Локальное окружение на Docker

Выбор пути

- ОС разработчика
- Стенды для разработки
- Образы виртуальных машин
- Контейнеры

#1 Каждый разрабочик всё устанавливает себе сам

- Комфортно
- Везде localhost
- Можно копипастить из мануала по установке

- Как правило ручная настройка
- Разные ОС, дистрибутивы
- Разные проекты или разные части поекта требуют разные версии ПО
- Версии, конфиги ПО нигде не закреплены, не хранятся в репозитории
- Поддержка legacy- проектов

"- Баг?? А у меня работает!"

#2 Стенды-песочницы для разработчика

➕
- Можно поднять до прихода нового сотрудника
- Возможность использовать систему управления конфигурацией(Ansible, Chef, Puppet...) для настройки нескольких машин
- Полная изоляция
- Всё делает админ, а не разработчик

➖
- Дорого
- Разработчики с root-правами: бардак, только админ с root: админ постоянно отвлекается
- В идеале это один стенд одного проектам для одного разработчика (т.е. в сумме n x m стендов)

🟰 Dev: "- А можешь ещё поставить Redis?" Ops: "- Нет! Занят!!"

#3 Локальные VM (Vagrant)

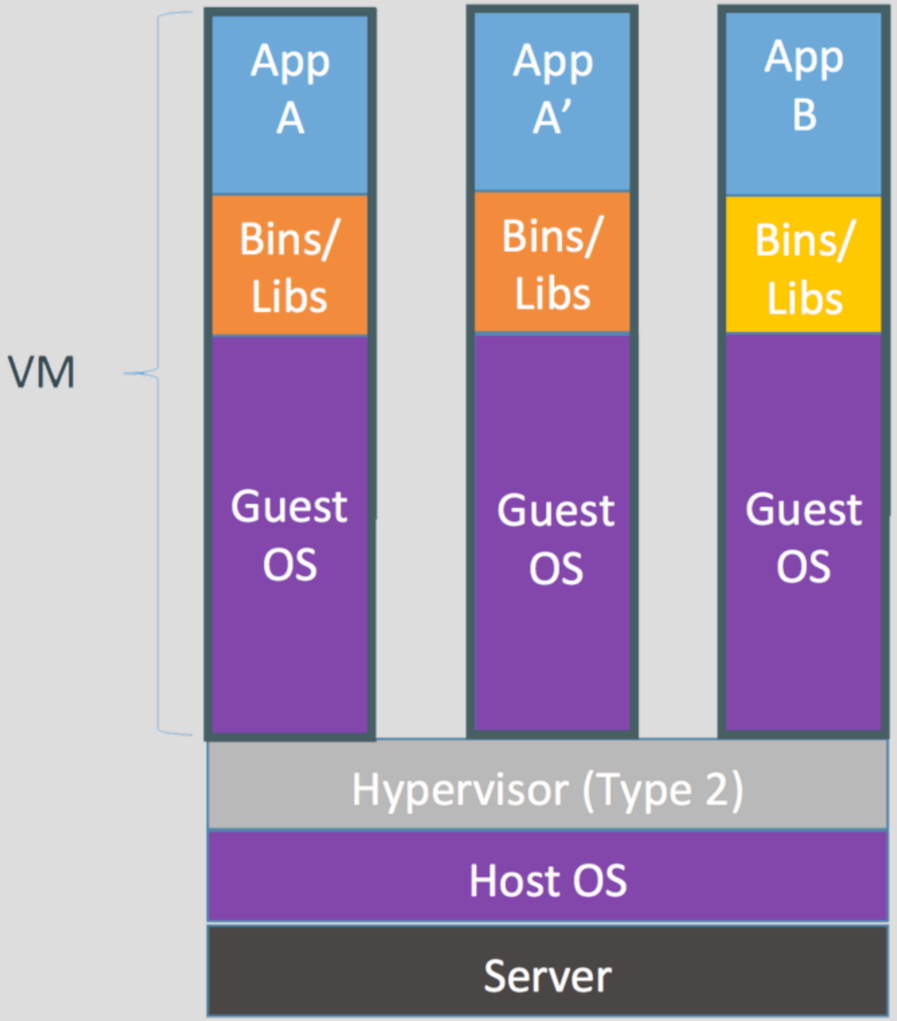
- Изоляция разных проектов в локальном окружении
- Возможность использовать систему управления конфигурацией для настройки и хранить её в общем репозитории т.е. у всех один и тот же конфиг
- Настройка окружения с Symfony есть в официальной документации

