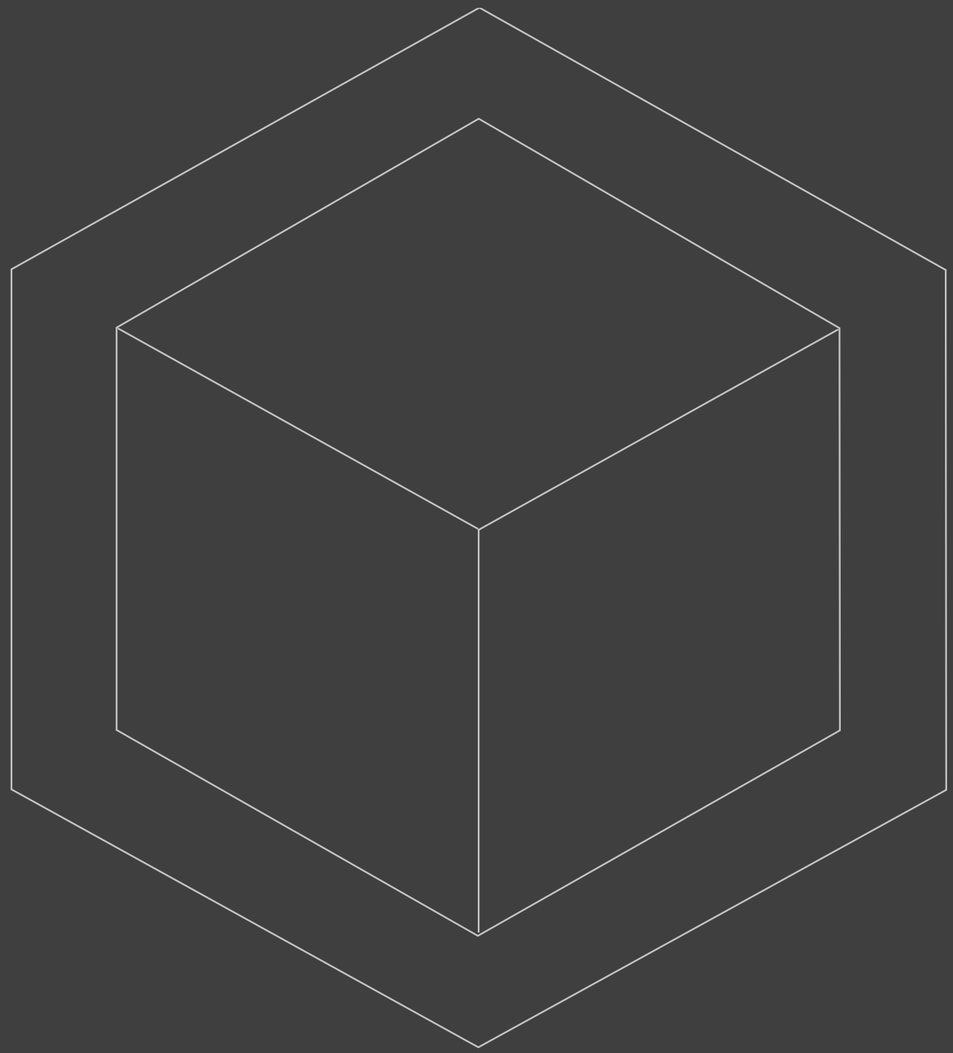
Хранение данных в контейнерах и Kubernetes

