



# Технологии контейнеризации:

## Введение

В рамках данного выступления рассматривается использование сканеров уязвимостей, сравнение и примеры работы с ними в различных конфигурациях. В частности, в фокусе выступления – обеспечение безопасности контейнеров в рамках разработки, тестирования и внедрения программных решений, направленных на развитие обеспечения безопасности конвейера DevSecOps.

# Содержание

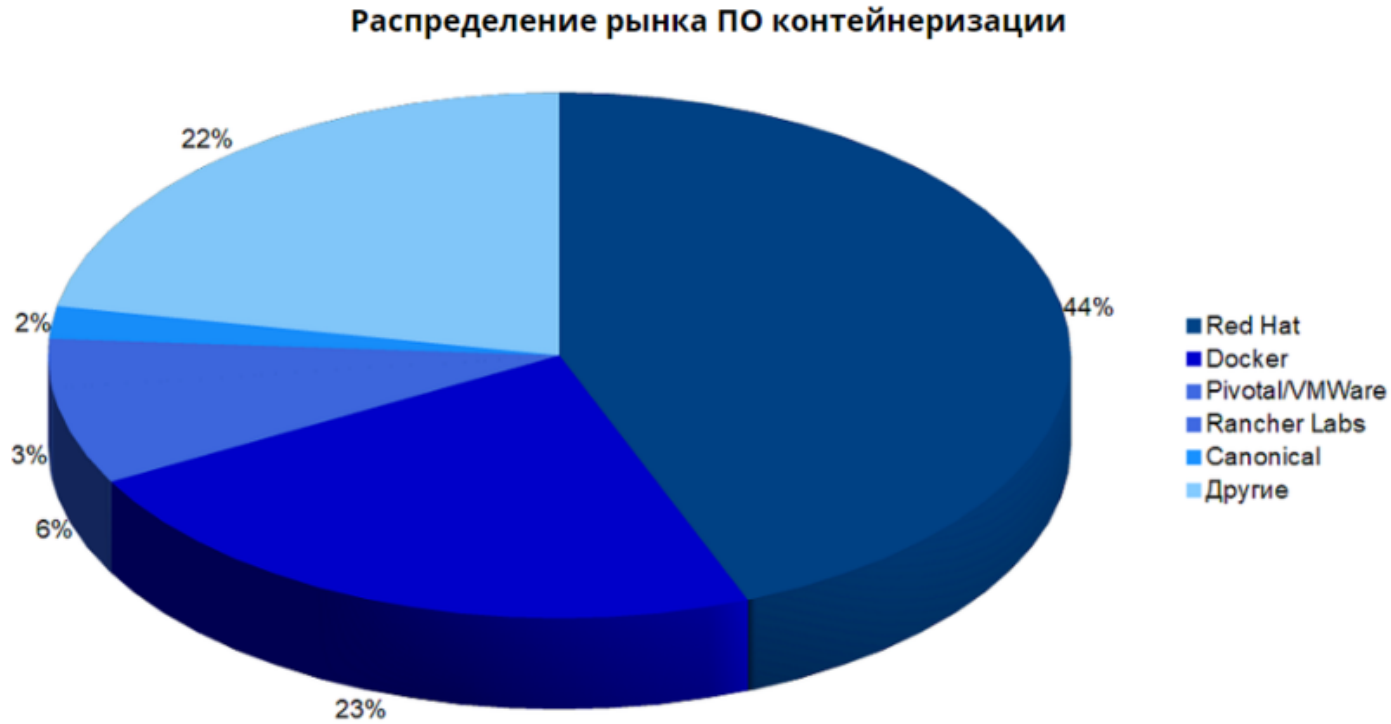
## 1. Введение

1. Системы контейнеризации
2. Контейнер и образ
3. Docker
4. Docker vs VM
5. Установка Docker и Docker-compose на Windows 10
6. Установка Docker и Docker-compose на Ubuntu и Debian
7. Задание: Проверка корректного завершения установки

## 1. Основы создания образов

2. Сетевые механизмы Docker
3. Docker-compose
4. Docker Swarm & Swarm Classic
5. Сервисы Docker
6. Kubernetes
7. Dockerfile
8. Основные инструкции
9. Создание образов
10. Как уменьшить размер образа? Как ускорить сборку?

# Системы контейнеризации



Источник: IHS Markit, 2019

Технологии контейнеризации приложений помогают сделать приложения более безопасными, облегчают их развертывание и улучшают возможности по их масштабированию.

# Системы контейнеризации

контейнер делит с хостом:

- ядро
- пространство памяти ядра

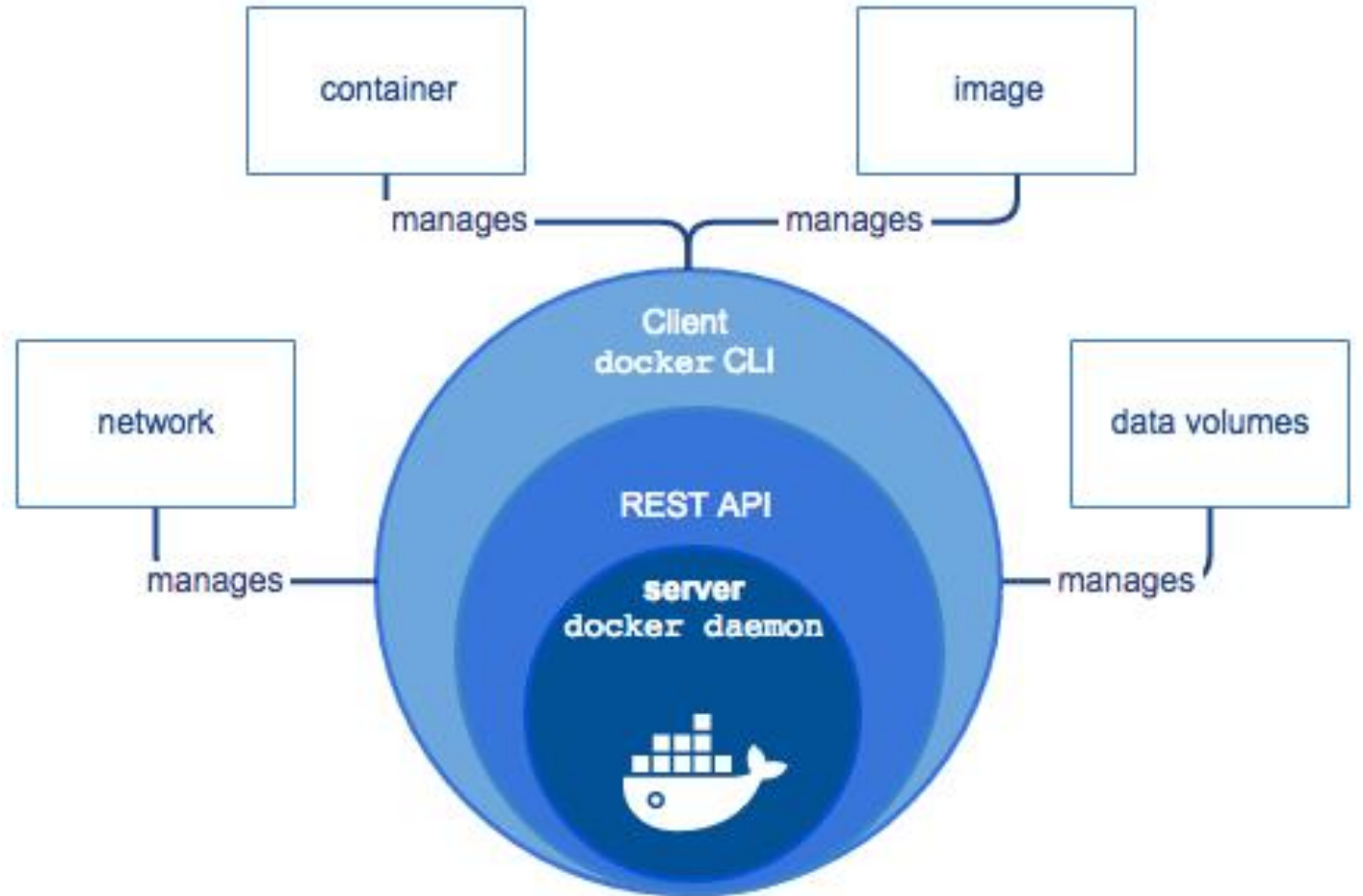
контейнер изолирует:

