Технология контейнеризации. Введение в Docker

Не забудь включить запись!

План

- Виртуализация
- Как работает контейнеризация
- Из чего состоит Docker
- Немного о безопасности

Виртуализация

- Виртуализация - программная имитация аппаратного обеспечения

Оптимизация

- Аппаратных ресурсов(больше на одном сервере)
- Стоимости(сокращение кол-ва серверов)

Изоляция

- От чужих зависимостей

  > Поддержка окружений с множеством приложений и сервисов со временем становится головной болью администраторов

- От других приложений

  > Тонкая конфигурация и оптимизация операционной системы требует человеческих ресурсов

- От сторонних пользователей

Эмуляция

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

Эмуляция - попытка зеркально сымитировать внутреннее устройство эмулируемой системы таким образом, чтобы программа, отвечающая за эмуляцию какой либо из систем, в точности повторяла все ее процессы и работу компонентов эмулируемой системы

Типы виртуализации

- Программная виртуализация:
  - Динамическая трансляция(VirtualBox)
  - Паравиртуализация(Xen)
- Аппаратная виртуализация (KVM, Xen, VMware, Hyper-V)

Контейнеризация

- Контейнеризация (LXC, OpenVZ, Jail, Zones) — виртуализация на уровне операционной системы

Как работает Docker

- Namespaces
- Cgroups
- UnionFS
- RunC

Namespaces

- Изолирование окружения
- Каждый контейнер работает со своими namespace’ами
  - `pid` : Изоляция процессов (PID: Process ID)
  - `net` : Изоляция сетей (NET: Networking)
  - `ipc` : Изоляция IPC (IPC: InterProcess Communication)
  - `mnt` : Изоляция файловой системы (MNT: Mount)
  - `uts` : Изоляция UTS (UTS: Unix Timesharing System)
  - `user`: Изоляция пользователей
- **Namespace закрывается, если PID 1 умер**