Николай Куликов
Консультант
по решениям VMware

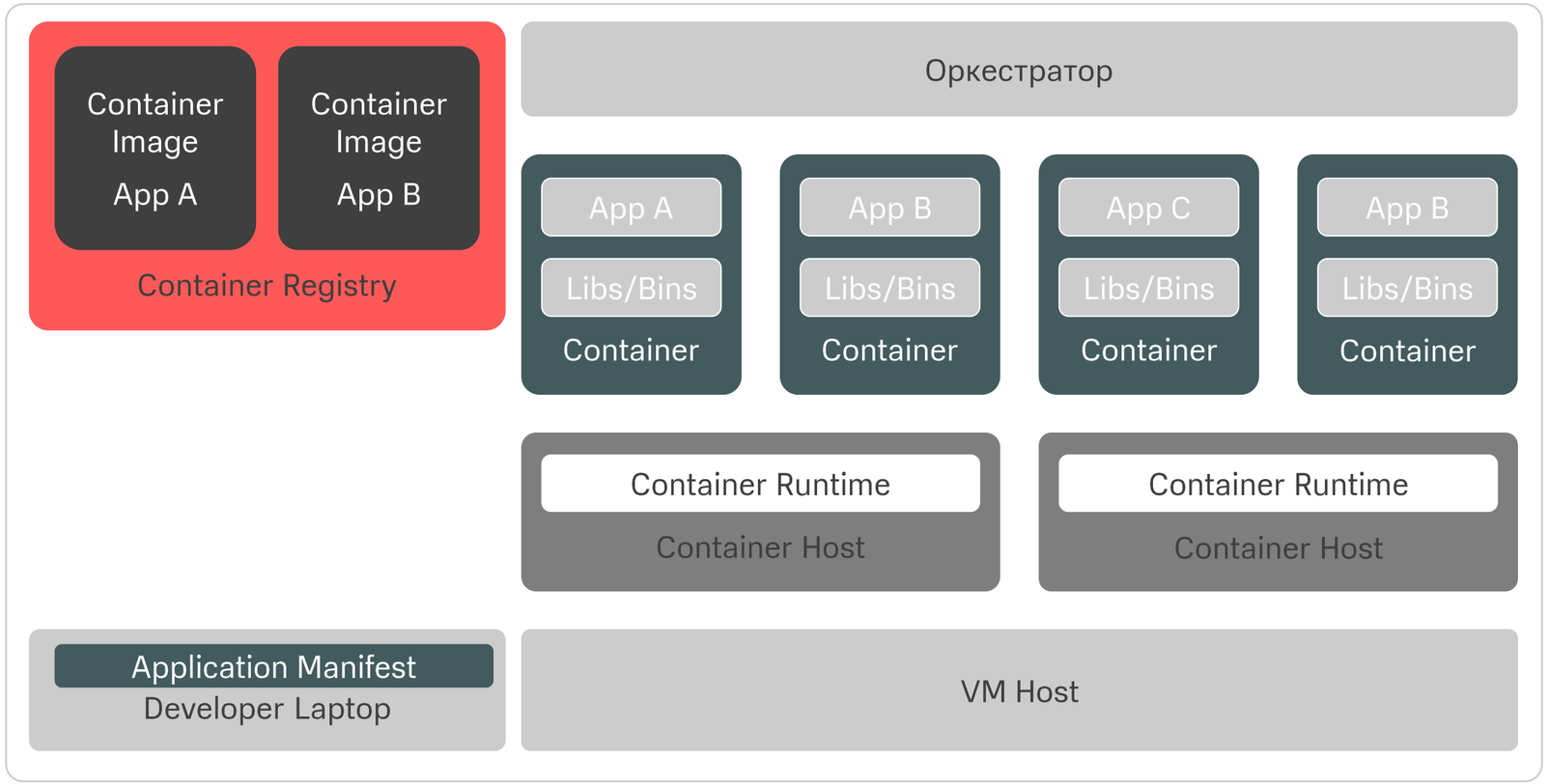
Cloud Thinking

Краткий обзор контейнеров

(с точки зрения хранения данных)

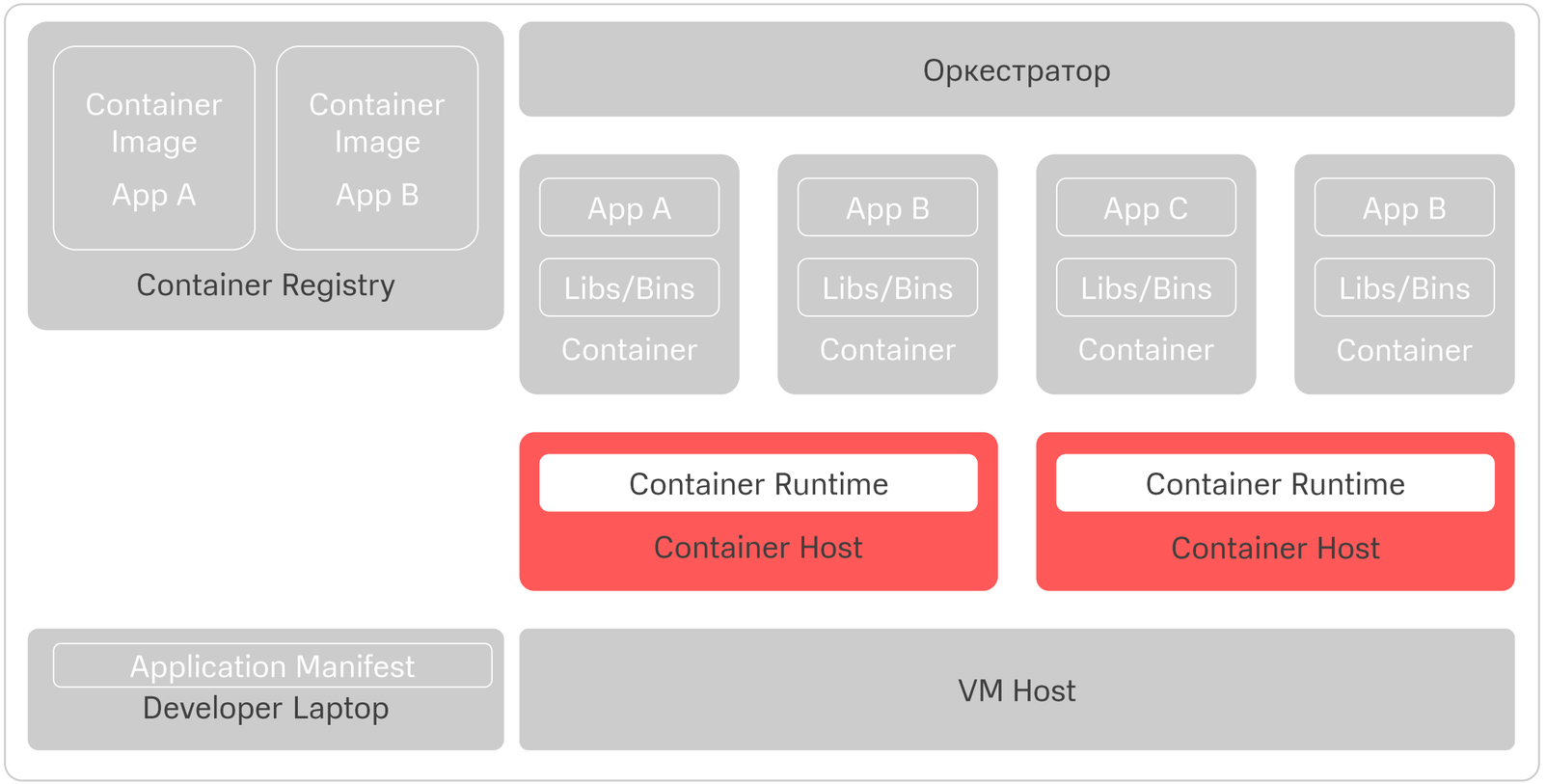


Основные компоненты контейнерной инфраструктуры



Container Runtime
Container Host
Application Manifest
Container Image
Container Registry
Container (Instance)
Container Orchestrator