- пользовательское окружение

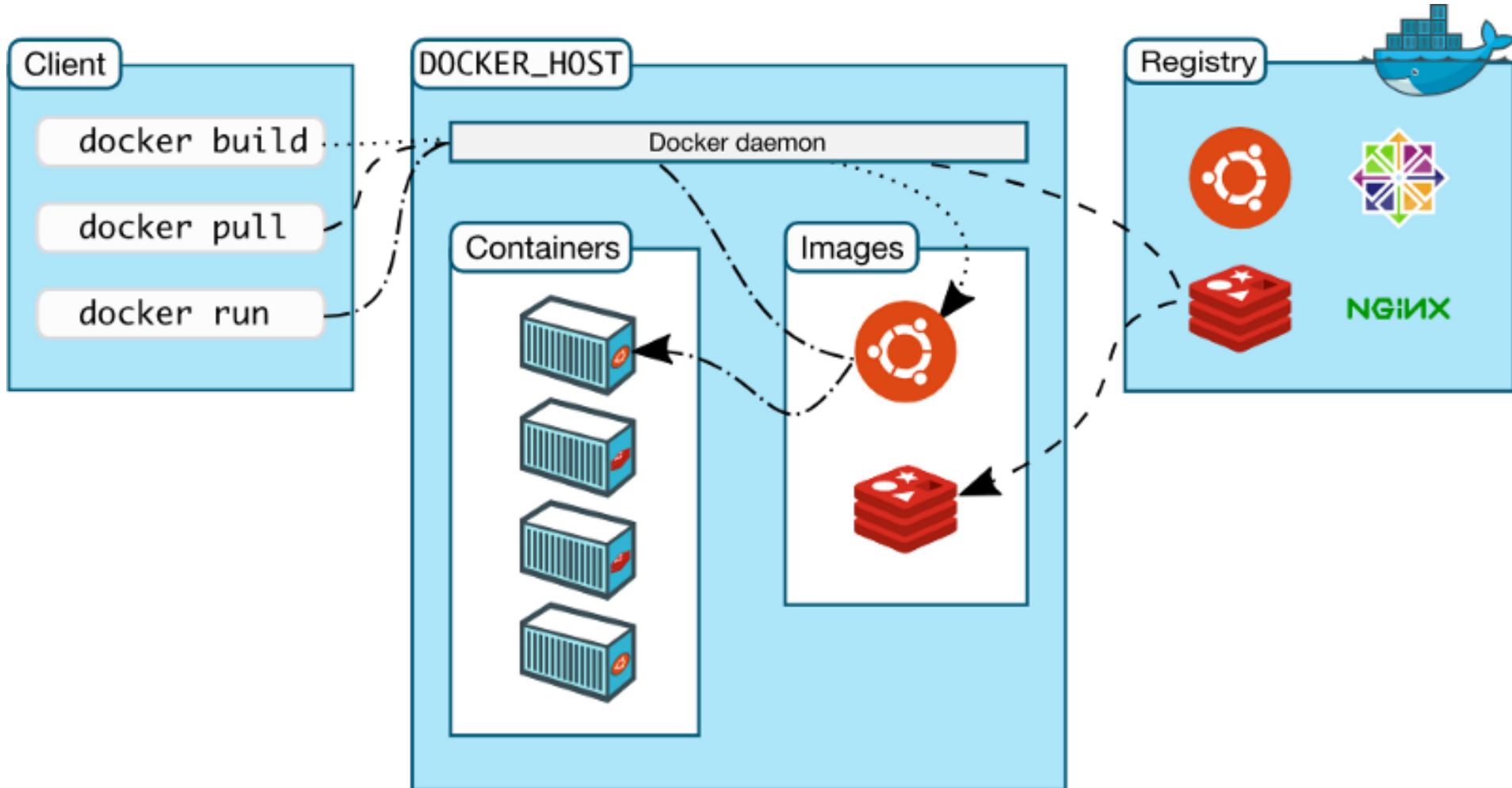
контейнер использует для изоляции возможности операционной системы (пространство имен). Применительно к Docker используются так называемые `cgroups` в ядре Linux.

# Docker

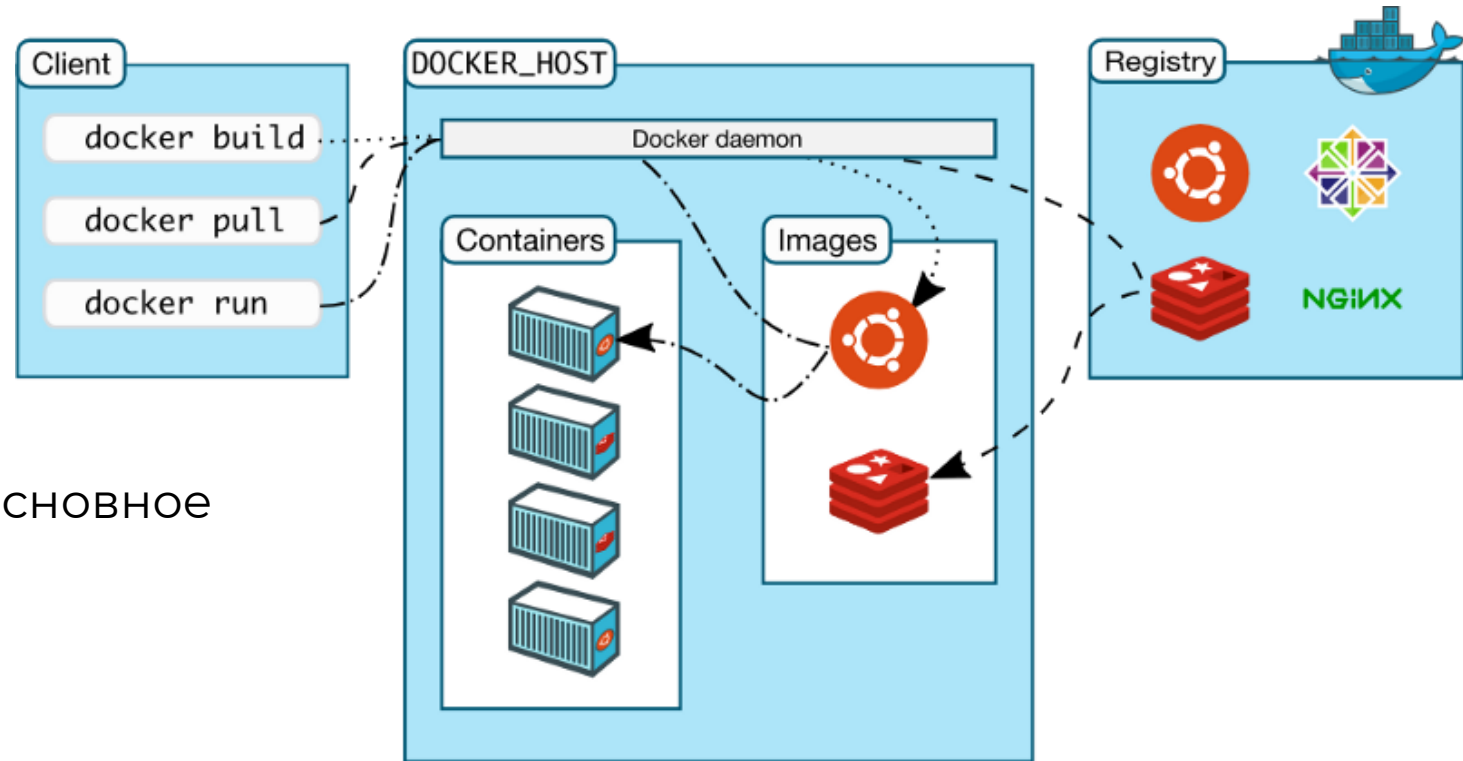
Docker (Docker Engine) - это одна из систем, позволяющих контейнеризировать приложения, предназначена для разработки, развертывания и запуска приложений в контейнерах