Control groups

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

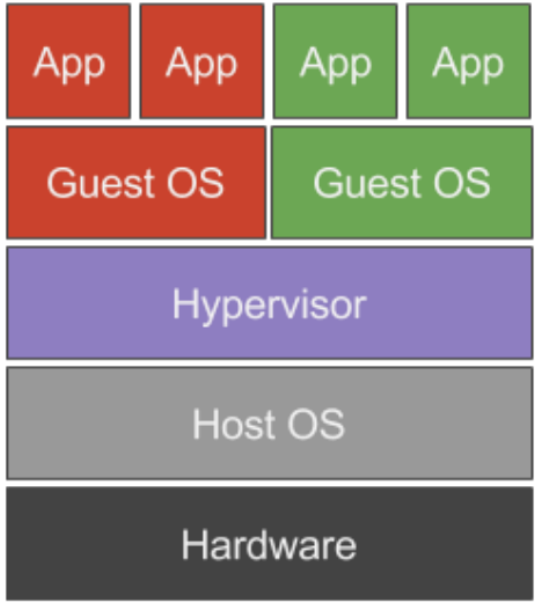
Union File Systems

- Разделение по слоям
- Переиспользование слоев

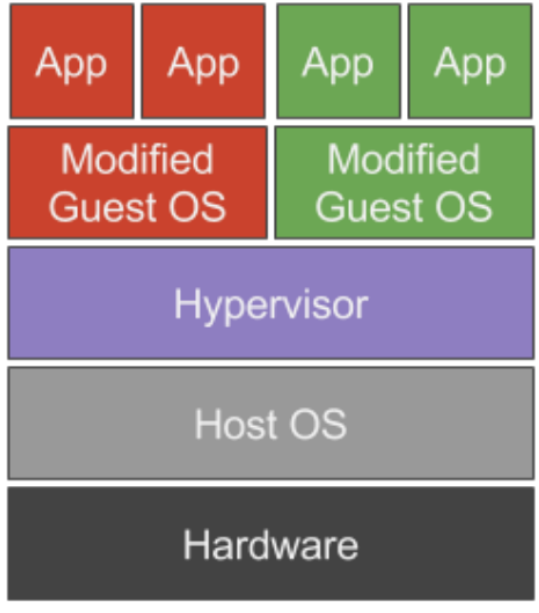
RunC

- Библиотека-обертка над Namespaces, cgroups, UnionFS

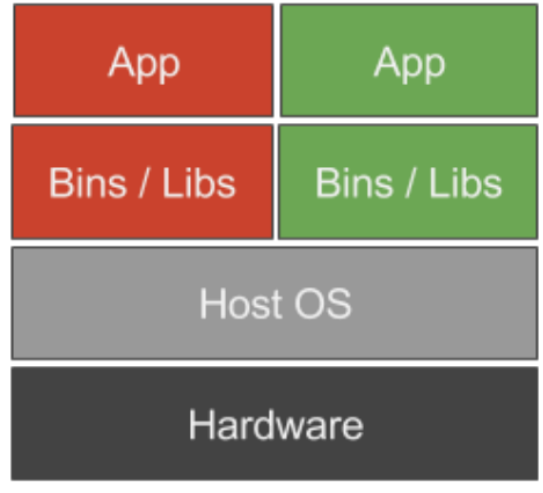
Типы виртуализации



Full Virtualization



