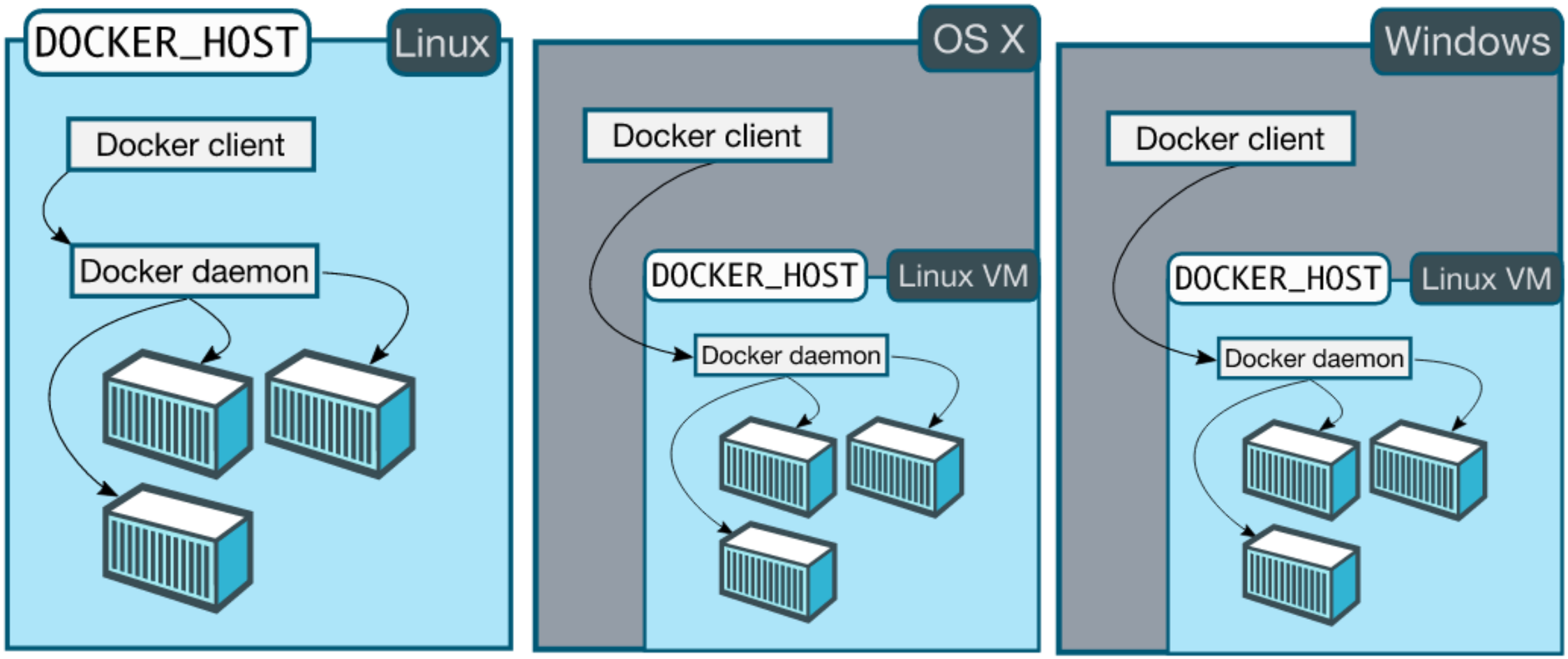
- Запускается на хост машине, где планируется запускать контейнеры

- Хост машина – vm, физический сервер(x86, arm64), aws ec2, ваш ноутбук, raspberry pi …

Docker client

- Принимает команды пользователя
- Общается по API с docker daemon’ом
- Может общаться с несколькими daemon’ами