# Docker



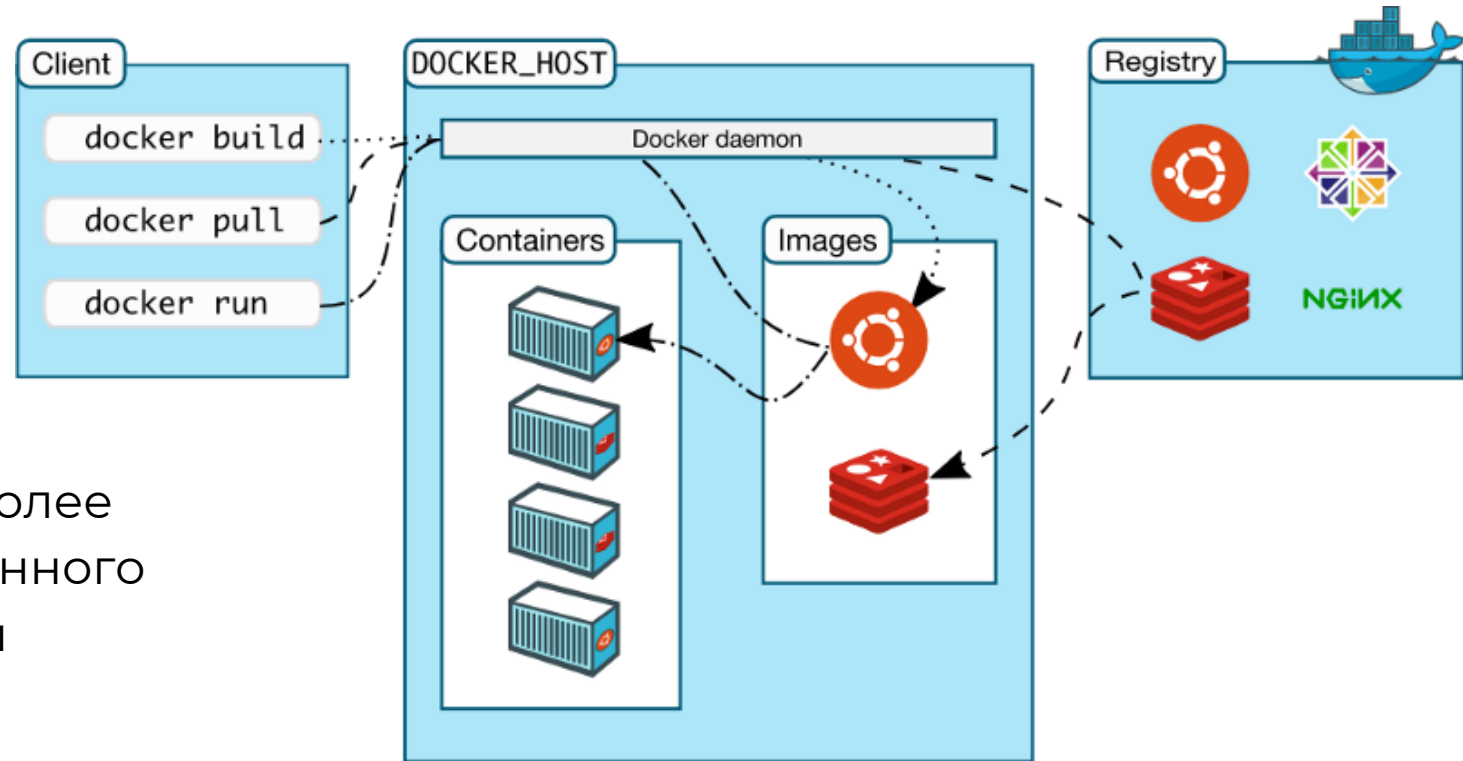
# Docker



Клиент Docker (Docker Client) — это основное средство, которое используют для взаимодействия с Docker

Демон Docker (Docker Daemon) — это сервер Docker, который ожидает запросов к API Docker. Демон Docker управляет образами, контейнерами, сетями и томами