- Долгая установка и конфигурация
- Это ОС: требует много памяти на диске, много оперативной памяти, много процессорных ресурсов
- Очень сложно воссоздать распределённую систему

-“О! А этот конфиг, это также как на продакшне?" -“Нет... :("

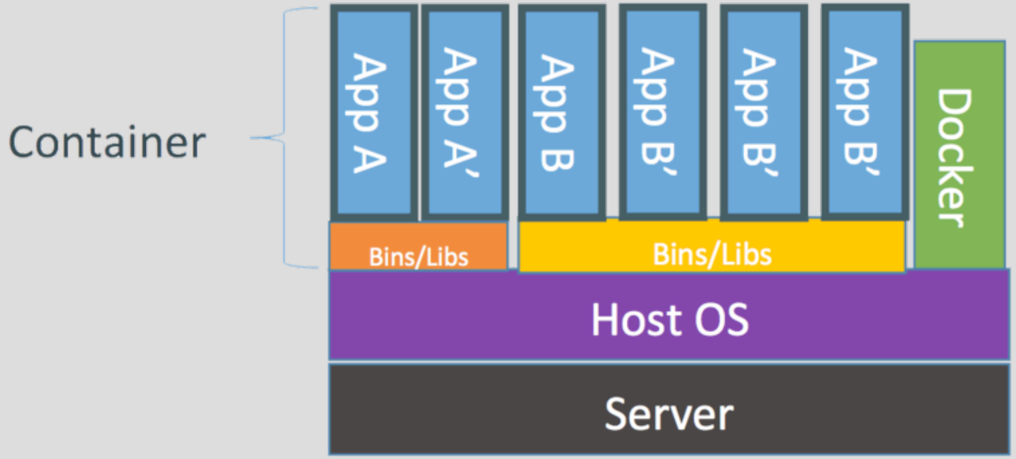
Docker vs Vagrant: отличия

Docker использует виртуализацию на уровне процессов ОС

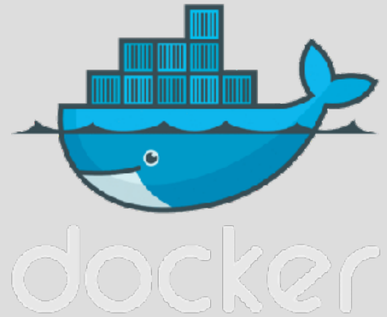


Containers are isolated, but share OS and, where appropriate, bins/libraries

...result is significantly faster deployment, much less overhead, easier migration, faster restart

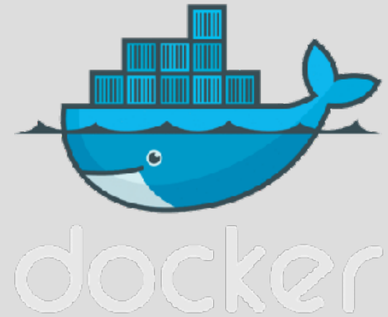


Docker: инструменты



- docker-engine: центральная часть докера, демон работающий на хост-машине и умеющий скачивать и заливать образы, запускать из них контейнеры, следить за запущенными контейнерами, собирать логи и настраивать сеть между контейнерами (а с версии 0.8 и между машинами).
- docker: консольная утилита для управления docker-демоном по HTTP
- docker-compose: инструмент для создания и запуска мульти- контейнерных докер- приложений
- docker-swarm: нативное решение докера для создания кластеров
- docker-machine: набор инструментов для запуска контейнеров на различных инфраструктура-как-сервис платформах, а также VMWare, Virtualbox итд
- Docker Hub, Docker Store: репозиторий для хранения образов контейнеров (неофициальные и официальные)