ОС, на которой запускаются контейнеры



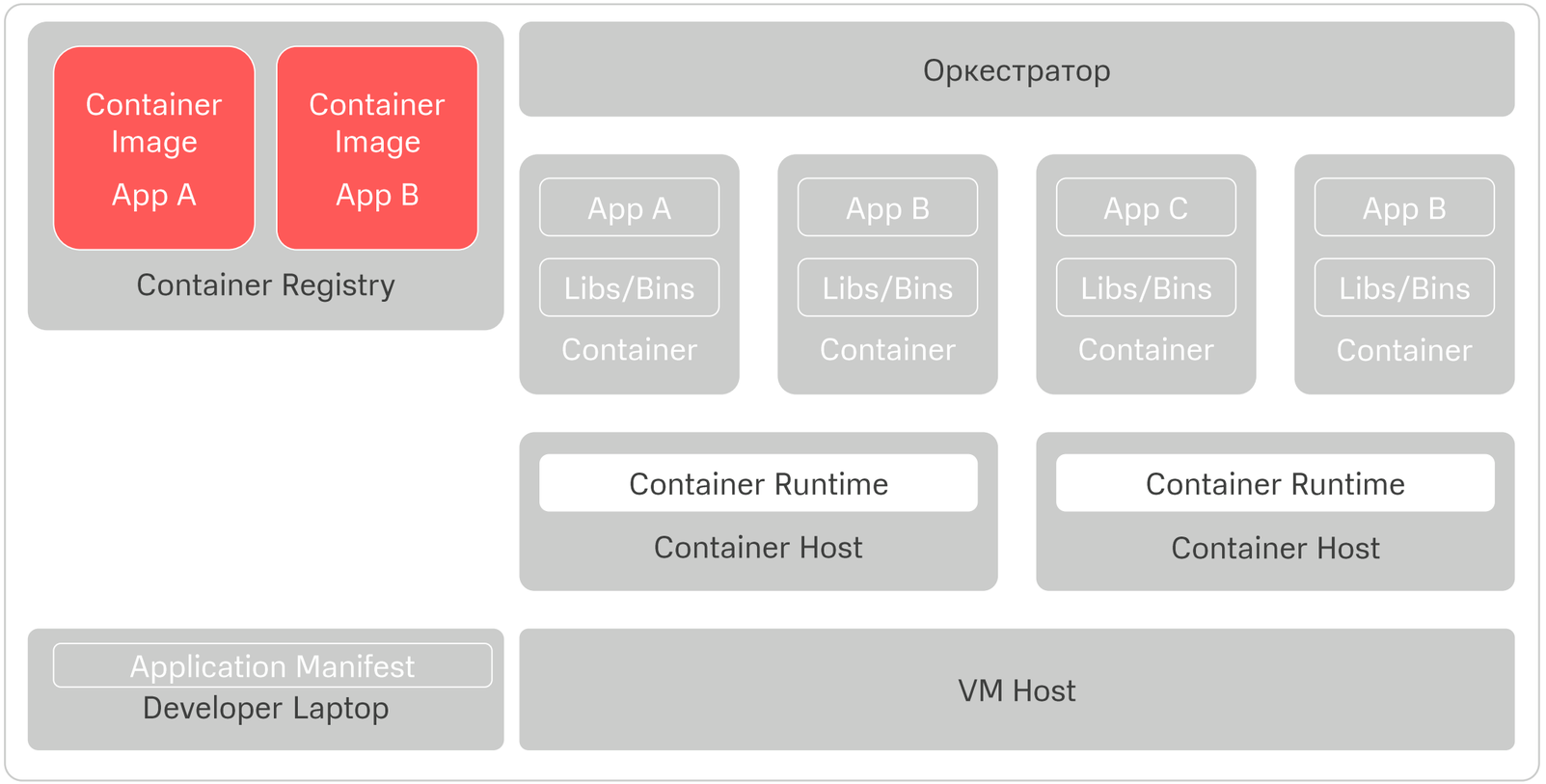
В большинстве случаев работает на ВМ, но может быть и на bare-metal

Обычно это Linux OS. Например, Ubuntu, CentOS, PhotonOS. Windows тоже есть, но это отдельная история

Внутри установлен Container Runtime, который запускает контейнеры в форме процессов