# Docker

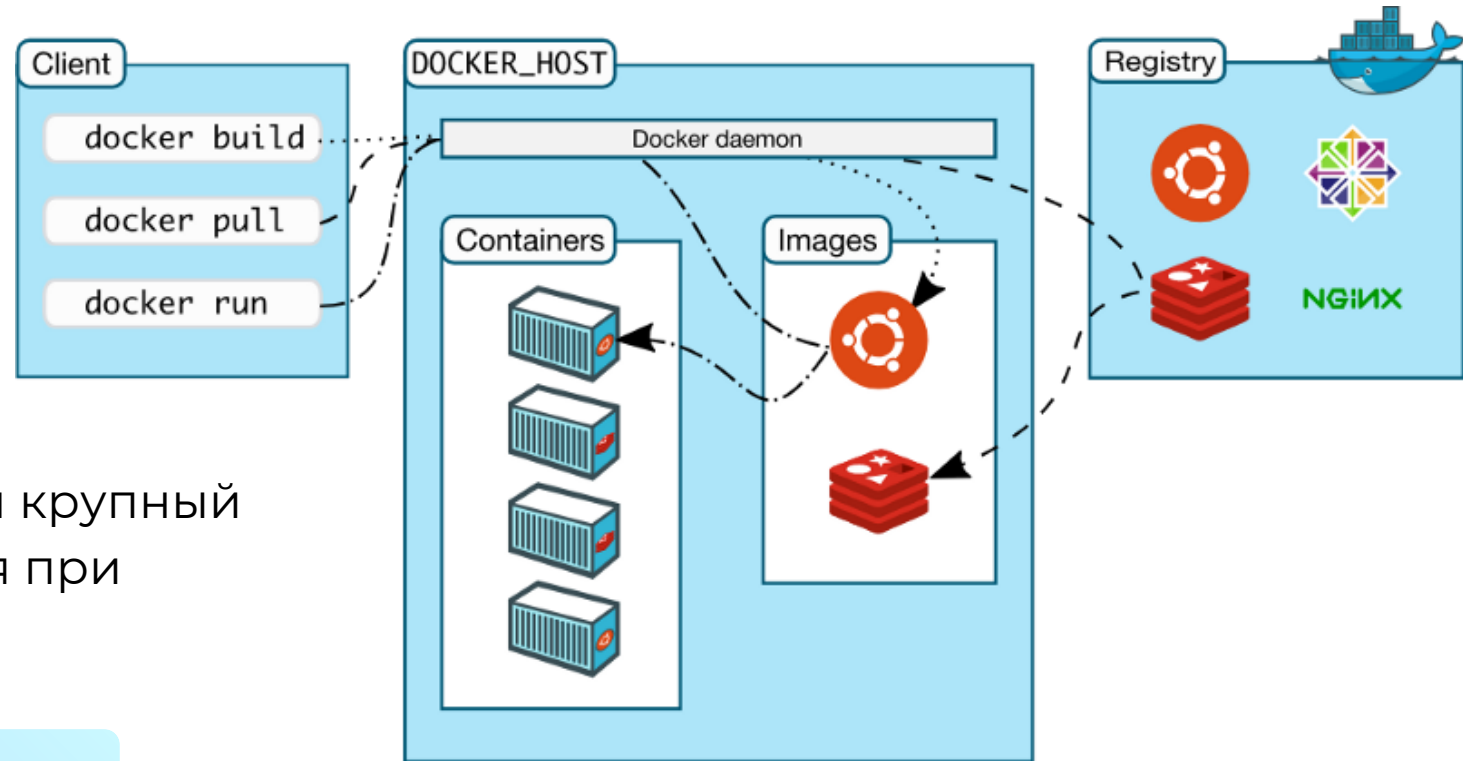


Тома Docker (Docker Volumes) – наиболее предпочтительный механизм постоянного хранения данных, потребляемых или производимых приложениями.

Реестр Docker (Docker Registry) – удаленная платформа, используемая для хранения образов Docker. В ходе работы с Docker образы отправляют в реестр и загружают из него.



# Docker



Docker Хаб (Docker Hub) — это самый крупный реестр образов Docker, используется при работе с Docker по умолчанию.

Репозиторий Docker (Docker Repository) – набор образов Docker, обладающих одинаковыми именами и разными тегами.

Теги — идентификаторы образов.

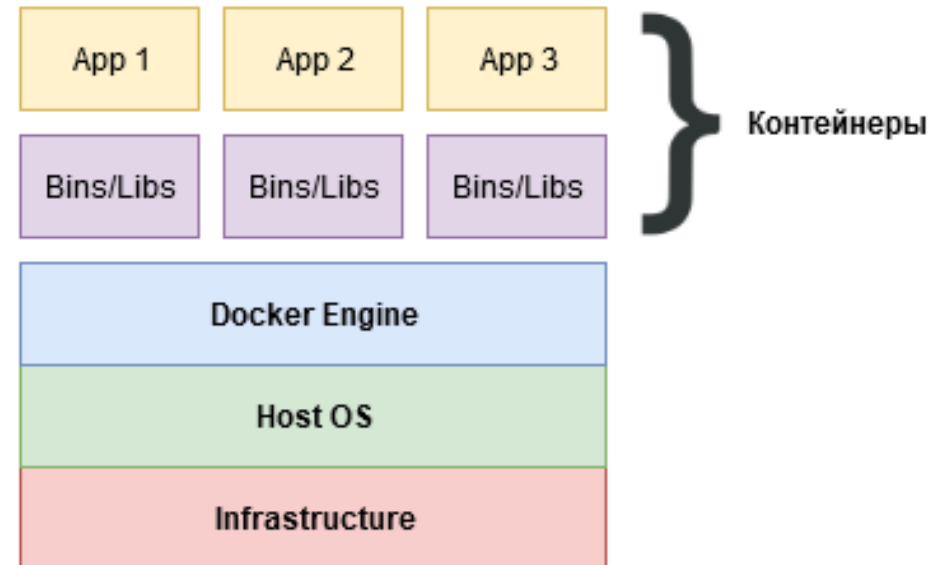
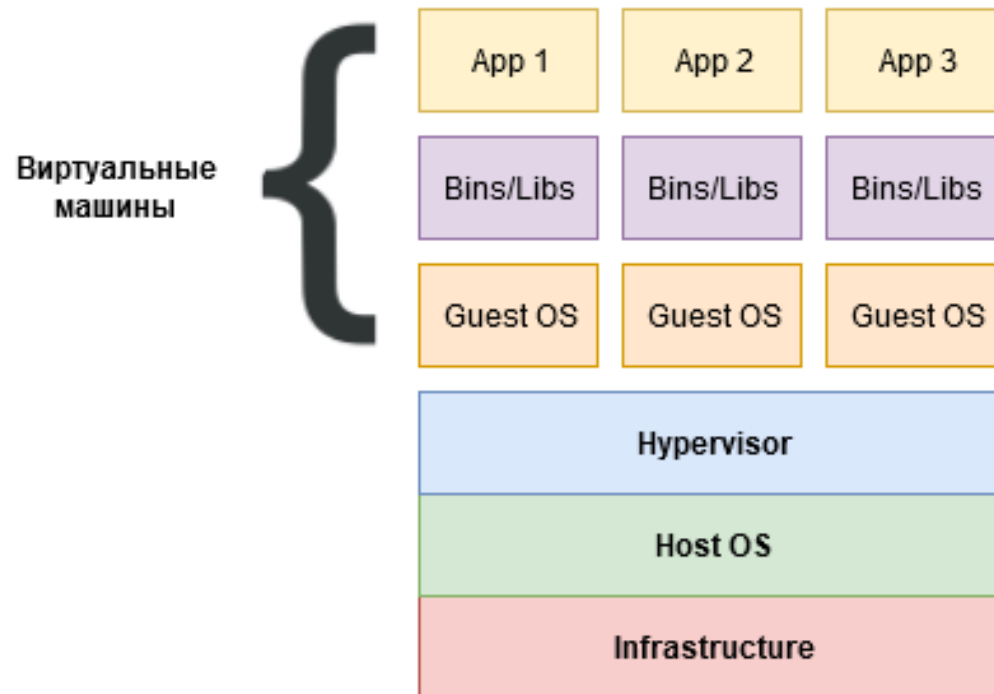
# Docker

**Контейнер** — среда, где разворачивается и запускается сохраненный заранее образ с отдельным ПО \ программой \ сервисом.

**Образ** — полностью работоспособный исполняемый пакет определенного ПО для запуска в контейнере Docker.

**Dockerfile** — файл инструкций для демона Docker под соответствующий образ. Для нового образа, сначала готовится Dockerfile после чего сам пакет образа.

# Docker vs VM



# Docker vs VM

	Docker	Виртуальные машины (VM)
Время загрузки	Загрузка через несколько секунд.	Загрузка виртуальных машин занимает несколько минут.
Работает на	Docker используют механизм исполнения.	ВМ используют гипервизор.
Эффективность памяти	Для виртуализации не требуется места, а значит, и меньше памяти.	Требуется загрузка всей ОС перед запуском поверхности, поэтому она менее эффективна.
Изоляция	Нет условий для систем изоляции.	Возможность вмешательства минимальна из-за эффективного механизма изоляции. Изолируется средствами процессора.
Развертывание	Развертывание легко, так как только одно изображение в контейнере может использоваться на всех платформах.	Развертывание сравнительно длительное, поскольку за выполнение отвечают отдельные экземпляры.

# Установка Docker и Docker-compose на Windows 10

## Системные требования

- Windows 10 64-bit: Pro, Enterprise, Education (Build 16299 или выше).

Для успешного запуска Client Hyper-V в Windows 10 требуются следующие предварительные требования к оборудованию:

- 64 bit процессор с поддержкой Second Level Address Translation (SLAT).
- 4GB системной памяти.
- Поддержка аппаратной виртуализации на уровне BIOS должна быть включена в настройках BIOS.