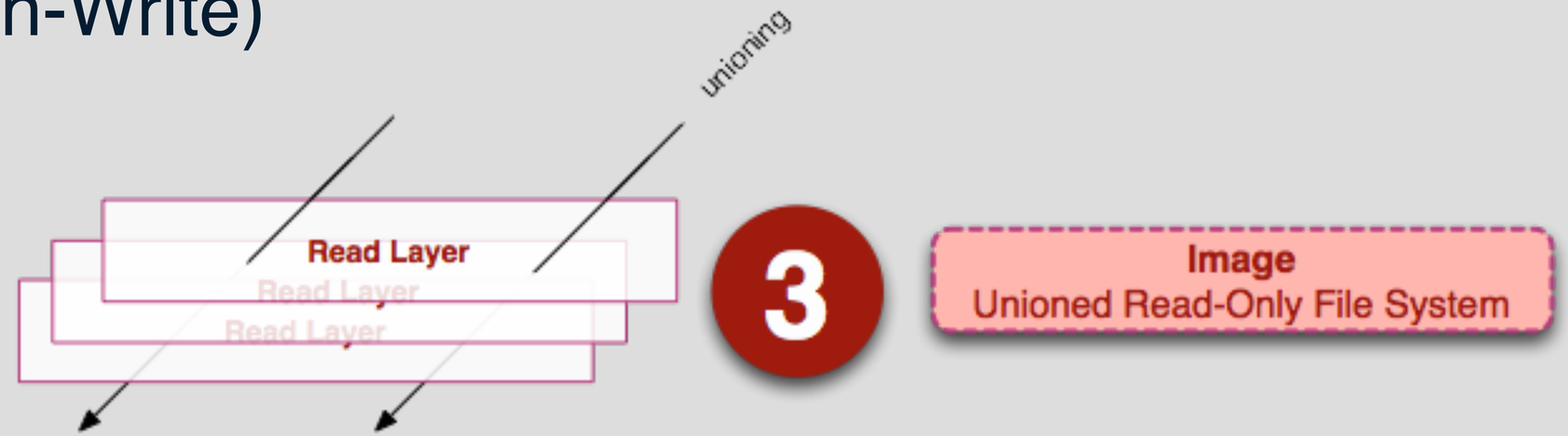
Docker: основные понятия



- Image: упакованная версия приложения вместе с зависимостями и необходимым окружением необходимым для запуска (как класс в ООП)
- Container: контейнер это запущенный экземпляр образа (как объект в ООП)
- Volume: область постоянного хранения файлов, данные не теряются при пересборке или удалении конейнера
- Dockerfile: файл конфигурации контейнера для сборки

Docker: внутреннее устройство

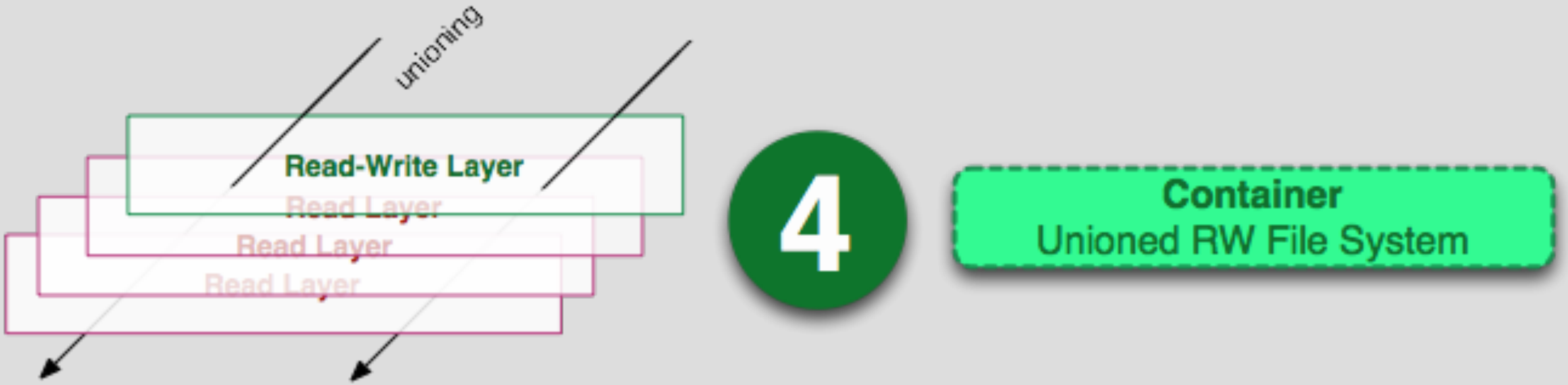
- Инструменты изоляции(наподобии chroot jail): LXC(<0.9), libcontainer (runc, >=0.9)
- "слоёная" фаловая система AUFS (переиспользуемость, Copy-On-Write)



Образ содержит слои(снепшоты) каждого отдельного изменения внутреннего содержания.