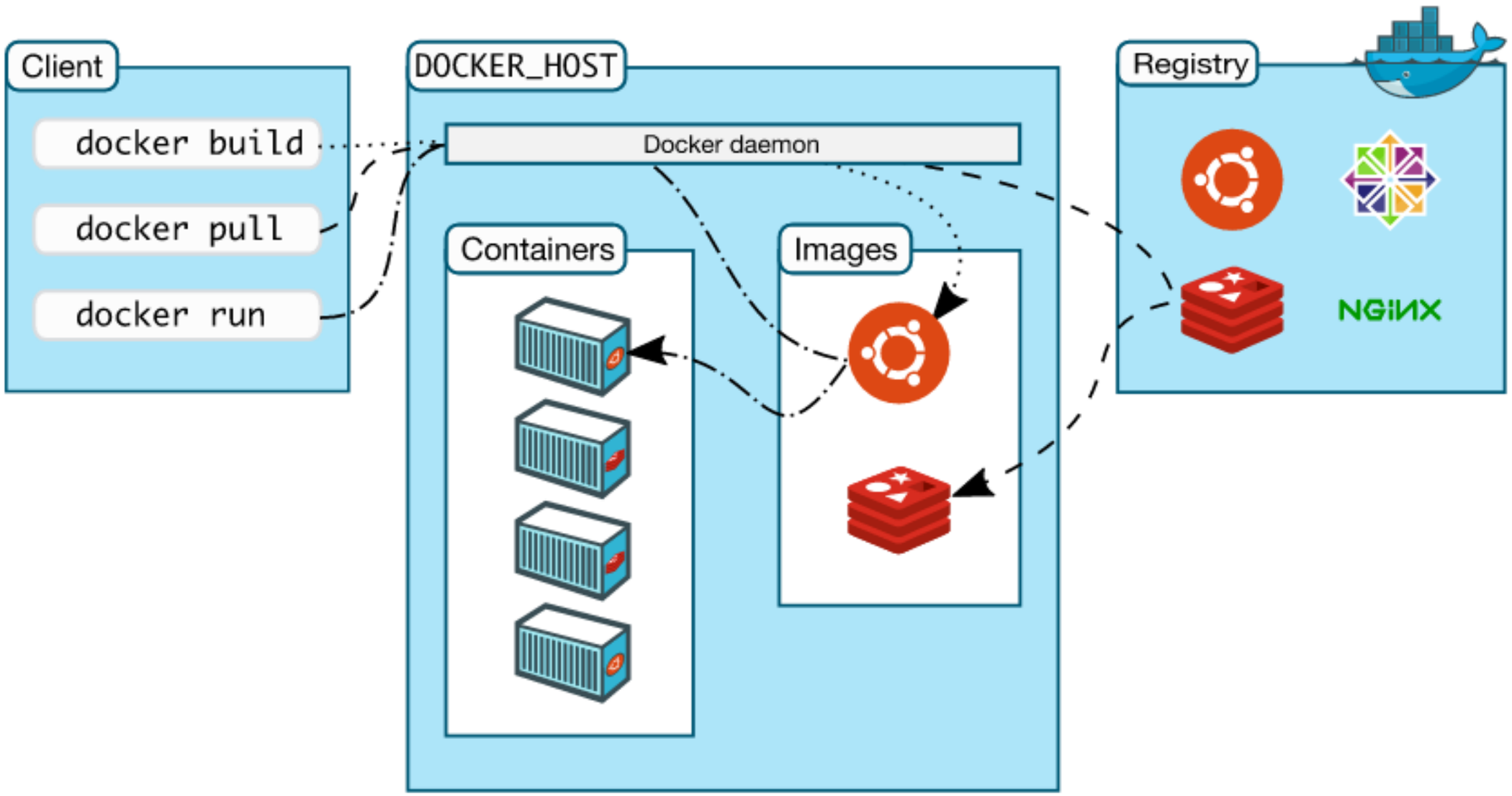
Docker engine



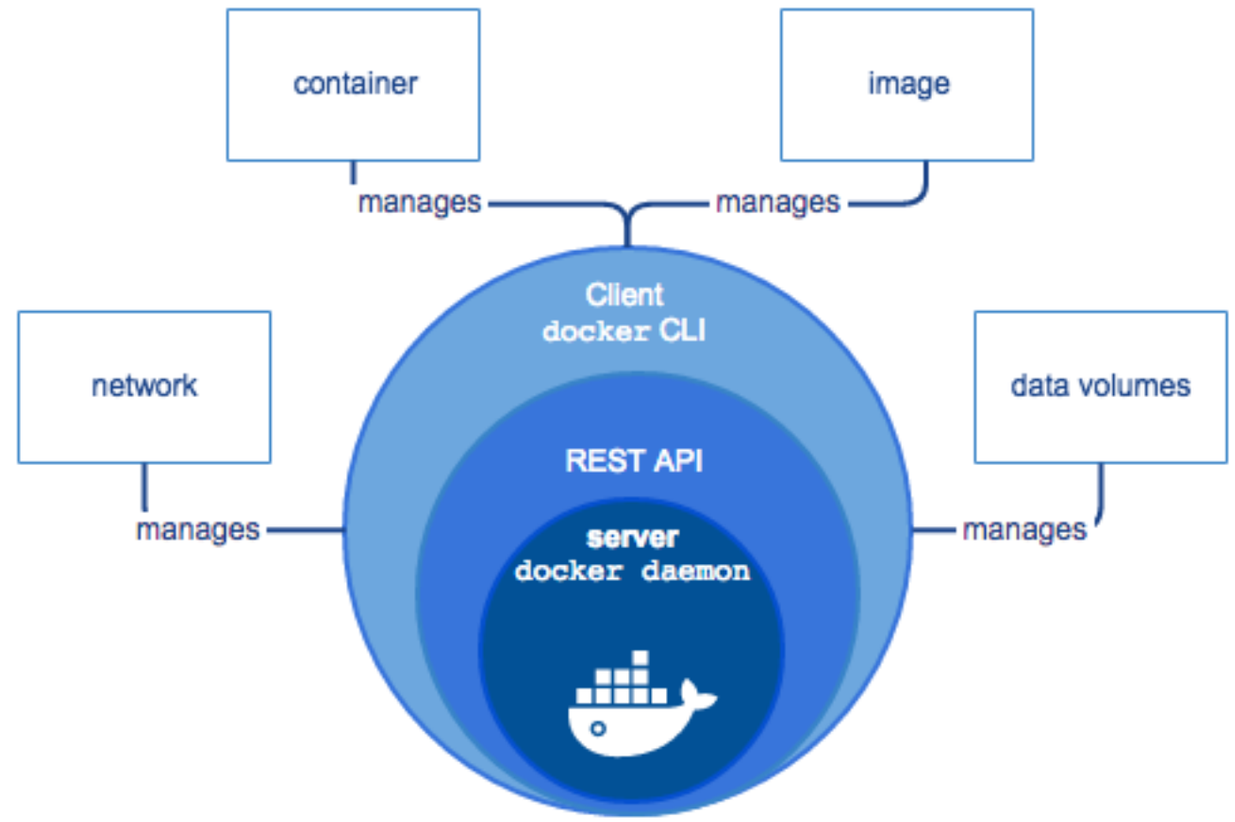
Docker registry

- Хранит образа Docker
- Docker Hub
- Private Registry
- Docker Trusted Registry, Docker Cloud
- Docker store