Ядро общее для всех контейнеров на хосте

Базовая единица приложения в контейнерном мире



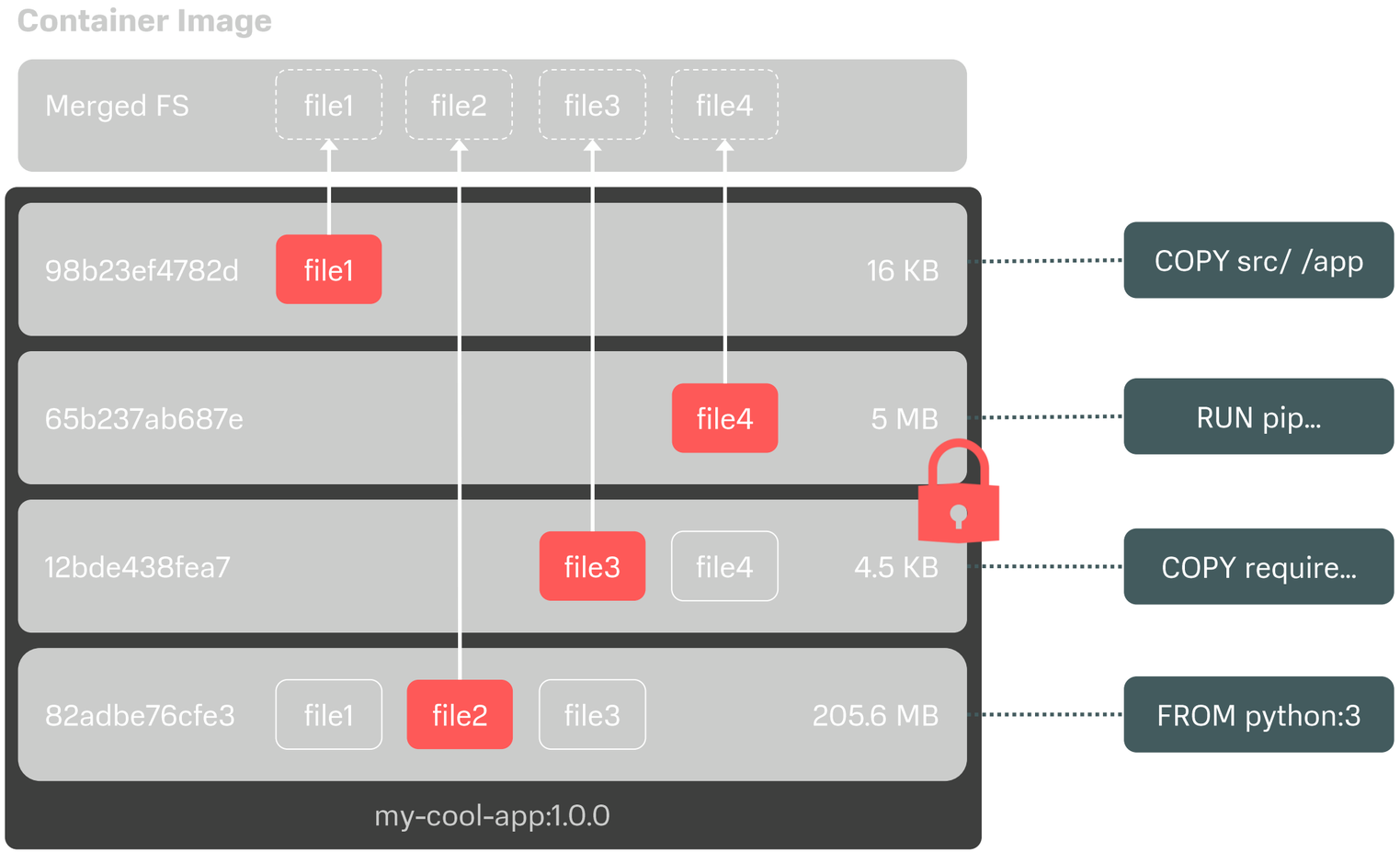
Обычно tarball

Использует Application Manifest для сбора в Container Image

Представляет собой immutable образ приложения

Содержит все, что нужно приложению для запуска

Слои на уровне файловой системы



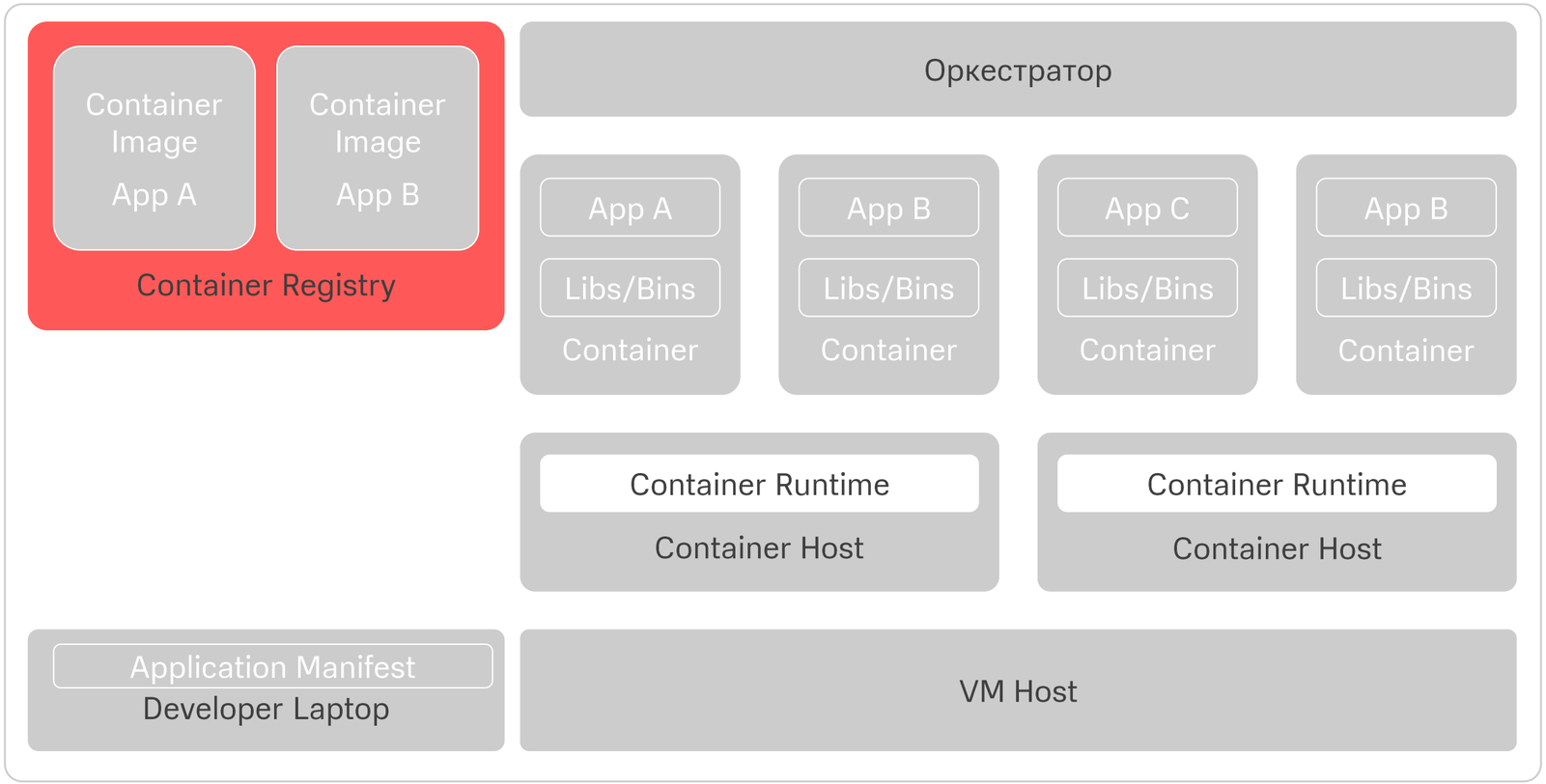
Известна как OverlayFS

Использует Copy-on-Write (CoW)