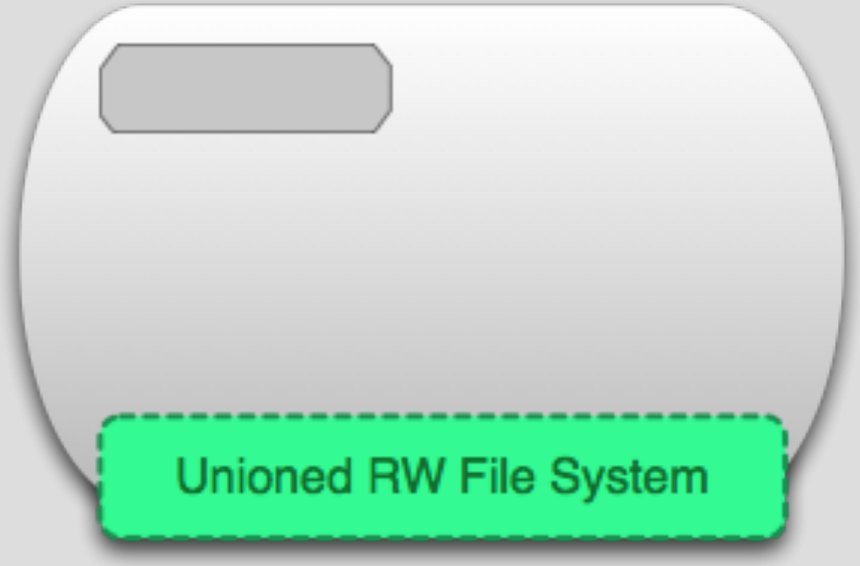
Docker: внутреннее устройство

Как только мы создаём контейнер мы ставим наверх пустой Read-write слой

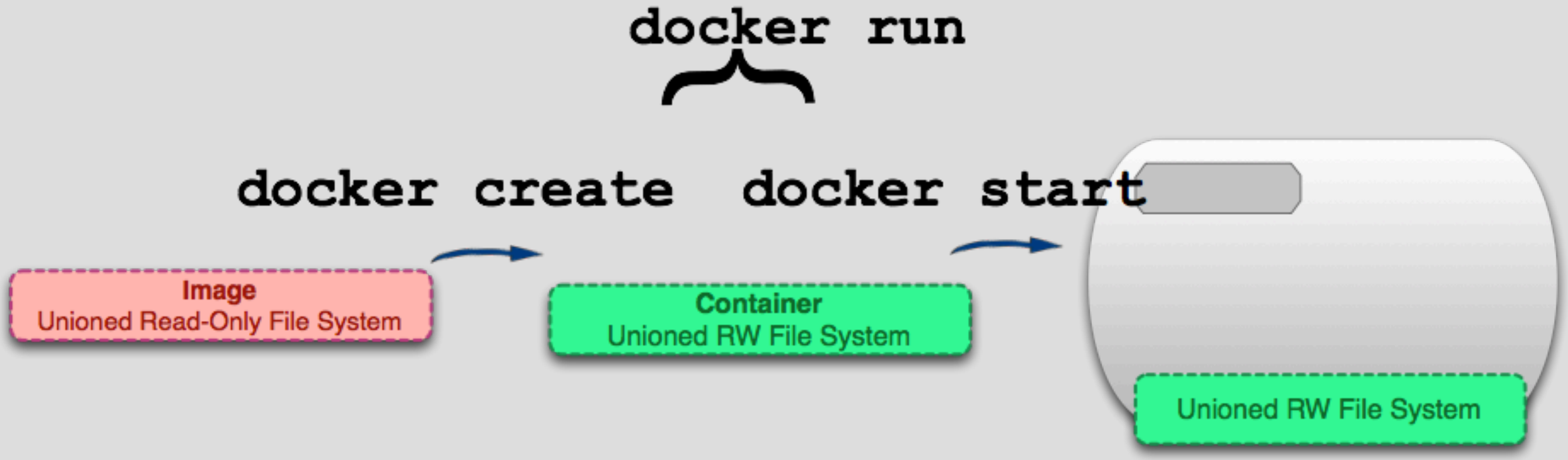


Docker: внутреннее устройство

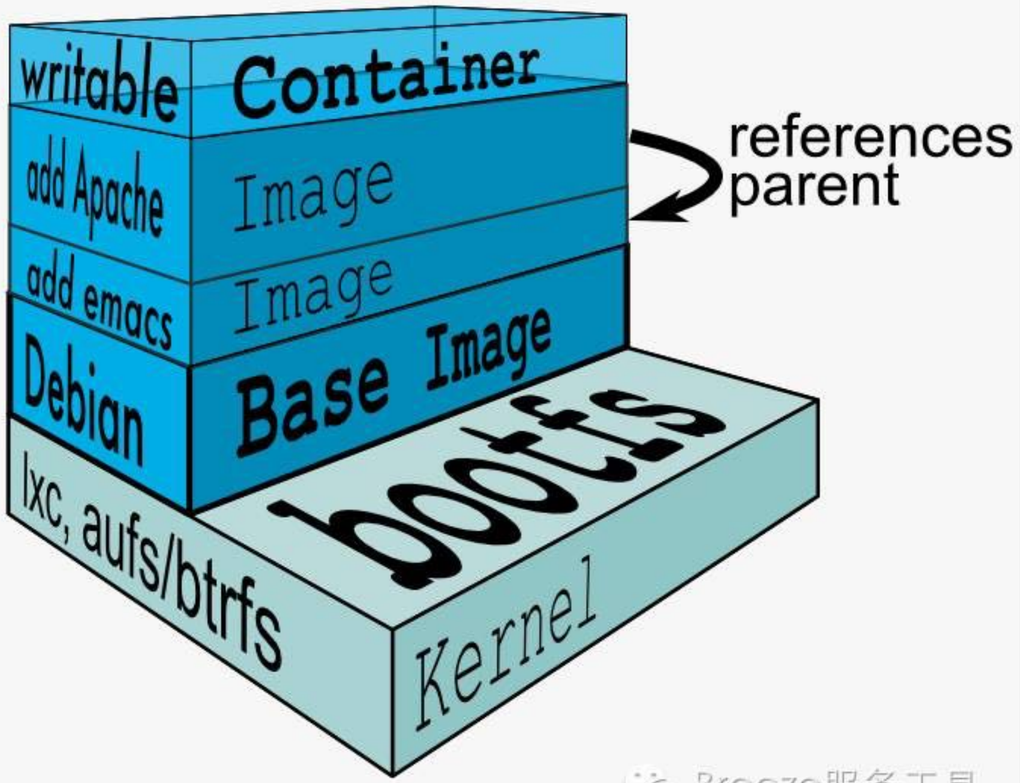
Запущенный контейнер можно описать как данный объединённый слой доступный для записи и изолированное от хост- машины пространство процессов с процессом внутри (исходя из концепции докера он должен быть только один)