# Установка Docker и Docker-compose на Windows 10

## Подготовка

### Включаем функции Hyper-V Containers Window

Панель управления -> установка и удаление программ -> включение или отключение компонентов Windows.

Активируем пункт Hyper-V, который включает Hyper-V Managment Tools, Hyper-V Platform.

это можно выполнить через powershell или dism (необходимы права администратора).

#### **Powershell:**

```
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V -All
```

#### **DISM:**

```
DISM /Online /Enable-Feature /All /FeatureName:Microsoft-Hyper-V
```

# Установка Docker и Docker-compose на Windows 10

## Установка

Скачиваем установщик Docker (Docker Desktop Installer) с Docker Hub:

<https://desktop.docker.com/win/main/amd64/Docker%20Desktop%20Installer.exe>

Установка Docker Desktop включает Docker Engine, Docker CLI client, Docker Compose, Notary, Kubernetes и Credential Helper.

Запускаем установщик Docker Desktop Installer.exe и ожидаем пока он скачает все необходимые компоненты. После установки система потребует перезагрузки. Перезагружаемся и входим в систему.

После входа может возникнуть запрос на установку дополнительного компонента WSL2. Переходим по ссылке и скачиваем необходимый пакет с официального сайта Microsoft.

После скачивания выполняем установку WSL2, после которой снова потребуется перезагрузка.

# Установка Docker и Docker-compose на Windows 10

## Установка

Скачиваем установщик Docker (Docker Desktop Installer) с Docker Hub:

<https://desktop.docker.com/win/main/amd64/Docker%20Desktop%20Installer.exe>

Установка Docker Desktop включает Docker Engine, Docker CLI client, Docker Compose, Notary, Kubernetes и Credential Helper.

Запускаем установщик Docker Desktop Installer.exe и ожидаем пока он скачает все необходимые компоненты. После установки система потребует перезагрузки. Перезагружаемся и входим в систему.

После входа может возникнуть запрос на установку дополнительного компонента WSL2. Переходим по ссылке и скачиваем необходимый пакет с официального сайта Microsoft.

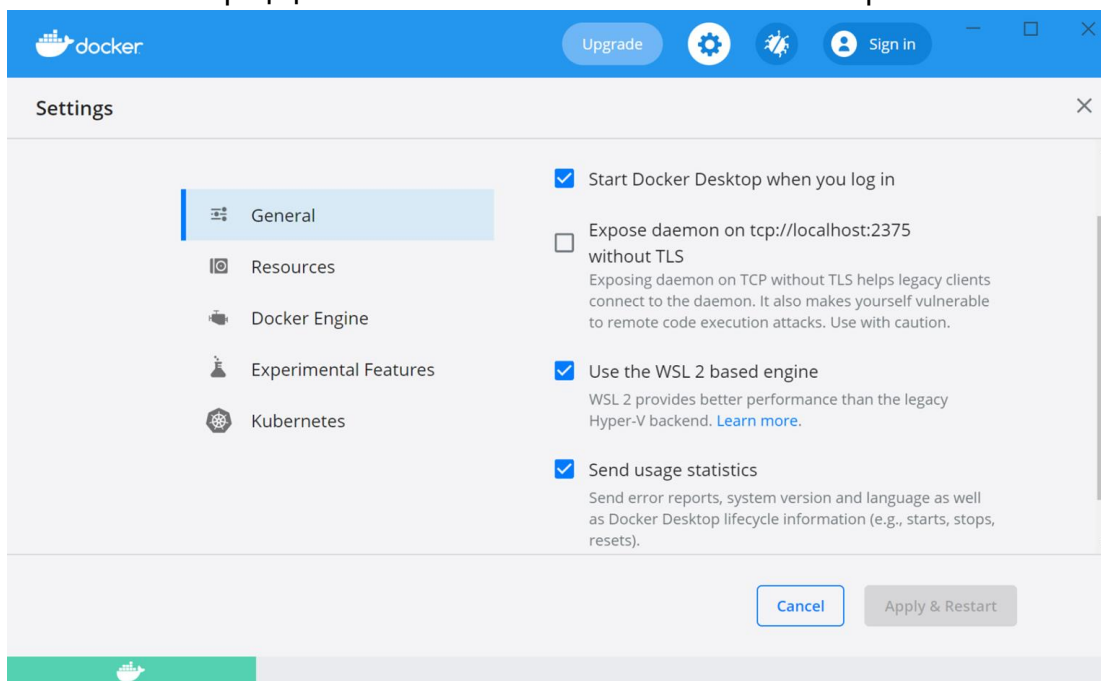
После скачивания выполняем установку WSL2, после которой снова потребуется перезагрузка.



# Установка Docker и Docker-compose на Windows 10

## Настройка и запуск

Входим в систему и ждем запуска всех служб Docker. Когда все службы будут запущены, мы увидим в трее классический значок Docker — это значит что служба установлена и запущена. Далее можно запустить приложение Docker desktop. Далее можно изменить настройки Docker при необходимости:



# Установка Docker и Docker-compose на Ubuntu и Debian

1. Проведите пакетное обновление сервера:

```
$ sudo apt update
```

1. Установите зависимости:

```
$ sudo apt install apt-transport-https ca-certificates curl gnupg2 software-properties-common
```

1. Добавьте ключ GPG от официального репозитория в систему управления пакетами APT :

```
$ curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -  
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian $(lsb_release -cs) stable"
```

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -  
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

1. Обновите базы данных индекс-пакетов APT системы и переключитесь в репозиторий Docker:

```
$ sudo apt update  
$ apt-cache policy docker-ce
```

1. Установить Docker:

```
$ sudo apt install docker-ce
```

# Проверка корректного завершения установки

```
$ docker run hello-world
```

>hello  
world

```
$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

$ docker images hello-world
REPOSITORY    TAG       IMAGE ID      SIZE
hello-world   latest    feb5d9fea6a5 13.26kB
```