Paravirtualization



OS Level virtualization

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**

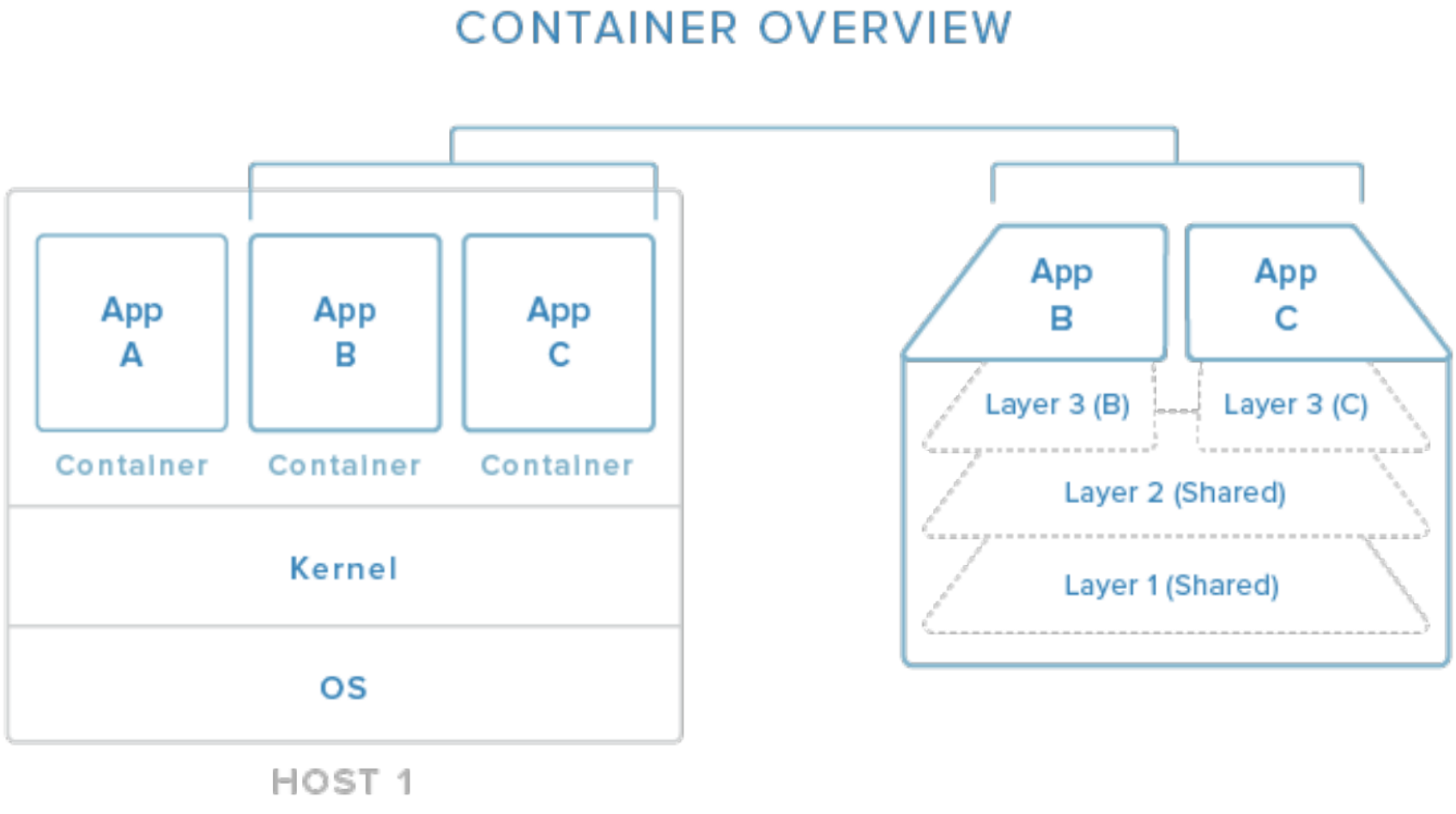
Docker это про стандарты

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

Docker

- Контейнер — это **НЕ** виртуальная машина, а приложение и его зависимости упакованные в стандартизированное, изолированное, легковесное окружение.