Docker engine



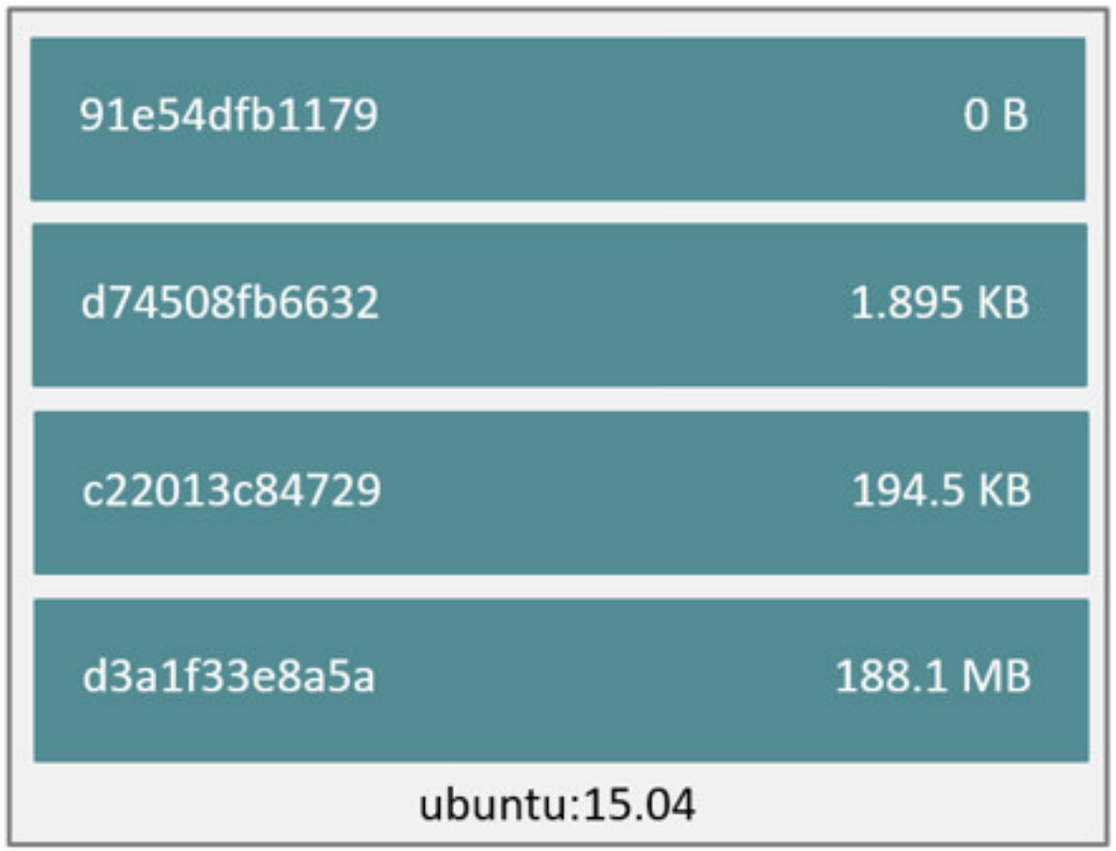
Объекты docker



Docker images

- image – неизменяемая сущность, snapshot контейнера
- image состоит из слоев(layers)
- layers – read-only diff изменений файловой системы