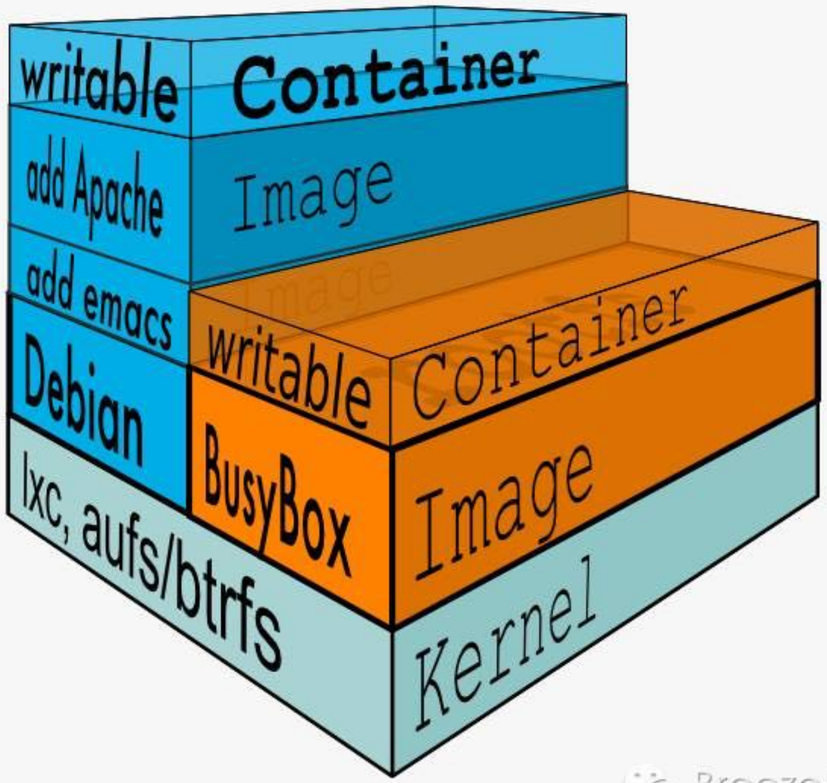


Docker: внутреннее устройство



Docker: внутреннее устройство





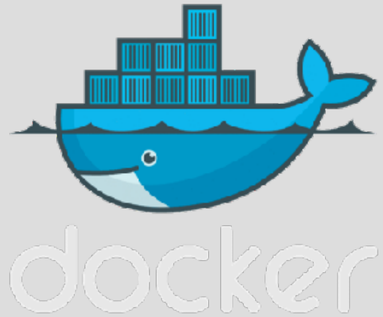
Dockerfile

- FROM - Указывается базовый образ
- ADD - Копирование файлов с локальной машины
- RUN - Запуск команды внути контейнера, например apt-get install
- EXPOSE - Открытие порта для внешнего доступа
- CMD - Команда для основного процесса контейнера (может быть только одна CMD)
- ENTRYPOINT - Executable для запуска CMD, по умолчанию "/bin/sh -c”

  в "docker run -i -t ubuntu bash": FROM=ubuntu CMD=bash ENTRYPOINT=/bin/sh -c
- USER - Установить юзера для RUN, CMD и ENTRYPOINT (по умолчанию, root)

- Команды выполняются декларативно (порядок имеет значение)

#4 Контейнеры (Docker)



- Изоляция на уровне процессов
- Можно точь-в-точь повторить продакшн локально