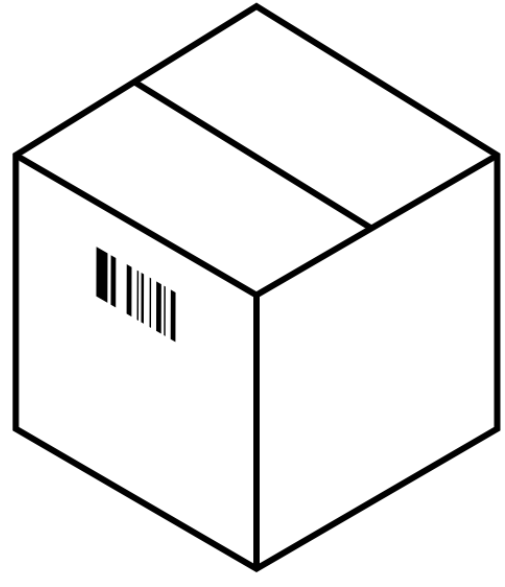
Docker


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)

Что внутри?

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, ...)



Created by Grant Fisher
from Noun Project

Из чего состоит Docker

- Daemon
- Client
- Registry

Docker daemon

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

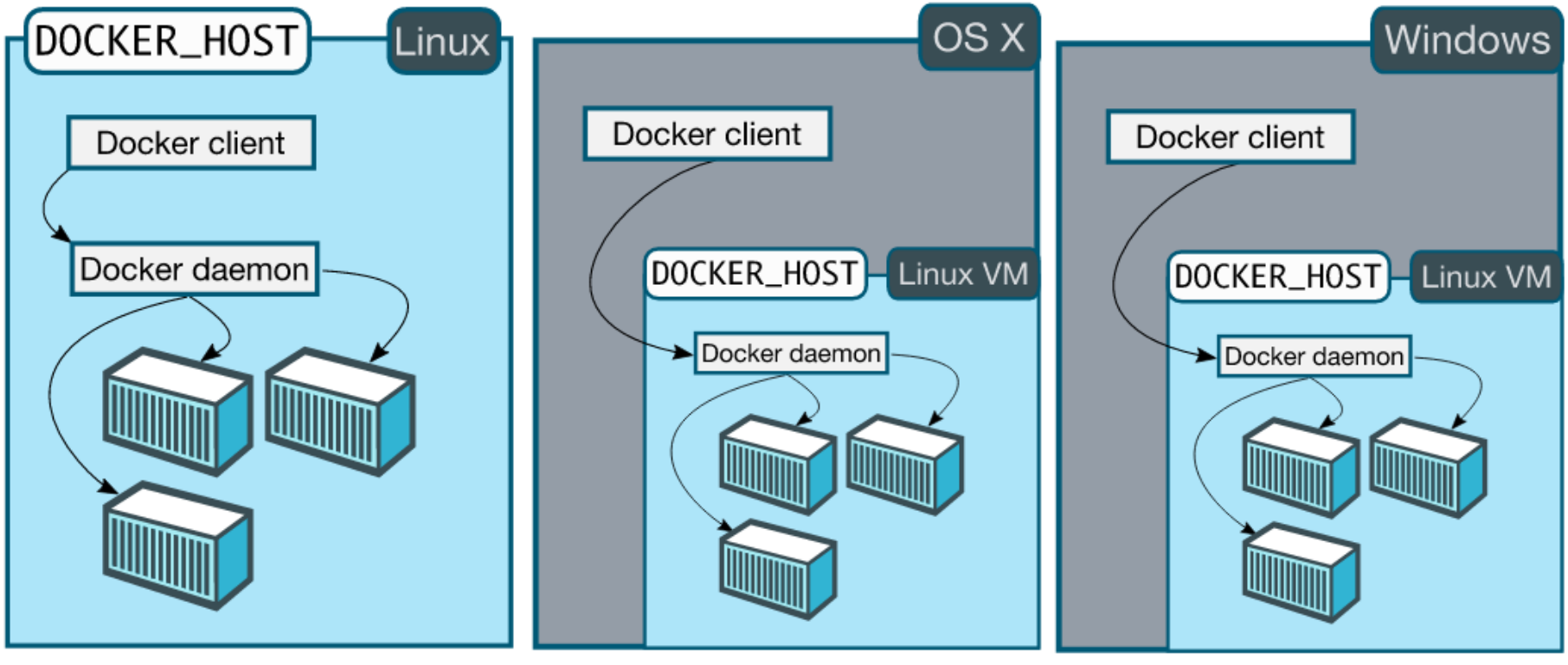
Docker daemon

- 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

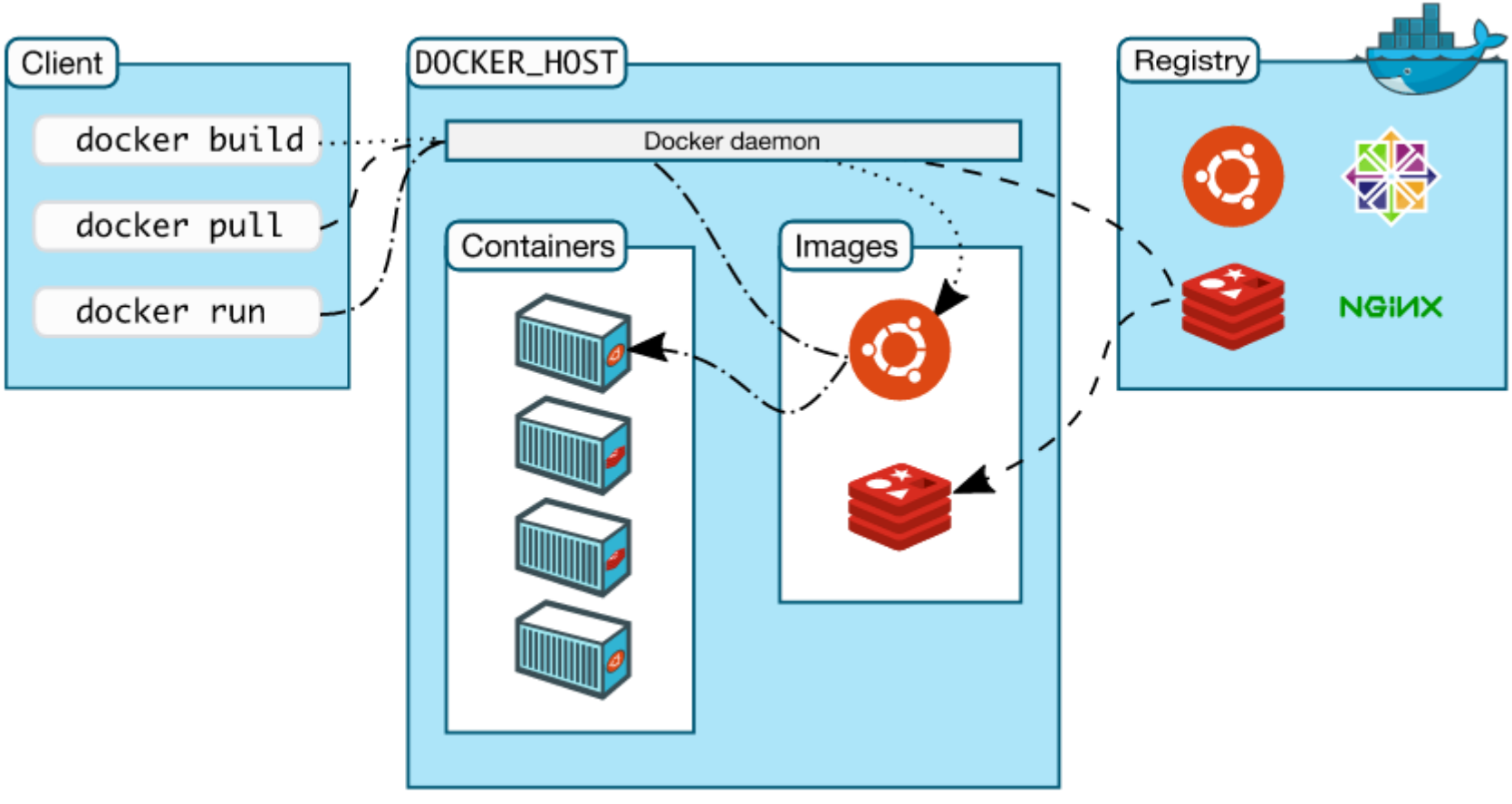
Docker engine



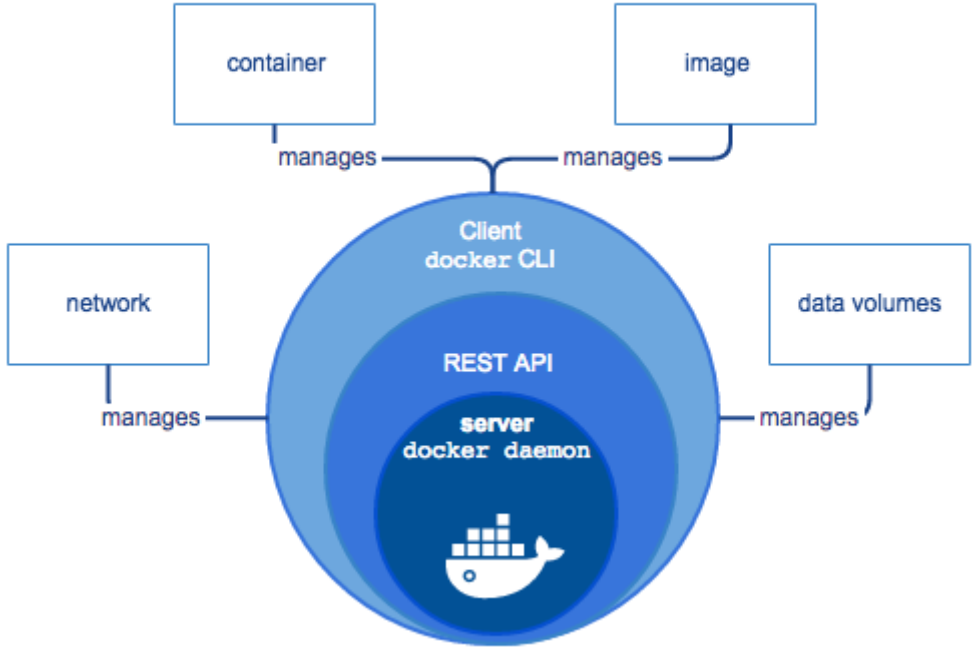
Docker registry

- Docker Hub
- Private Registry