Слои добавляются к образу при его изменение/обновлении

Каждый слой в образе Read-Only

Место хранения образов



Хранит ваши Container Images

Может быть приватным или публичным
Docker Hub, Harbor

Часто занимается сканированием
на уязвимости в образе

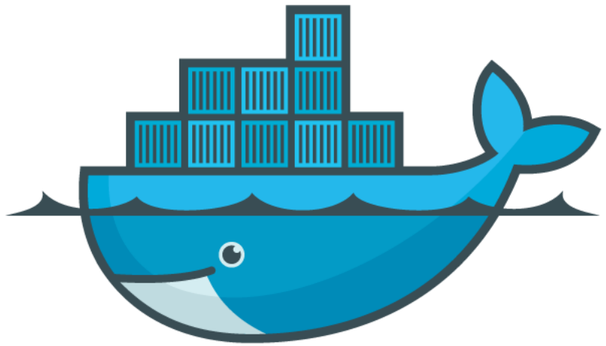
Проверка целостности через подписи

Место хранения образов

Container Registry

Публичный

Docker Hub – hub.docker.com

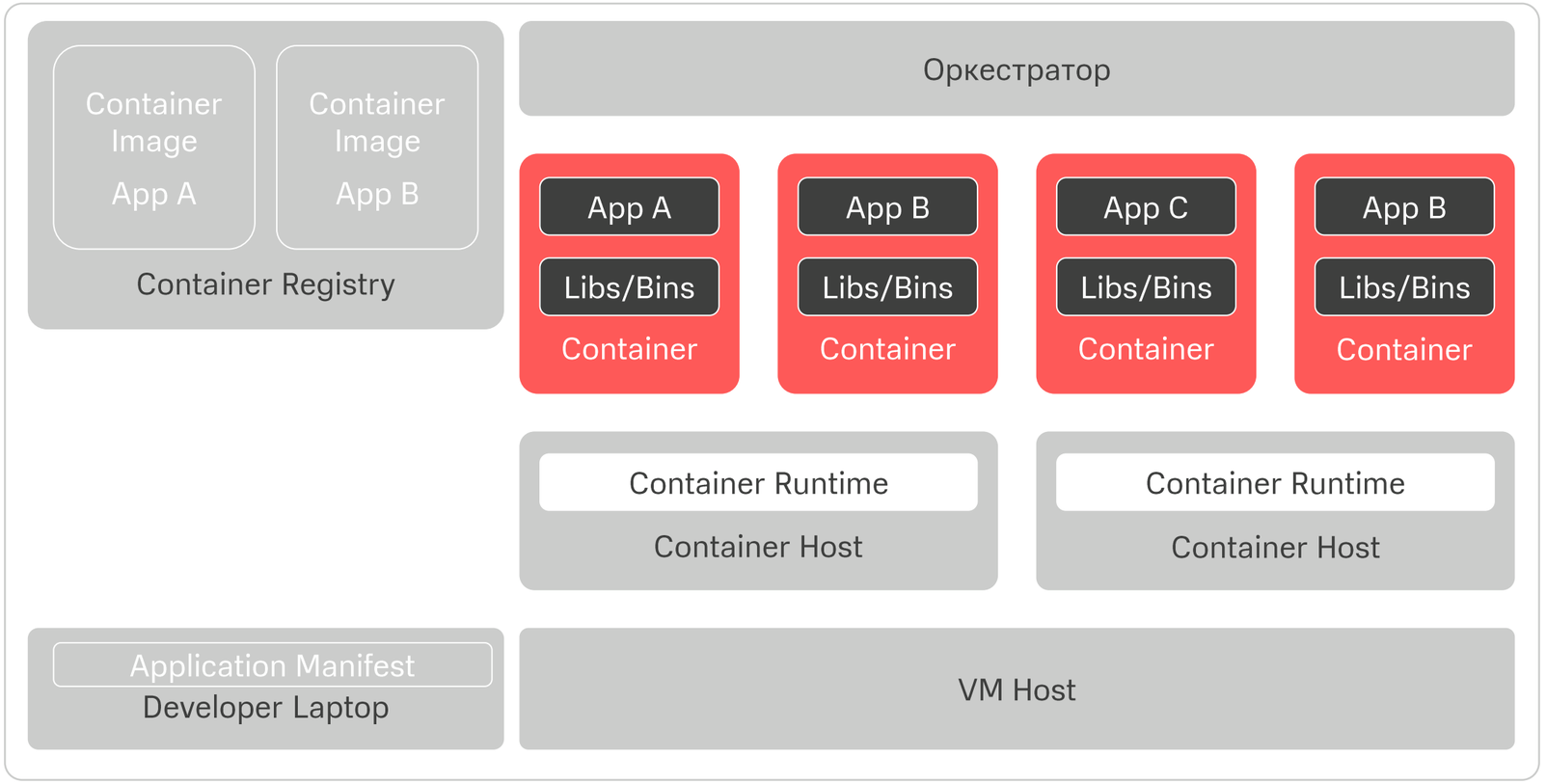


Частный

Harbor – goharbor.io

Запуск контейнера

Container (Instance)