- Хорошо подходит для описания микросервисов, локально можно разрабатывать, дебажить несколько сервисов одновременно
- Чем крупнее проект, тем больше прибыль от внедрения докера

- Порог вхождения довольно высокий, сложнее чем другие варианты
- Продукт очень бурно развивался, в сети много примеров с конфигурацией прошлых версий и вовсе нерабочих

Единообразие

TIPS AND TRICKS

Образы или сборка локально

Для разработки в репозитории удобнее хранить исходники контейнеров: Dockerfile от образа официального репозитория дистрибутива (base image)

Там где достаточно дефолтной конфигурации (например, сборка логов) удобно брать готовые официальные образы.

Логирование в стиле Docker

Переполнение файла логов docker

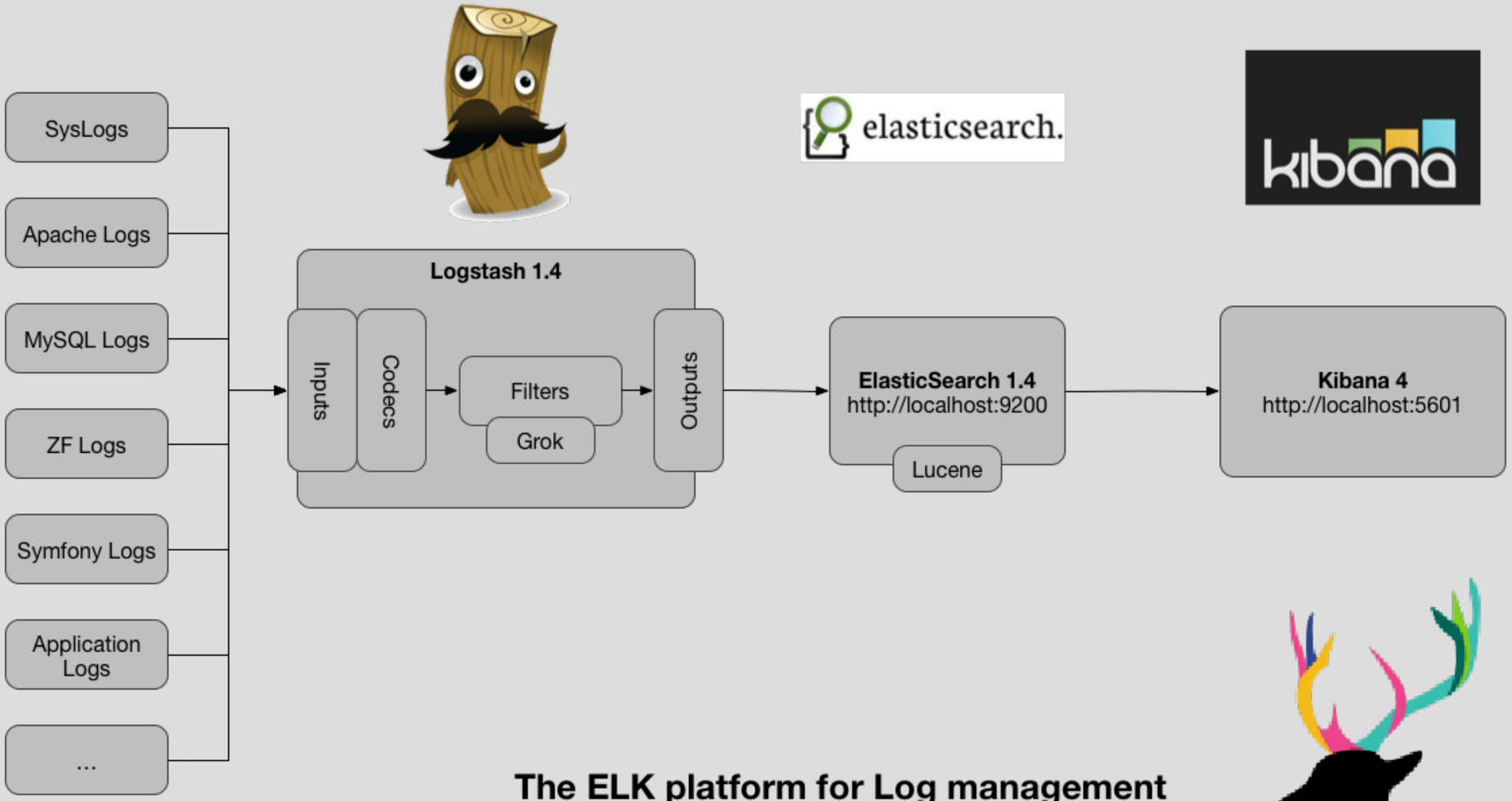
```
λ ~/ sudo find /var/lib/docker/containers -type f -name '*.log' -exec du -ch {} + | grep total
61G     total
```