- Docker Trusted Registry
- Docker store

Docker engine



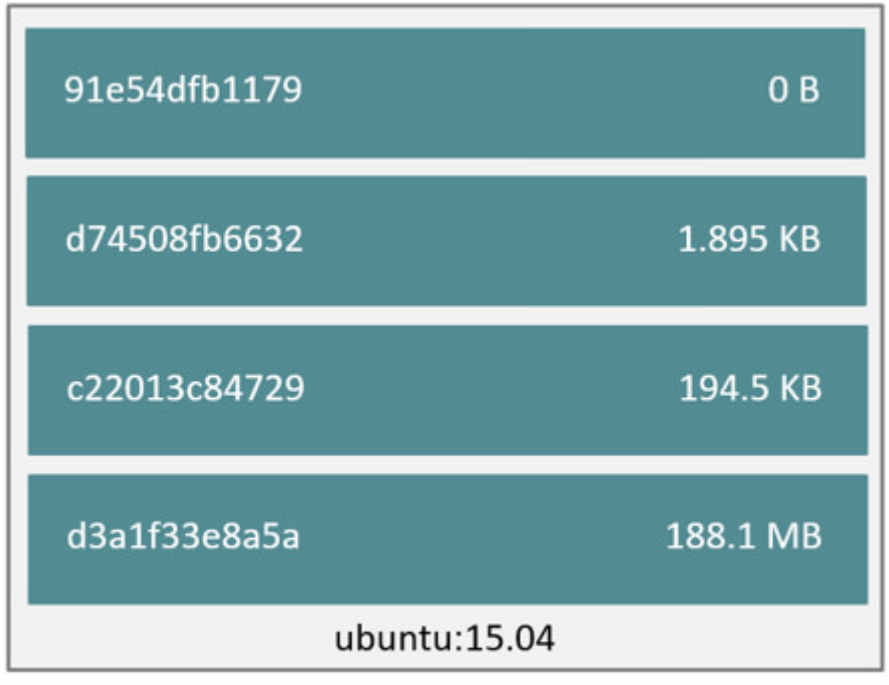
Объекты docker



Docker images

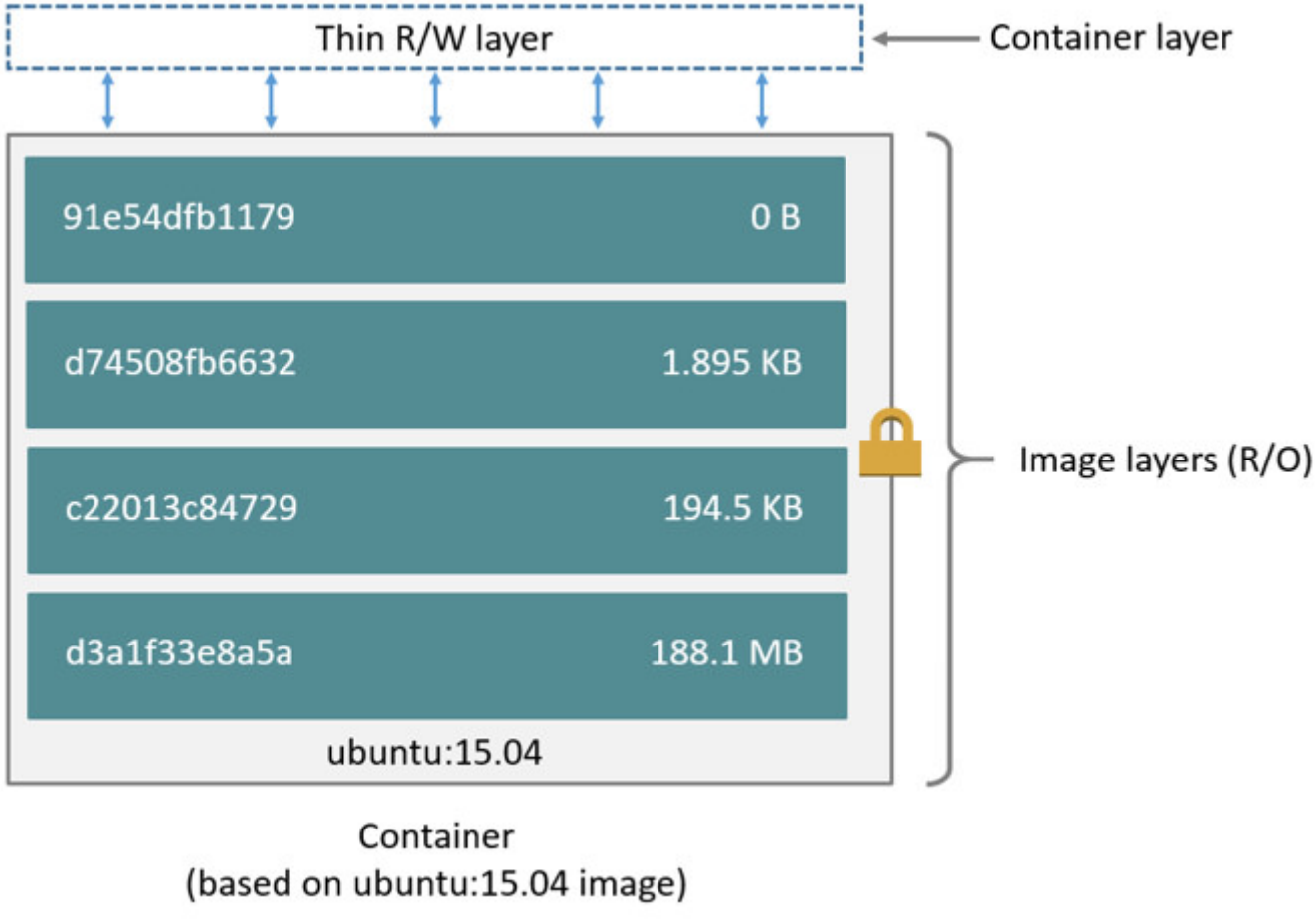
- `image` – неизменяемая сущность, snapshot контейнера
- `image` состоит из слоев(`layers`)
- `layers` – read-only diff изменений файловой системы