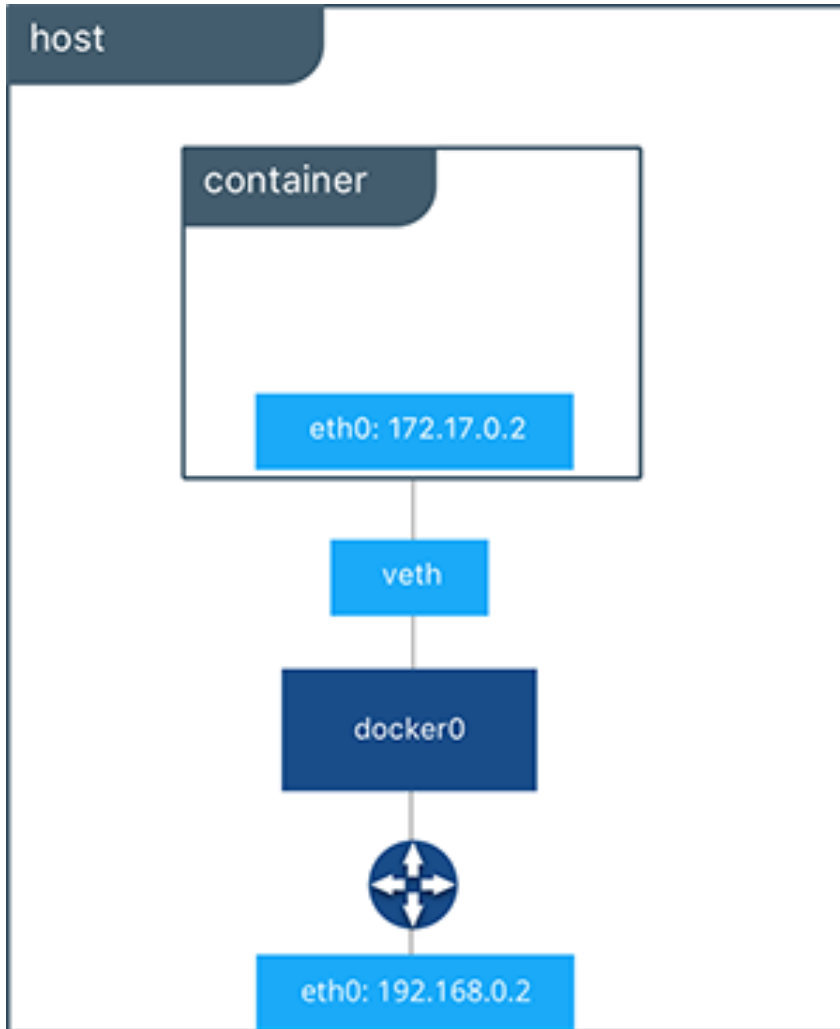
>hello  
world



# Обеспечение безопасности контейнеров: Основы создания образов

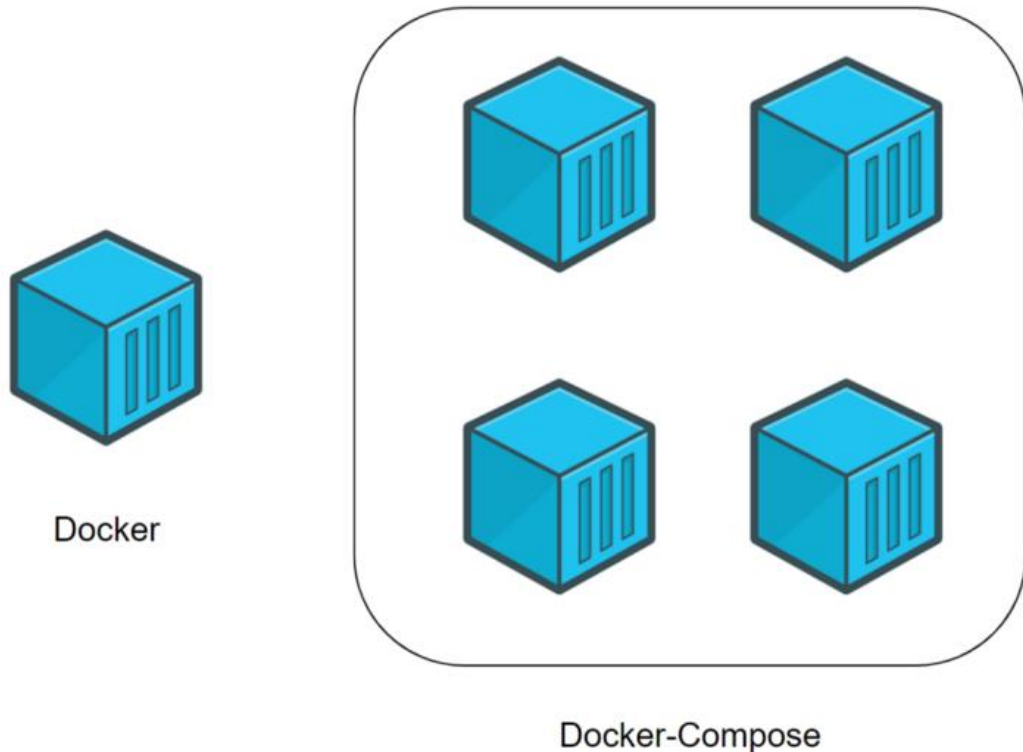
В рамках данного выступления рассматривается использование сканеров уязвимостей, сравнение и примеры работы с ними в различных конфигурациях. В частности, в фокусе выступления – обеспечение безопасности контейнеров в рамках разработки, тестирования и внедрения программных решений, направленных на развитие обеспечения безопасности конвейера DevSecOps.

# Сетевые механизмы Docker



- Сетевые механизмы Docker (Docker Networking) позволяют организовывать связь между контейнерами Docker.
- Соединенные с помощью сети контейнеры могут выполняться на одном и том же хосте или на разных хостах.

# Docker-compose



- Docker Compose — инструмент, упрощающий развёртывание приложений, для работы которых требуется несколько контейнеров Docker, позволяет выполнять команды, описываемые в файле `docker-compose.yml`. Эти команды можно выполнять столько раз, сколько потребуется.
- Интерфейс командной строки Docker Compose упрощает взаимодействие с многоконтейнерными приложениями.

# Docker Swarm & Swarm Classic



- Swarm Classic — нативная кластеризация Docker. Превращает пул докер-контейнеров в один виртуальный хост:

<https://github.com/docker-archive/classicswarm>

- Docker Swarm — это решение, предназначенное для управления контейнерными развертываниями (то есть, для оркестрации контейнеров):

<https://github.com/moby/swarmkit>

# Сервисы Docker

- Сервисы Docker (Docker Services) — это различные части распределенного приложения.
- Сервисы Docker позволяют масштабировать контейнеры в пределах нескольких демонов Docker, благодаря им существует и технология Docker Swarm.



# Kubernetes



**kubernetes**

- [Kubernetes](#) — это технология, которая позволяет автоматизировать развертывание и масштабирование контейнеризированных приложений, а также управление ими.
- Если вам нужен инструмент для работы с группами контейнеров, для масштабирования решений, основанных на них, используйте не Docker Swarm, а Kubernetes.
- Kubernetes не является частью Docker.

# Dockerfile

- Контейнер Docker — самодостаточная операционная система, в которой имеется только самое необходимое и код приложения.
- Образы Docker являются результатом процесса их сборки, а контейнеры Docker — это выполняющиеся образы.
- Dockerfile файлы сообщают Docker о том, как собирать образы, на основе которых создаются контейнеры.
- Каждому образу Docker соответствует файл, который называется Dockerfile.
- При запуске команды `docker build` для создания нового образа подразумевается, что Dockerfile находится в текущей рабочей директории. Если этот файл находится в каком-то другом месте, его расположение можно указать с использованием флага `-f`.