Варианты:

- Использовать logrotate: /var/lib/docker/containers/*/*.log
- Или выключить сохранение логов: --log-driver=none

Логирование в стиле Docker

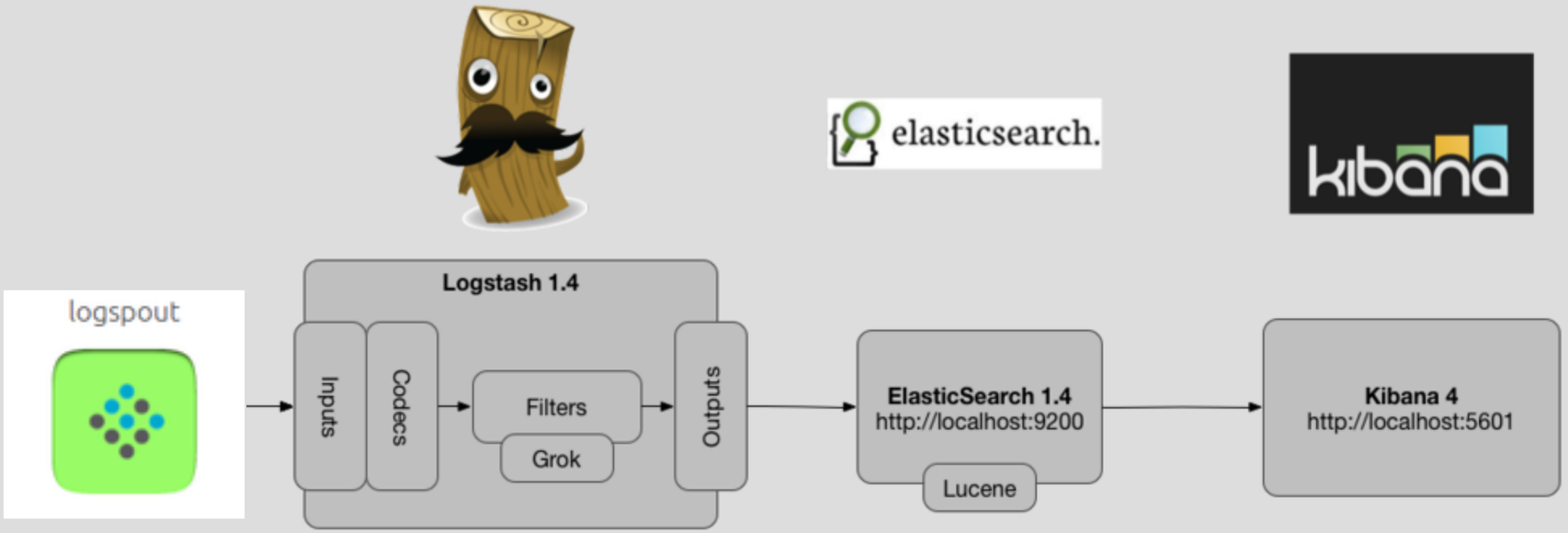
ELK stack



The ELK platform for Log management

Логирование в стиле Docker

Logspout! собирает сообщения из STDIN и STDERR контейнеров, которые мы можем прочитать в Logstash