Docker images

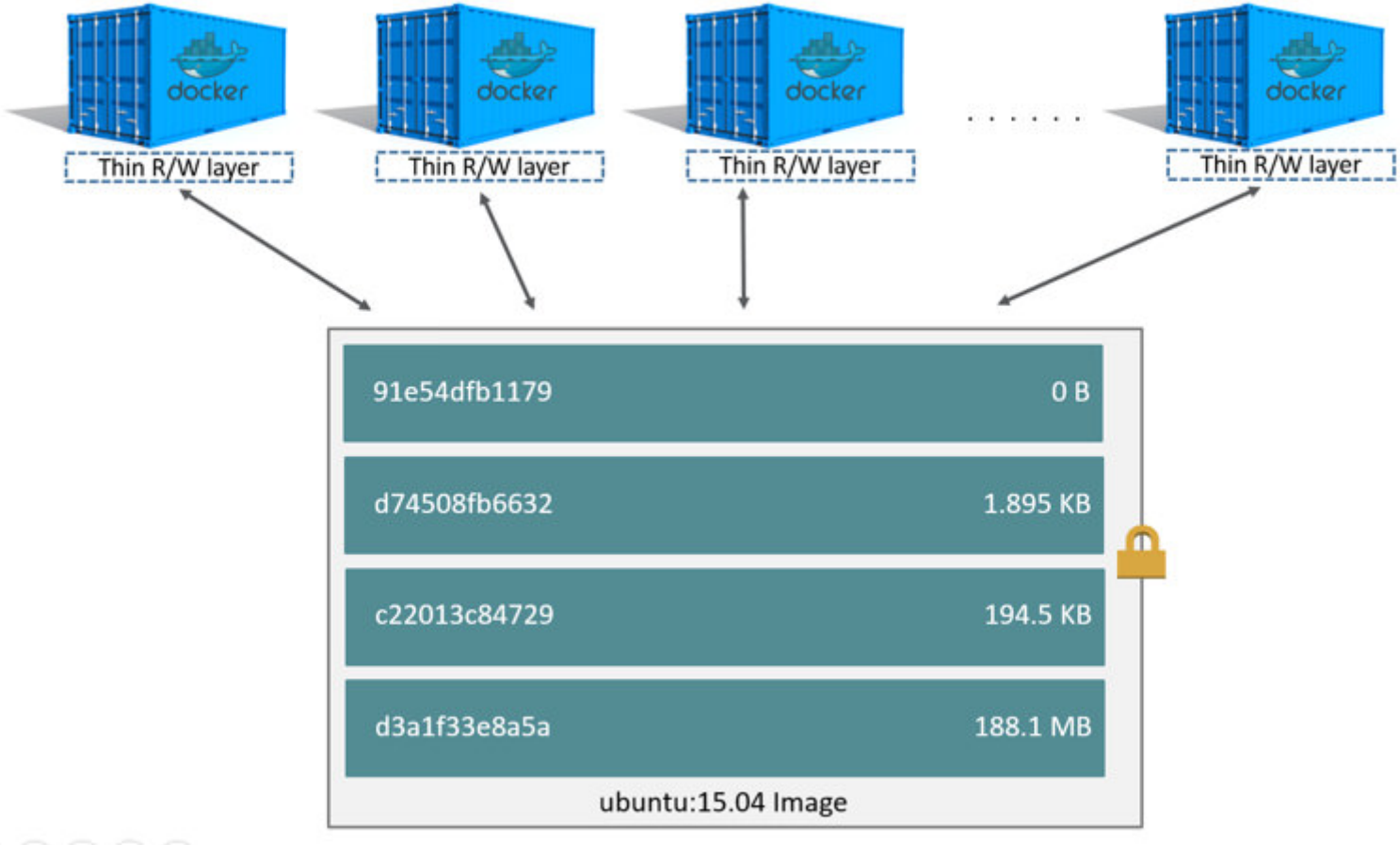


Image

Docker images



Docker containers



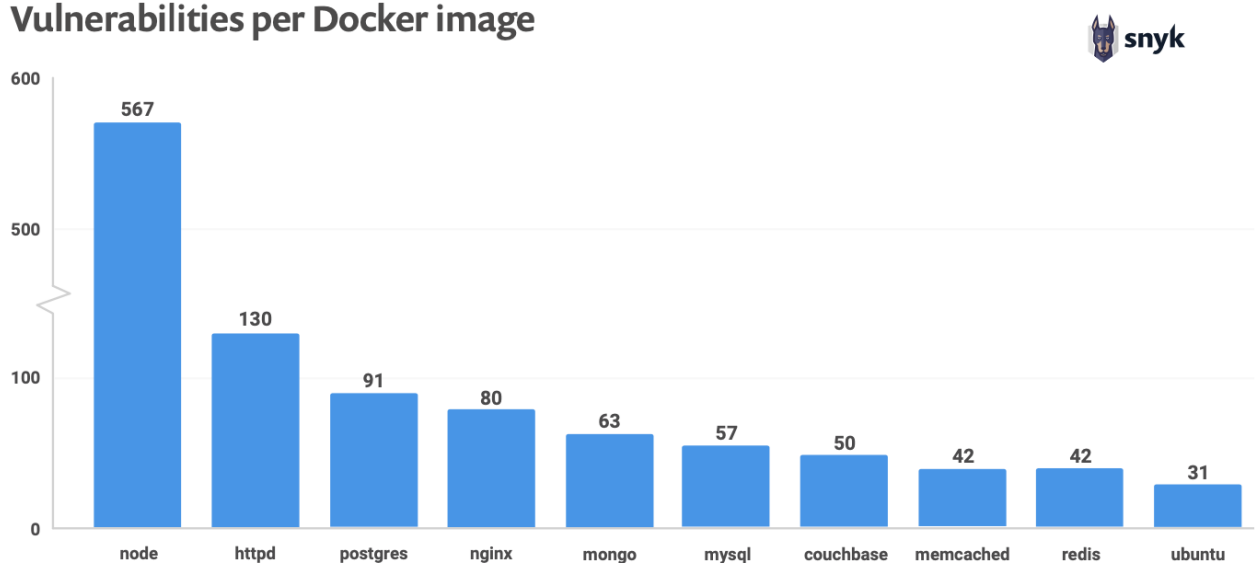
Безопасность

- `Docker` — это всего лишь тонкая прослойка
- Привилегии пользователей ограничены Whitelistом
- Можно включить user-namespace
- Не запускайте приложения от `root`
- Надо заниматься патч-менеджментом
- Не использовать передачу паролей и прочих секретов через `ENV`-переменные