Запущенный экземпляр Container Image

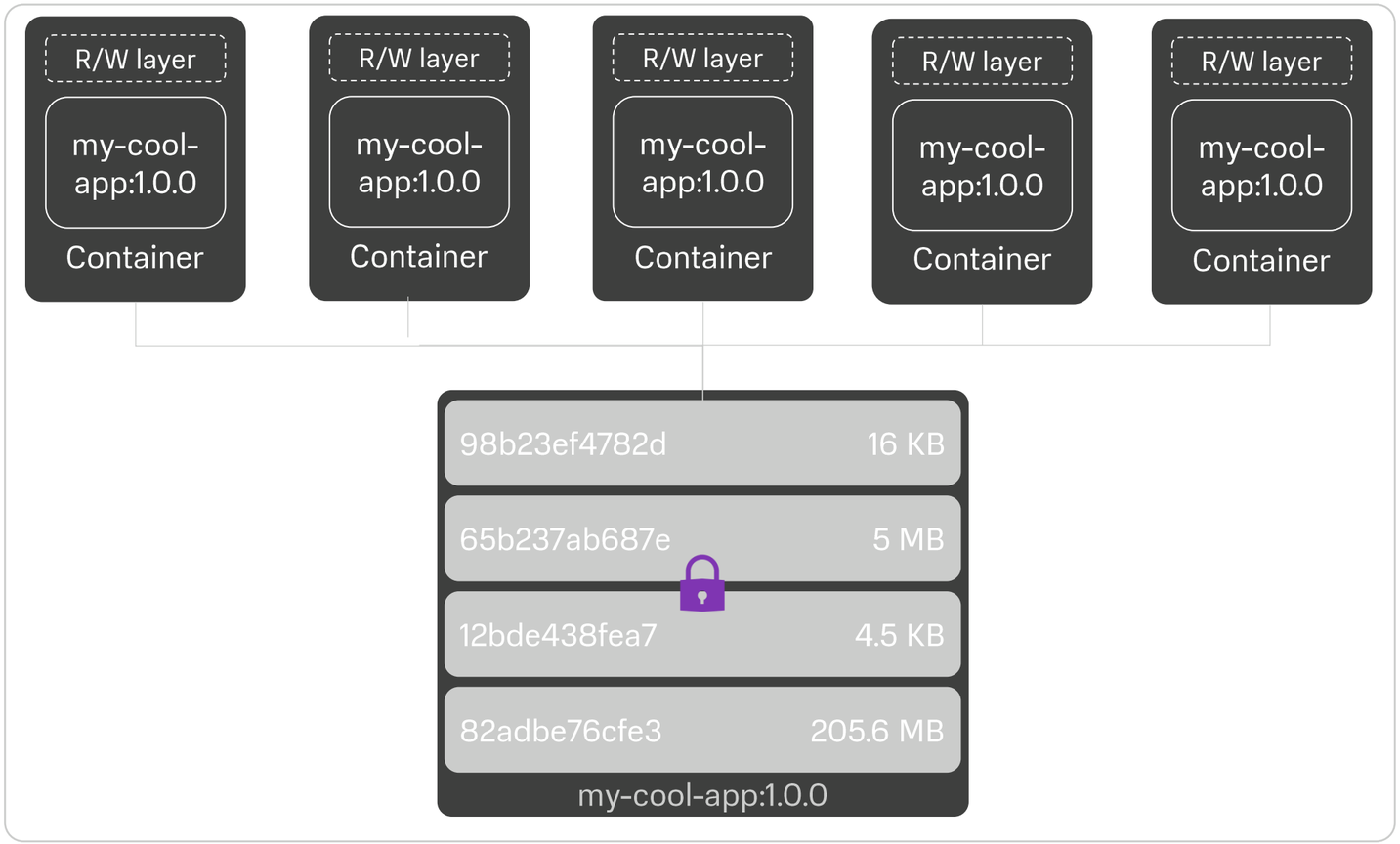
Immutable

Не хранит в себе состояние

Множество контейнеров запускается из одного образа

Изолированы друг от друга

Общий образ для множества контейнеров



То, что образ immutability позволяет ему быть общим

Каждый контейнер содержит собственный тонкий R/W

Свое индивидуальное состояние в слое

Все изменения не сохраняются, когда контейнер удаляется

Изменения должны быть сохранены в формате нового слоя образа, то есть мы меняем сам образ

Смещение парадигмы: Pets vs Cattle

Домашние животные

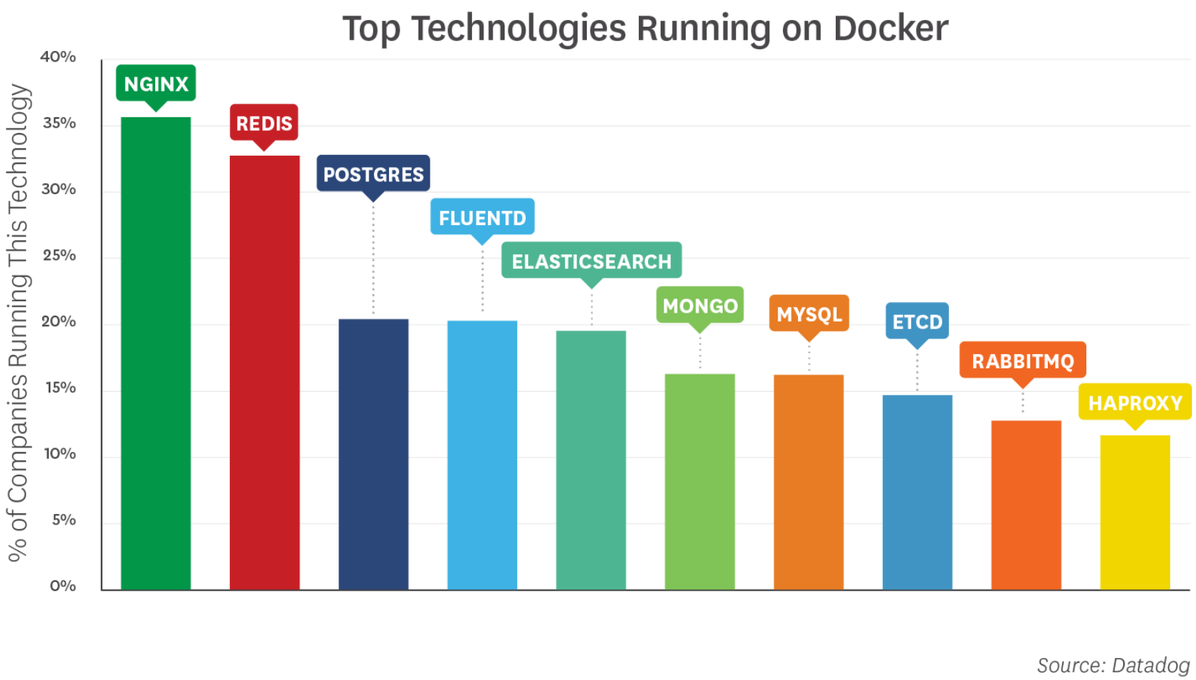
- Каждый из них уникален
- Управляем, заботимся, обслуживаем исходя из индивидуальных потребностей
- Стараемся избежать гибели всеми силами, так как.
  они «незаменимы». Лечим при необходимости

Скот

- Относимся как к «голове в стаде»
- Автоматически создаются – управляем только количеством
- Гибель – нормальный процесс и часто дешевле/быстрее заменить, чем лечить

Реальность: Огромное число реальных контейнеров – stateful

Со stateless все только начиналось



Source: Datadog

7/10 из наиболее популярных контейнерных приложений – stateful

Потому что разработчикам удобно использовать K8s для любых типов приложений

А как же жить с stateful-приложениями?

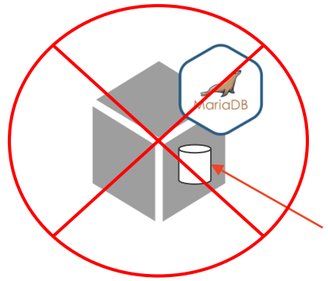
Например, СУБД – stateful-приложения.

Stateful-приложения требуют хранить данные – например, БД. Или пользователь загружает свои документы/картинки в сервис.

Но, как мы помним, при перезапуске контейнера данные в слое не сохраняются.

Есть несколько вариантов решения проблемы.