# Dockerfile

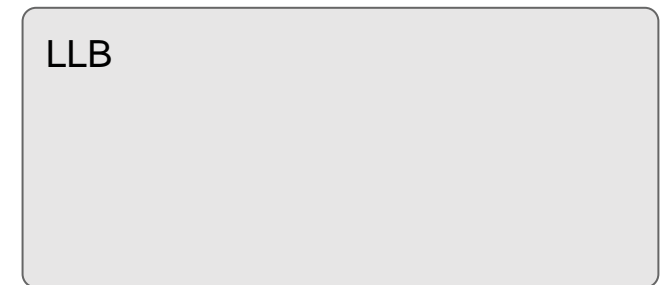
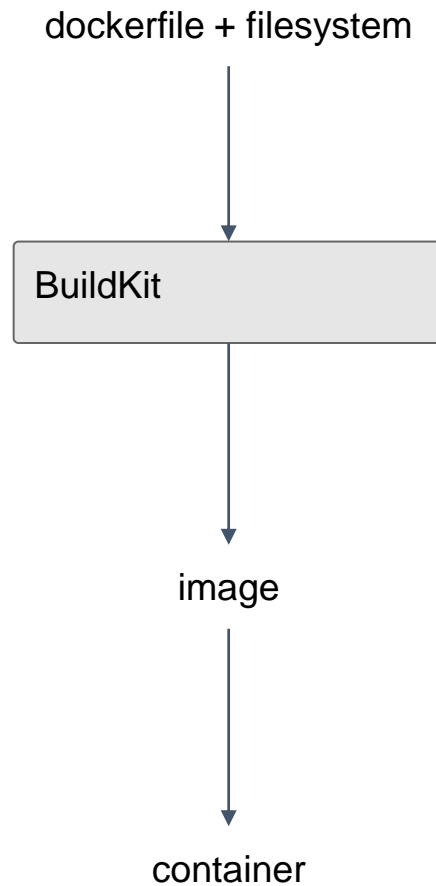
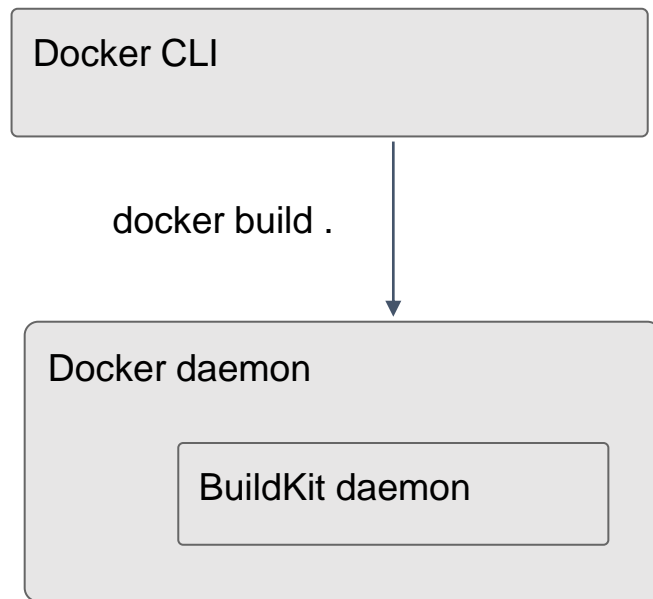
- Контейнеры состоят из слоёв.
- Каждый слой, кроме последнего, находящегося поверх всех остальных, предназначен только для чтения.
- Dockerfile сообщает системе Docker о том, какие слои и в каком порядке надо добавить в образ.
- Каждый слой — это файл, который описывает изменение состояния образа в сравнении с тем состоянием, в котором он пребывал после добавления предыдущего слоя.
- Базовый образ — это то, что является исходным слоем (или слоями) создаваемого образа. Базовый образ ещё называют родительским образом.

# Dockerfile

- В файлах Dockerfile содержатся инструкции по созданию образа (набираются заглавными буквами). С них начинаются строки этого файла.
- После инструкций идут их аргументы.
- Инструкции, при сборке образа, обрабатываются сверху вниз.
- Слои в итоговом образе создают только инструкции FROM, RUN, COPY, и ADD.

```
FROM python:3.6-alpine3.8      # Скачиваем образ python-  
alpine  
  
# Копируем test.py с компьютера в директорию /myapp образа  
COPY test.py /myapp/  
  
# Делаем /app рабочей директорией для след. команды  
WORKDIR /myapp  
  
# Ассоциируем запуск контейнера с запуском test.py  
ENTRYPOINT ["python3", "/myapp/test.py"]
```

# BuildKit

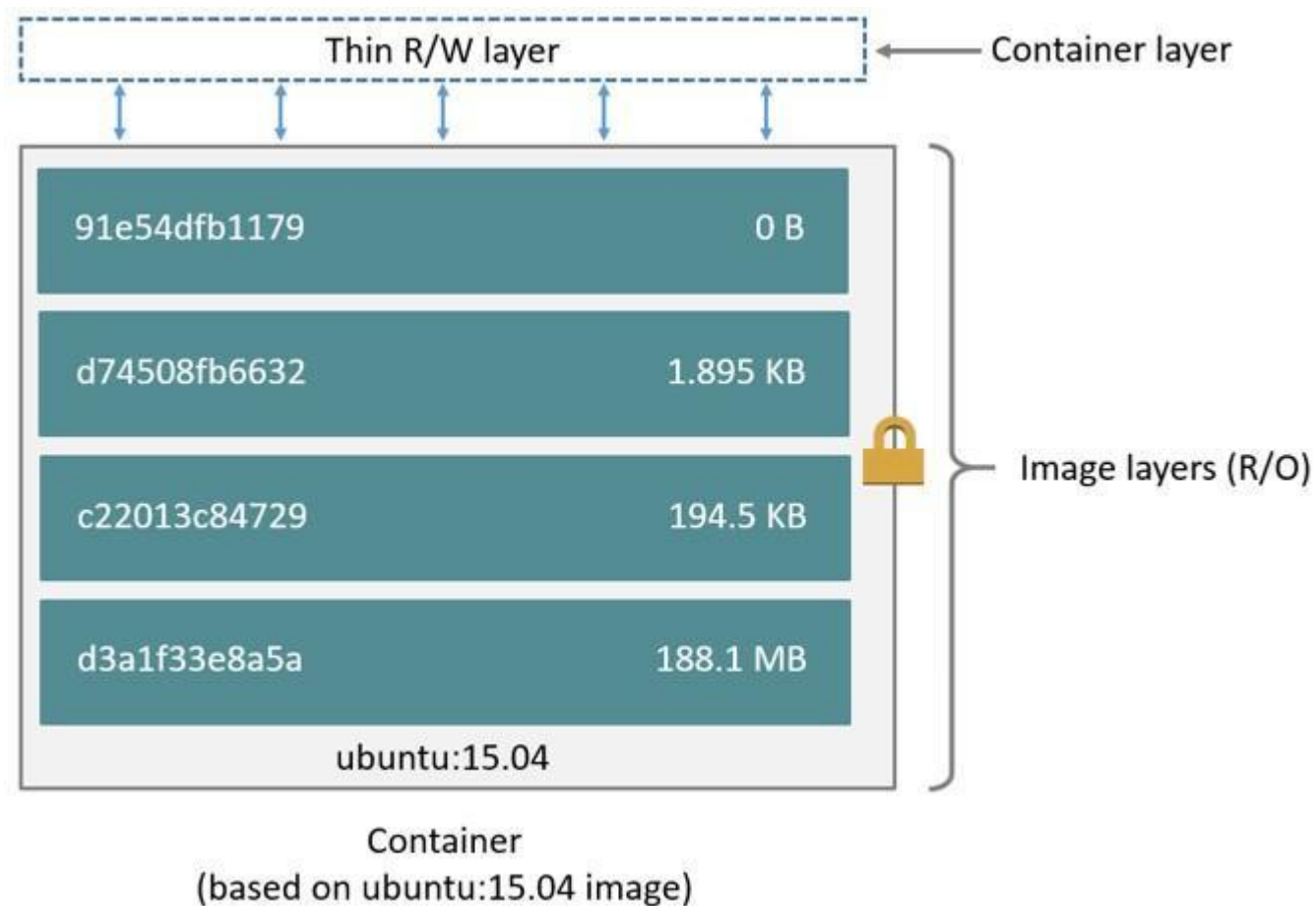


# Основные инструкции

FROM	Задаёт базовый (родительский) образ.
COPY	Копирует в контейнер файлы и папки.
ADD	Копирует файлы и папки в контейнер, может распаковывать локальные .tar-файлы.
WORKDIR	Задаёт рабочую директорию для следующей инструкции.
ENV	Устанавливает постоянные переменные среды.
ARG	Задаёт переменные для передачи Docker во время сборки образа.
LABEL	Описывает метаданные. Например — сведения о том, кто создал и поддерживает образ.
RUN	Выполняет команду и создаёт слой образа. Используется для установки в контейнер пакетов.
CMD	Описывает команду с аргументами, которую нужно выполнить когда контейнер будет запущен. Аргументы могут быть переопределены при запуске контейнера. В файле может присутствовать лишь одна инструкция CMD.
ENTRYPOINT	Предоставляет команду с аргументами для вызова во время выполнения контейнера. Аргументы не переопределяются.
EXPOSE	Указывает на необходимость открыть порт.
VOLUME	Создаёт точку монтирования для работы с постоянным хранилищем.