Docker images



Image

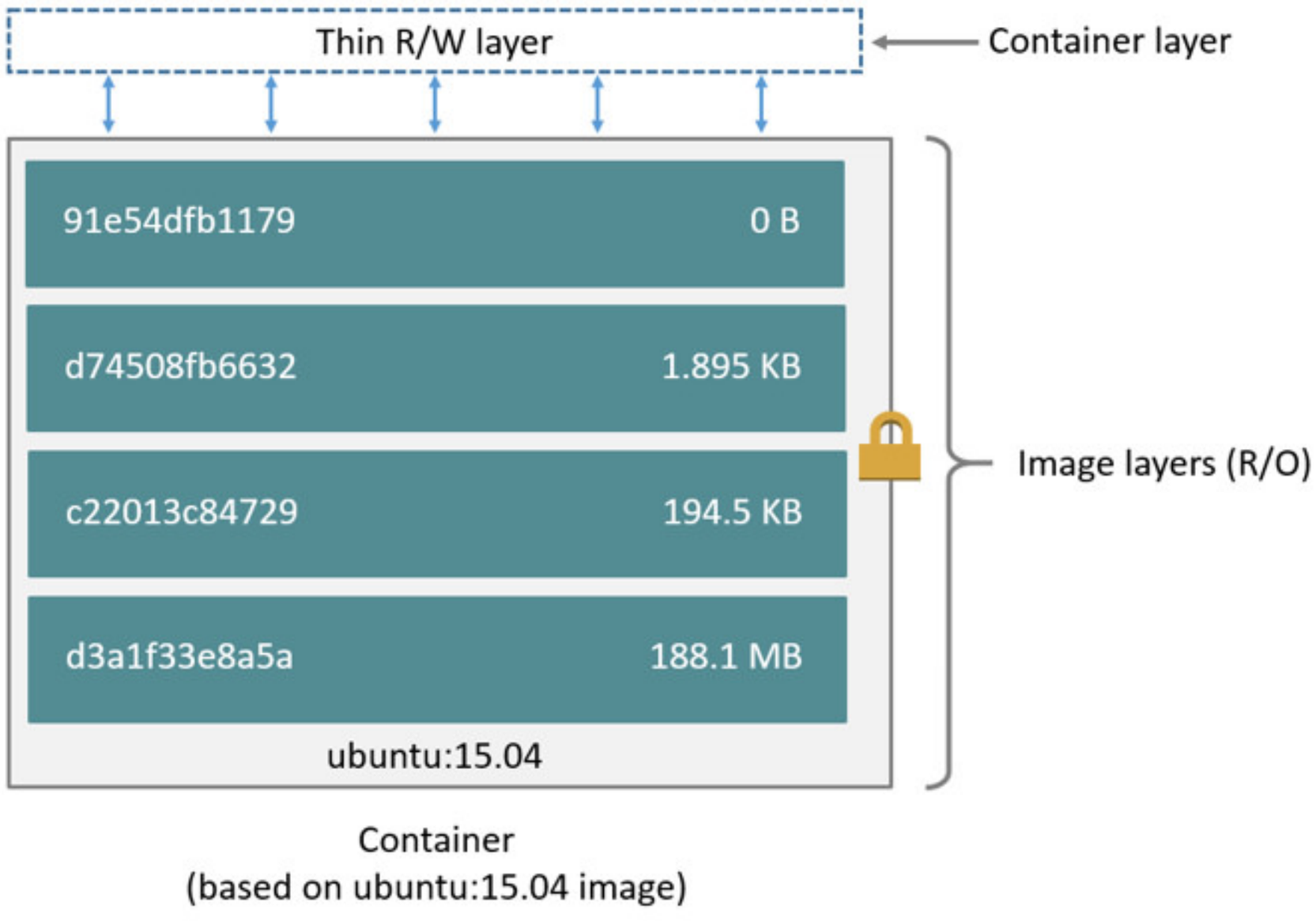
CMD

RUN …

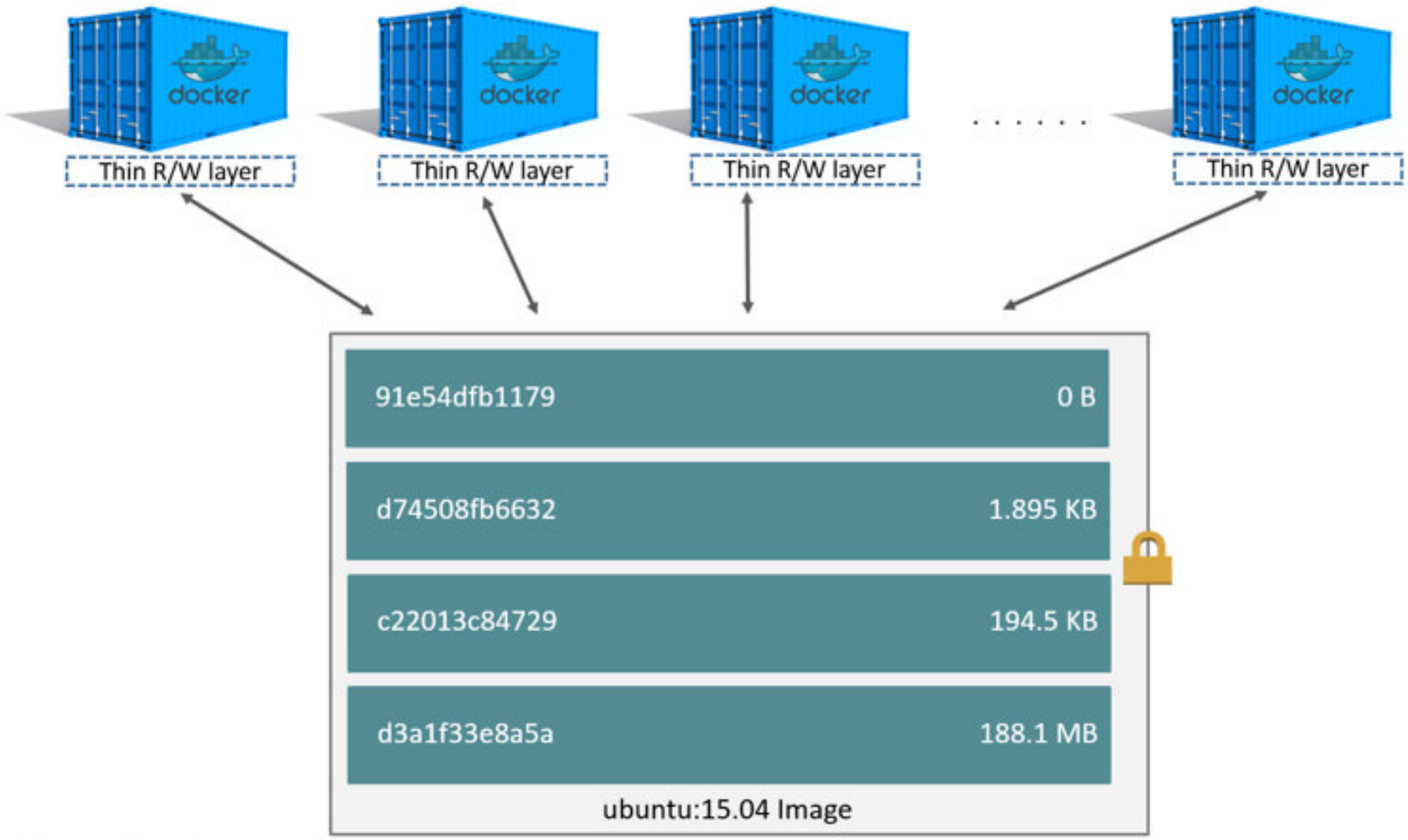
RUN …

ADD/COPY

Docker images



Docker containers



Docker под капотом

- Namespaces
- Cgroups
- UnionFS

Namespaces

- Изолирование окружения

- Каждый контейнер работает со своими namespace’ами

- Pid, net, mnt …

Control groups

- Позволяет контейнерам использовать общие ресурсы
- Ограничивает набор доступных ресурсов
- ЦПУ, память, IO …

Union File Systems

- Разделение по слоям
- Переиспользование слоев