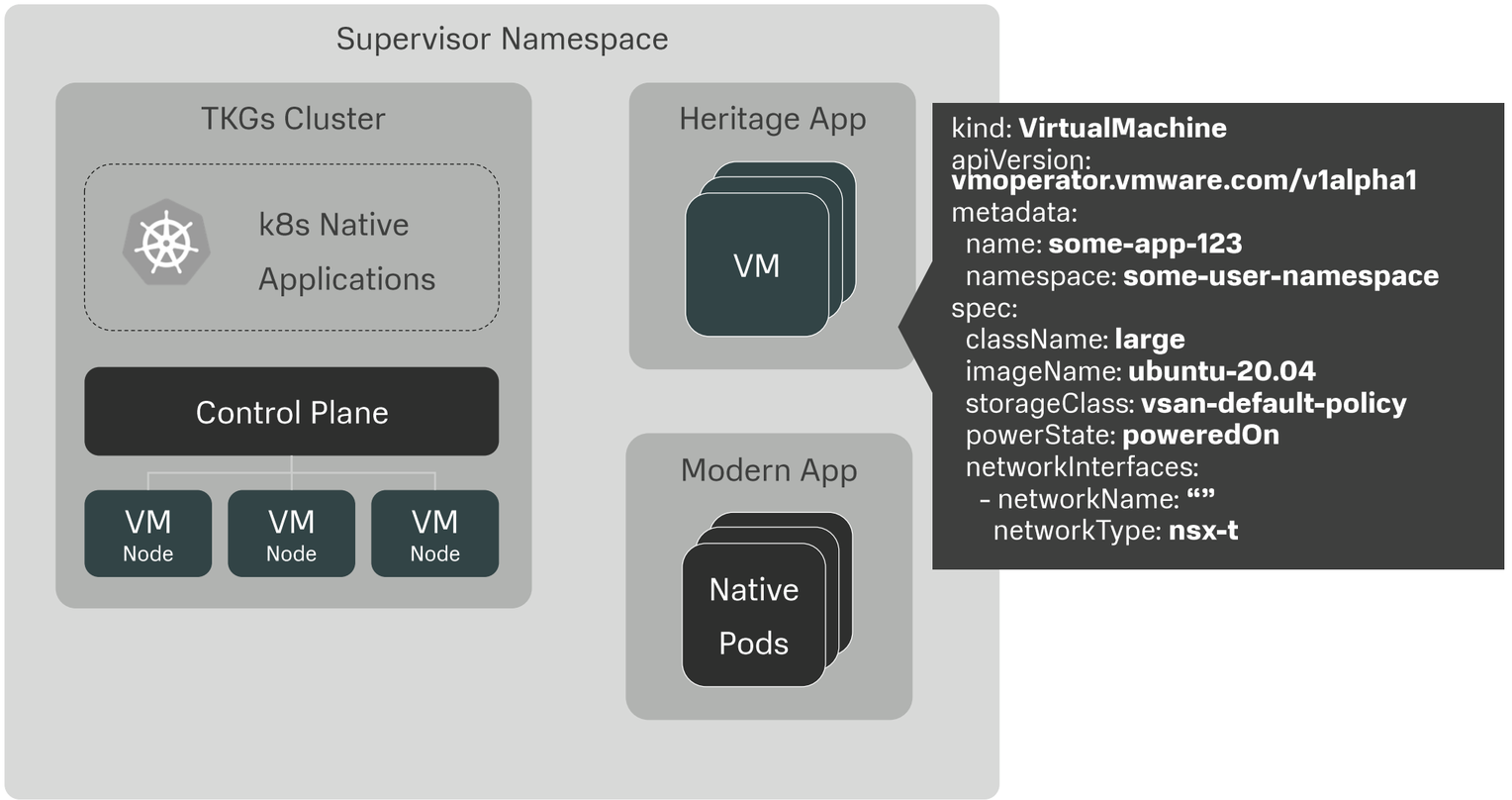
Data lost!



Volume

Вариант 1. Развернуть классические ВМ, но вместе с контейнерами и одними средствами

VM Operator



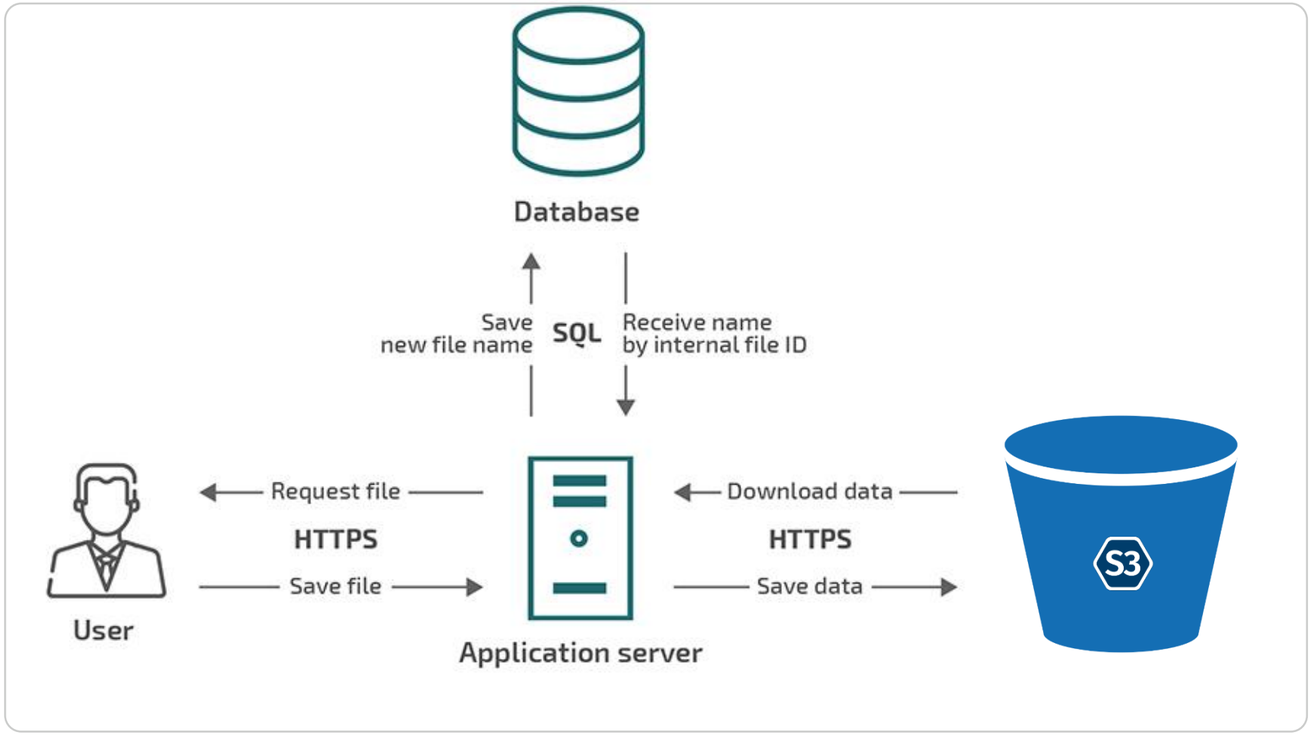
Объединение подходов Pets and Cattles в рамках одной инфраструктуры

Для Dev-команды практически ничего не меняется (только делаем другое описание для ВМ) – они пользуются привычными им инструментами

Для Ops-команды сохраняются существующие средства обеспечения защиты данных, резервного копирования, управления и т. д.