# Основные инструкции

- При создании контейнера слой, в который можно вносить изменения, добавляется поверх всех остальных слоёв. Данные, находящиеся в остальных слоях, можно только читать.
- Docker использует стратегию копирования при записи. Если слой в образе существует на предыдущем уровне и какому-то слою нужно произвести чтение данных из него, Docker использует существующий файл.
- Когда образ выполняется, если слой нужно модифицировать средствами контейнера, то соответствующий файл копируется в самый верхний, изменяемый слой.



# Создаем образ

```
FROM python:3.7.2-alpine3.8
```

```
LABEL maintainer="boss@example.com"
```

```
ENV ADMIN="boss"
```

```
RUN apk update && apk upgrade && apk add bash
```

```
COPY . ./app
```

```
ADD https://raw.githubusercontent.com/discdiver/pachy-vid/master/sample_vids/vid1.mp4 \  
/my_app_directory
```

```
RUN ["mkdir", "/a_directory"]
```

```
CMD ["python", "./my_script.py"]
```



# Создаем образ

```
FROM python:3.7.2-alpine3.8
LABEL maintainer="boss@example.com"
# Устанавливаем зависимости
RUN apk add --update git
# Задаём текущую рабочую директорию
WORKDIR /usr/src/my_app_directory
# Копируем код из локального контекста в рабочую директорию образа
COPY . .
# Задаём значение по умолчанию для переменной
ARG my_var=my_default_value
# Настраиваем команду, которая должна быть запущена в контейнере во время его выполнения
ENTRYPOINT ["python", "./app/my_script.py", "my_var"]
# Открываем порты
EXPOSE 8000
# Создаём том для хранения данных
VOLUME /my_volume
```

# Как уменьшить размер образа? Как ускорить сборку?

- Использование кэша способно ускорить сборку образов, но есть одна проблема:

Например, если в Dockerfile обнаруживается инструкция `RUN pip install -r requirements.txt`, то Docker выполняет поиск такой же инструкции в своём локальном кэше промежуточных образов. **При этом содержимое старой и новой версий файла requirements.txt не сравнивается.**

- В отличие от других инструкций Docker, при выполнении инструкций `ADD` и `COPY` от Docker требуется проверка содержимого файла или файлов для определения того, можно ли, при формировании образа, воспользоваться кэшем.

# Как уменьшить размер образа? Как ускорить сборку?

- Кэширование можно отключить, передав ключ `--no-cache=True` команде `docker build`.
- Если вы собираетесь вносить изменения в инструкции `Dockerfile`, тогда каждый слой, созданный инструкциями, идущими после изменённых, будет достаточно часто собираться повторно, без использования кэша. Для того чтобы воспользоваться преимуществами кэширования, помещайте инструкции, вероятность изменения которых высока, как можно ближе к концу `Dockerfile`.
- Объединяйте команды `RUN apt-get update` и `apt-get install` в цепочки для того, чтобы исключить проблемы, связанные с неправильным использованием кэша.
- Если вы используете менеджеры пакетов, наподобие `pip`, с файлом `requirements.txt`, тогда придерживайтесь нижеприведённой схемы работы для того, чтобы исключить использование устаревших промежуточных образов из кэша, содержащих набор пакетов, перечисленных в старой версии файла `requirements.txt`:

```
COPY requirements.txt /tmp/  
RUN pip install -r /tmp/requirements.txt  
COPY . /tmp/
```

# Как уменьшить размер образа? Как ускорить сборку?

Тщательный подбор  
базового образа

Файл `.dockerignore`

Многоступенчатая  
сборка образов

Исследование  
размеров образов

# Как уменьшить размер образа? Как ускорить сборку?

## Многоступенчатая сборка образов

```
FROM golang:1.7.3 AS build
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .
FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=build /go/src/github.com/alexellis/href-counter/app .
CMD ["/app"]
```

# Как уменьшить размер образа? Как ускорить сборку?

## Файл .dockerignore

- Это позволяет исключать из состава образа файлы, содержащие секретные сведения наподобие логинов и паролей.
- Это позволяет уменьшить размер образа. Чем меньше в образе файлов — тем меньше будет его размер и тем быстрее с ним можно будет работать.
- Это даёт возможность уменьшить число поводов для признания недействительным кэша при сборке похожих образов. Например, если при повторной сборке образа меняются некие служебные файлы проекта, наподобие файлов с журналами, из-за чего данные, хранящиеся в кэше, по сути, необоснованно признаются недействительными, это замедляет сборку образов.

# Как уменьшить размер образа? Как ускорить сборку?

## Исследование размеров образов

- Для того чтобы выяснить примерный размер выполняющегося контейнера, можно использовать команду вида `docker container ls -s`.
- Команда `docker image ls` выводит размеры образов.
- Узнать размеры промежуточных образов, из которых собран некий образ, можно с помощью команды `docker image history my_image:my_tag`.
- Команда `docker image inspect my_image:tag` позволяет узнать подробные сведения об образе, в том числе — размер каждого его слоя. Слои немного отличаются от промежуточных образов, из которых состоит готовый образ, но, в большинстве случаев их можно рассматривать как одинаковые сущности.
- Для того чтобы исследовать содержимое контейнеров можно установить пакет `dive`.

# Как уменьшить размер образа? Как ускорить сборку?

## Рекомендации по уменьшению размеров образов и ускорению процесса их сборки

1. Используйте всегда, когда это возможно, официальные образы в качестве базовых образов. Официальные образы регулярно обновляются, они безопаснее неофициальных образов.
2. Для того чтобы собирать как можно более компактные образы, пользуйтесь базовыми образами, основанными на Alpine Linux.



# Как уменьшить размер образа? Как ускорить сборку?

## Рекомендации по уменьшению размеров образов и ускорению процесса их сборки

Если вы пользуетесь apt, комбинируйте в одной инструкции RUN команды apt-get update и apt-get install. Кроме того, объединяйте в одну инструкцию команды установки пакетов. Перечисляйте пакеты в алфавитном порядке на нескольких строках, разделяя список символами \. Например, это может выглядеть так:

```
RUN apt-get update && apt-get install -y \  
package-one \  
package-two \  
package-three && rm -rf /var/lib/apt/lists/*
```

Этот метод позволяет сократить число слоёв, которые должны быть добавлены в образ, и помогает поддерживать код файла в приличном виде.

1. Включайте конструкцию вида `&& rm -rf /var/lib/apt/lists/*` в конец инструкции RUN, используемой для установки пакетов. Это позволит очистить кэш apt и приведёт к тому, что он не будет сохраняться в слое, сформированном командой RUN. Подробности об этом можно почитать в документации.
2. Разумно пользуйтесь возможностями кэширования, размещая в Dockerfile команды, вероятность изменения которых высока, ближе к концу файла.
3. Пользуйтесь файлом `.dockerignore`.
4. Взгляните на `dive` — отличный инструмент для исследования образов Docker, который помогает в деле уменьшения их размеров.
5. Не устанавливайте в образы пакеты, без которых можно обойтись.



СПАСИБО ЗА ВНИМАНИЕ!