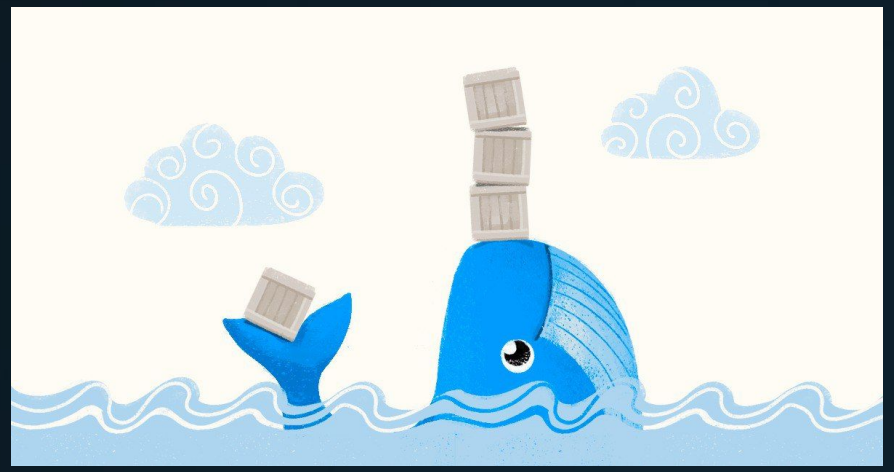


Docker

Information Technology

Docker

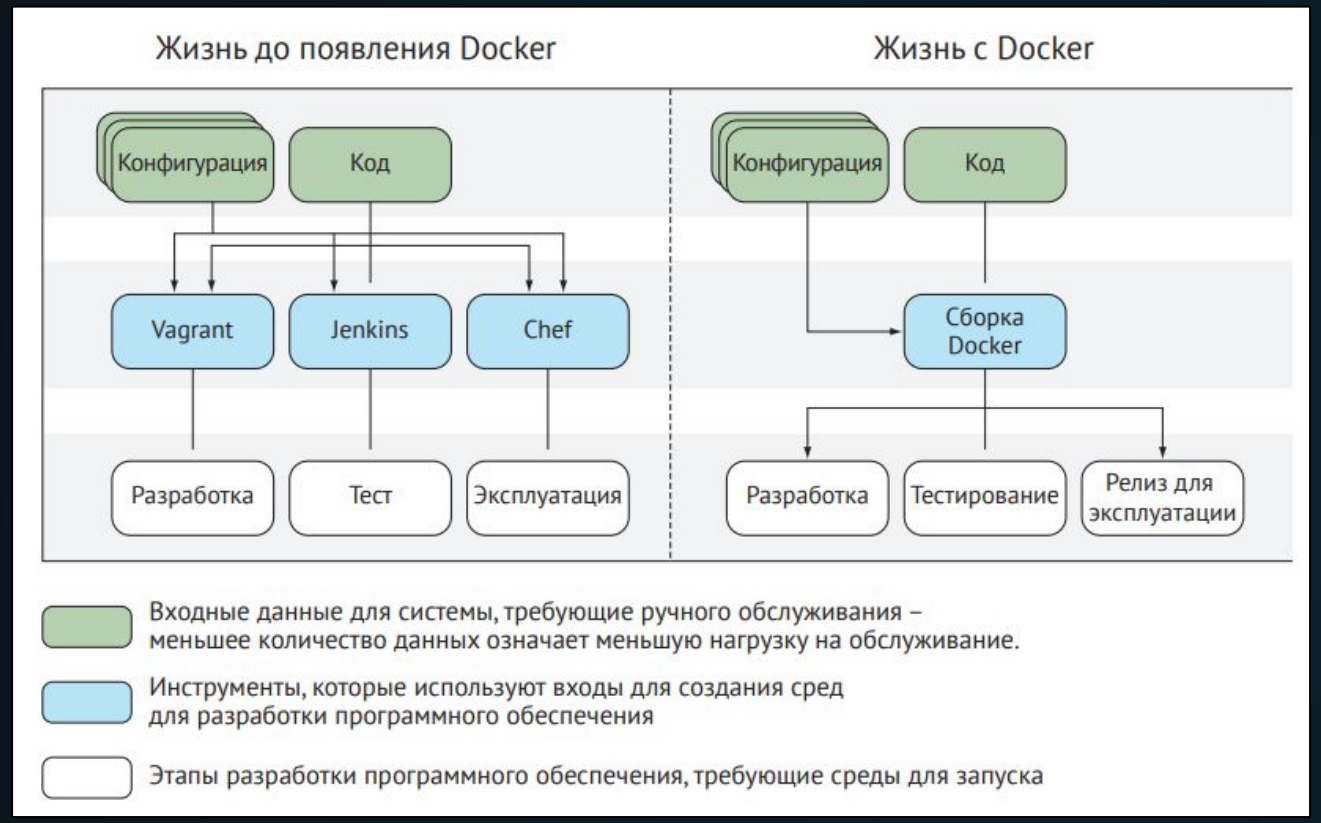
Docker – это платформа, которая позволяет «создавать, поставлять и запускать любое приложение повсюду». За невероятно короткое время она прошла большой путь и теперь считается стандартным способом решения одного из самых дорогостоящих аспектов программного обеспечения – развертывания.