Относительно новый подход

Вариант 2. Внешнее хранилище (обычно S3-совместимое)



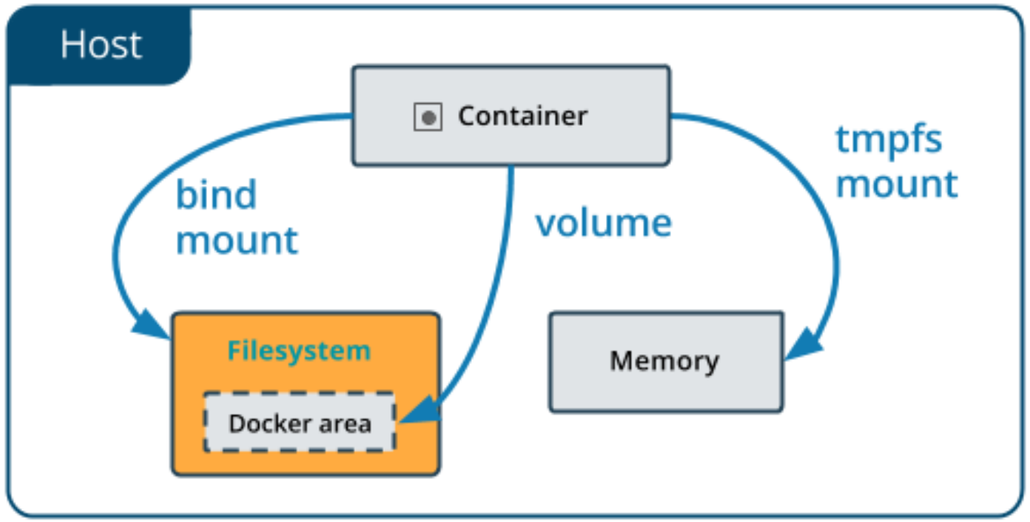
Храним данные в виде объектов во внешнем постоянном хранилище

Обращаемся к данным через HTTPS-вызовы

Приложение должно «уметь» работать с S3

Относительно «медленное» хранилище с фокусом на последовательный доступ и долгое хранение

Вариант 3. Docker Volumes



Монтируем внутрь контейнера том/папку по ссылке/пути на «внешнее» (относительно)контейнера) хранилище

Высокопроизводительное хранение

Может быть общим для нескольких контейнеров

Работа с Volumes

У Docker есть компонент, управляющий хранилищем

```
$ docker volume create myvol
```

Посмотрим описание

```
$ docker volume inspect myvol
...
        "CreatedAt": "2021-09-10T09:25:30Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/myvol/_data",
        "Name": "myvol",
        "Options": {},
        "Scope": "local"
...
```

Существуют различные драйверы для работы с хранилищами

Тестируем

Можем ли мы примонтировать том к запущенному контейнеру?

Нет

```
$ docker run --name cont1 -v myvol:/mnt/volume -ti bitnami/minideb
```

Volume Name : Mount Point

Тогда запускаем новый контейнер с volume

Может ли один volume быть подключен к нескольким контейнерам?

Да

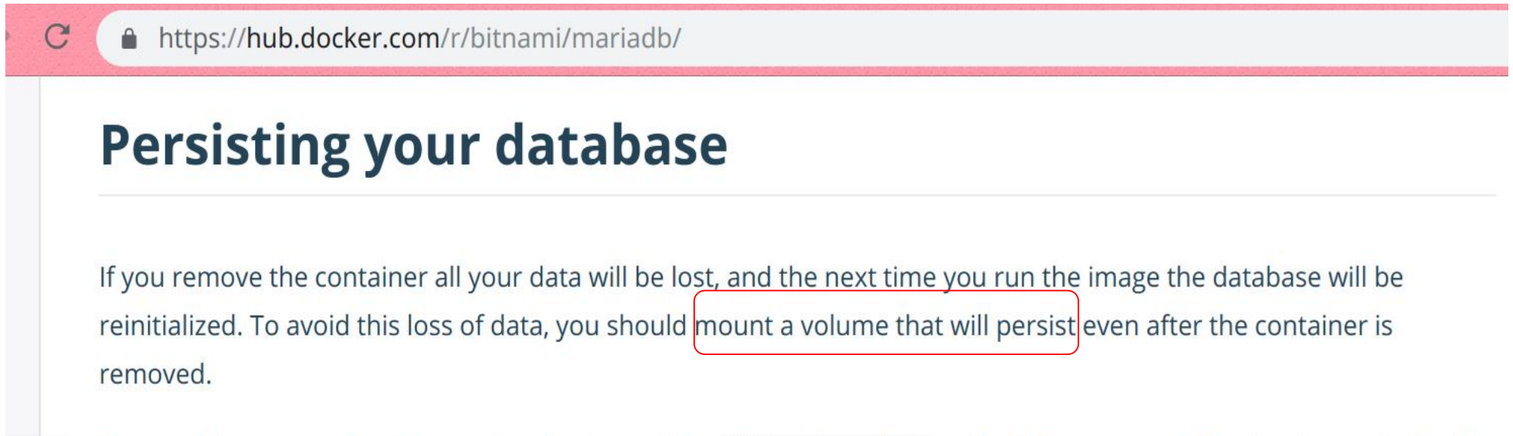
```
$ docker run --name cont2 -v myvol:/mnt/volume -ti bitnami/minideb
```

Теперь останавливаем и удаляем контейнеры, а затем создаем новый

```
$ docker stop cont1 cont2 && docker rm cont1 cont2
$ docker run --name cont4 -v myvol:/mnt/volume -ti bitnami/minideb
```

Приложение должно уметь работать с Volume

Смотрите документацию к приложению




```
$ docker run -d -p 3306:33306 --name maria2 \
  --hostname mymaria \
  --network testnetwork \
  -e MARIADB_ROOT_PASSWORD="rootroot" \
  -e MARIADB_USER="test" \
  -e MARIADB_PASSWORD="admin123" \
  -e MARIADB_DATABASE="bootcamp" \
  -v mariadb_data:/bitnami/mariadb bitnami/mariadb
```

Спасибо!