State of Docker security (2019)

В настоящее время Docker Hub содержит:

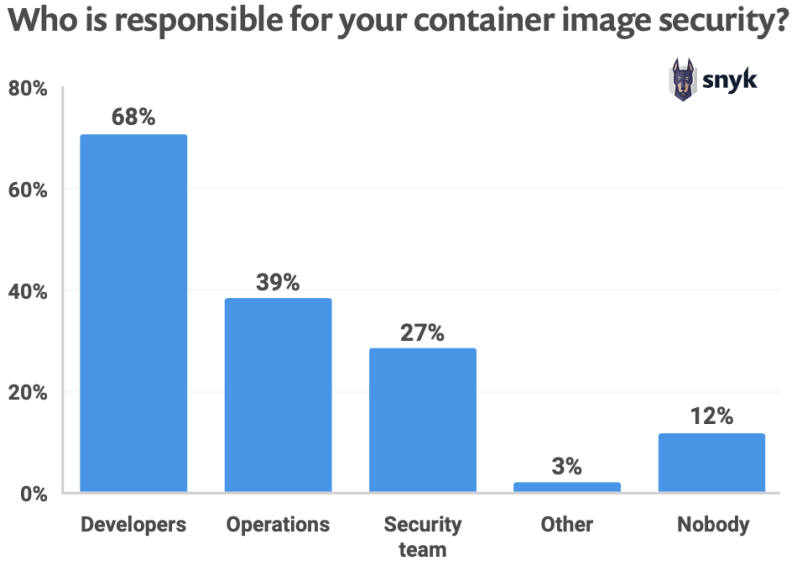
- 223 images от проверенных издателей
- 151 официальный image
- 40 сертифицированных docker images

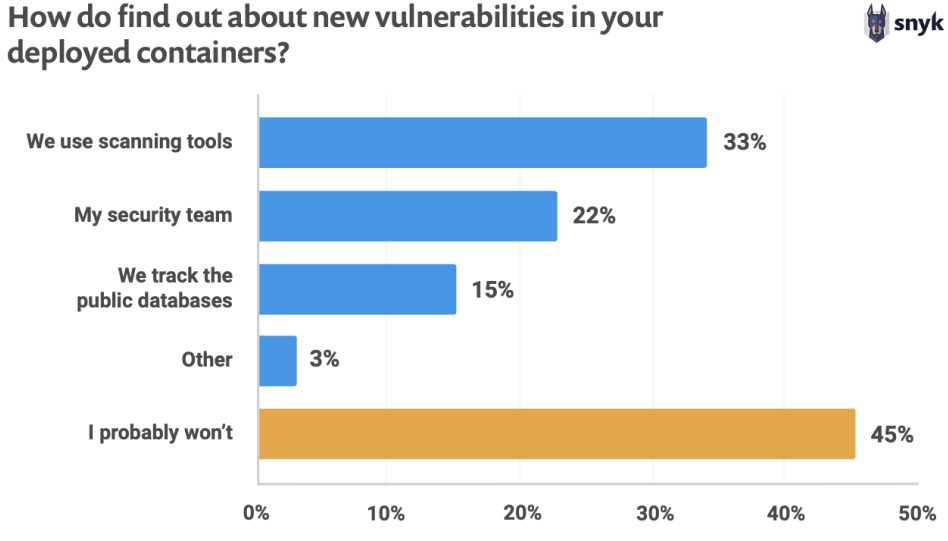


Отчет Snyk

Docker security

- **68%** разработчиков сами отвечают за безопасность контейнеров
- **50%** не сканируют *operating system (OS) layer* в docker image
- **45%** никогда не находят новые уязвимости в их контейнерах
- **80%** не тестируют свои Docker images во время разработки





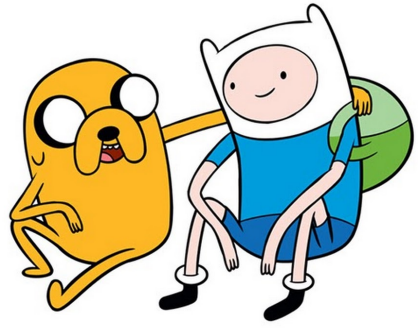
Как улучшить Docker security

- Правильный выбор base image
  - используйте самое минимальный image: *не упаковывайте то, что вам не нужно!*
- Использование multi-stage сборок
- Пересборка images
  - при пересборке образов используйте ––no–cache
- Сканирование images на стадии development
- Сканирование контейнеров в production

10 Docker Image Security Best Practices

Ссылки

- IBM Research Report
  - “Проникновение в Docker”
- "Контейнеры в Linux"



Спасибо за внимание!

Время для ваших вопросов!