The ELK platform for Log management

Типы томов (volumes)

- Bind-mounts — монтирование внешних папок /host:/container
- Data-only containers - сохранение данных из контейнера /container, использование в других контейнерах с помощью --volumes-from(deprecated)
- Named volumes (>=1.10) - возможность закрепить имя за томом volume_name:/container, не нужен volumes-from.
- После первого использования том получает правильные права доступа и наполняется контентом (initial volume seeding)

Переменные окружения

Для того чтобы удобно собирать и пользоваться окружением на разных машинах, можно использовать переменные окружения и класть их в .env

Можно пересобирать PHP с Xdebug или Blackfire в зависимости от надобности

```
COMPOSE_PROJECT_NAME=test

# Applications path(absolute or relative)
BACKEND_PATH=<CHANGE_ME>
FRONTEND_PATH=<CHANGE_ME>
SEARCH_SERVICE_PATH=<CHANGE_ME>

# MariaDB
MARIADB_DB=<CHANGE_ME>
MARIADB_USER=<CHANGE_ME>
MARIADB_PASS=<CHANGE_ME>

# RabbitMQ
RABBITMQ_USER=<CHANGE_ME>
RABBITMQ_PASS=<CHANGE_ME>

# Extra hosts
REMOTE_HOST=frontend.spir.dev:<CHANGE_ME>

# Build
PHP_CONTAINER_VERSION=php-xdebug

...
```

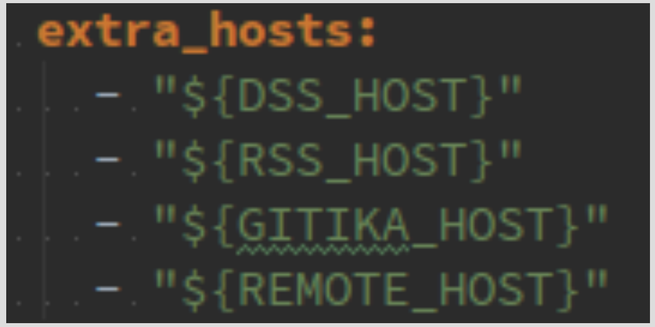
networks: vs links:

Вместо того чтобы указывать строго определённые связи links(deprecated) от контейнера к контейнеру, окружить весь проект одной bridge сетью
Docker-compose зарезолвит имена контейнеров в IP адреса автоматически


```
"Networks": {
    "spir_default": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": [
            "c75d3a656050",
            "php"
        ],
        "NetworkID": "3a0187c73dfcf27ffe96dcf8a87a78ded8fb442ec8c69db0c5dc2e52846dbe7d",
        "EndpointID": "48763c4263988f2cbf43a7daafba9523984bcc8a14e7b1178e9ee8aa73260dff",
        "Gateway": "172.18.0.1",
        "IPAddress": "172.18.0.8",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:ac:12:00:08"
    }
}
```

Внешние сервисы и доступ к хост машине

Для доступа к внешним сервисам, а так же хост машине пока приходится использовать extra_hosts, прописывающий их в /etc/hosts контейнера



Запущенный контейнер != готовый к работе контейнер

Wait_for_it в entrypoint:

```
#!/bin/bash
WAIT_FOR=${WAIT_FOR:=""}
WAIT_SLEEP=${WAIT_SLEEP:="1"}

# wait until is ready
if [ ! -z "$WAIT_FOR" ]; then
    IFS=', ' read -r -a WAIT_FOR_CONTAINER <<< "$WAIT_FOR"
    for var in "$@"
    do
        IFS=: read HOST PORT <<< "$WAIT_FOR_CONTAINER"
        while true; do
            nmap -Pn -p"$PORT" "$HOST" | awk "\$1 ~ /$PORT/ {print \$2}" | grep open
            if [ $? -eq 0 ]; then
                break
            fi
            sleep "$WAIT_SLEEP"
        done
    done
fi

echo -e "\e[1;48;3;33mDependencies ready, starting master process.\e[0m"
set -e
exec "$@"
```