Развертывание – это все действия, которые делают программную систему готовой к использованию

Как Docker облегчил бремя обслуживания инструментов

До появления Docker в конвейере разработки обычно использовались комбинации различных технологий для управления движением программного обеспечения, такие как виртуальные машины, инструменты управления конфигурацией, системы управления пакетами и комплексные сети библиотечных зависимостей. Все эти инструменты должны были управляться и поддерживаться специализированными инженерами, и у большинства из них были свои собственные уникальные способы настройки.

Docker изменил все это, позволив различным инженерам, вовлеченным в этот процесс, эффективно говорить на одном языке, облегчая совместную работу. Все проходит через общий конвейер к одному выходу, который можно использовать для любой цели, – нет необходимости продолжать поддерживать запутанный массив конфигураций инструментов, как показано на рисунке.

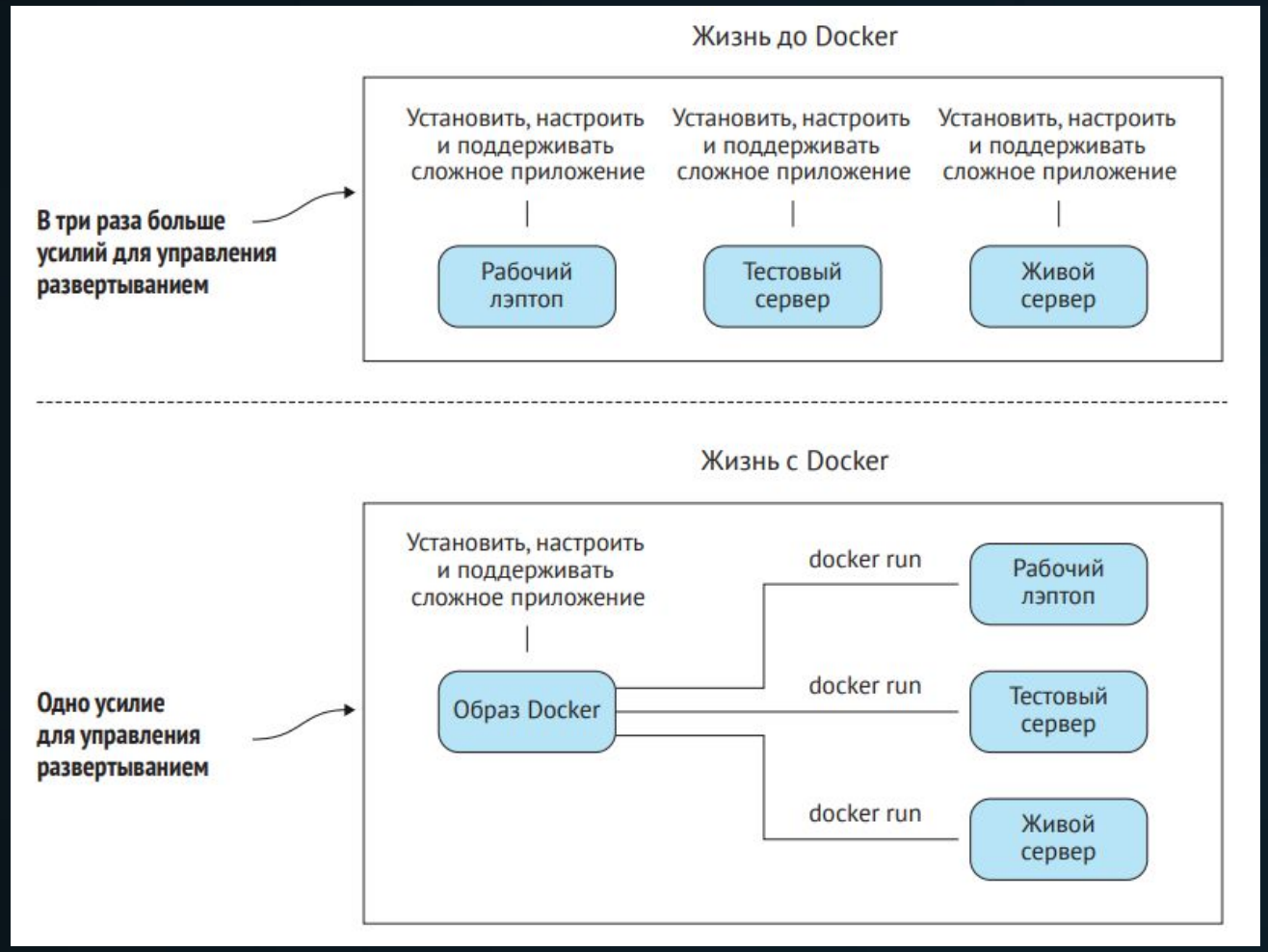


Что такое Docker?

Что такое Docker?

Доставка программного обеспечения до и после Docker

До появления Docker развертывание программного обеспечения в различных средах требовало значительных усилий.



Чем хорош Docker?

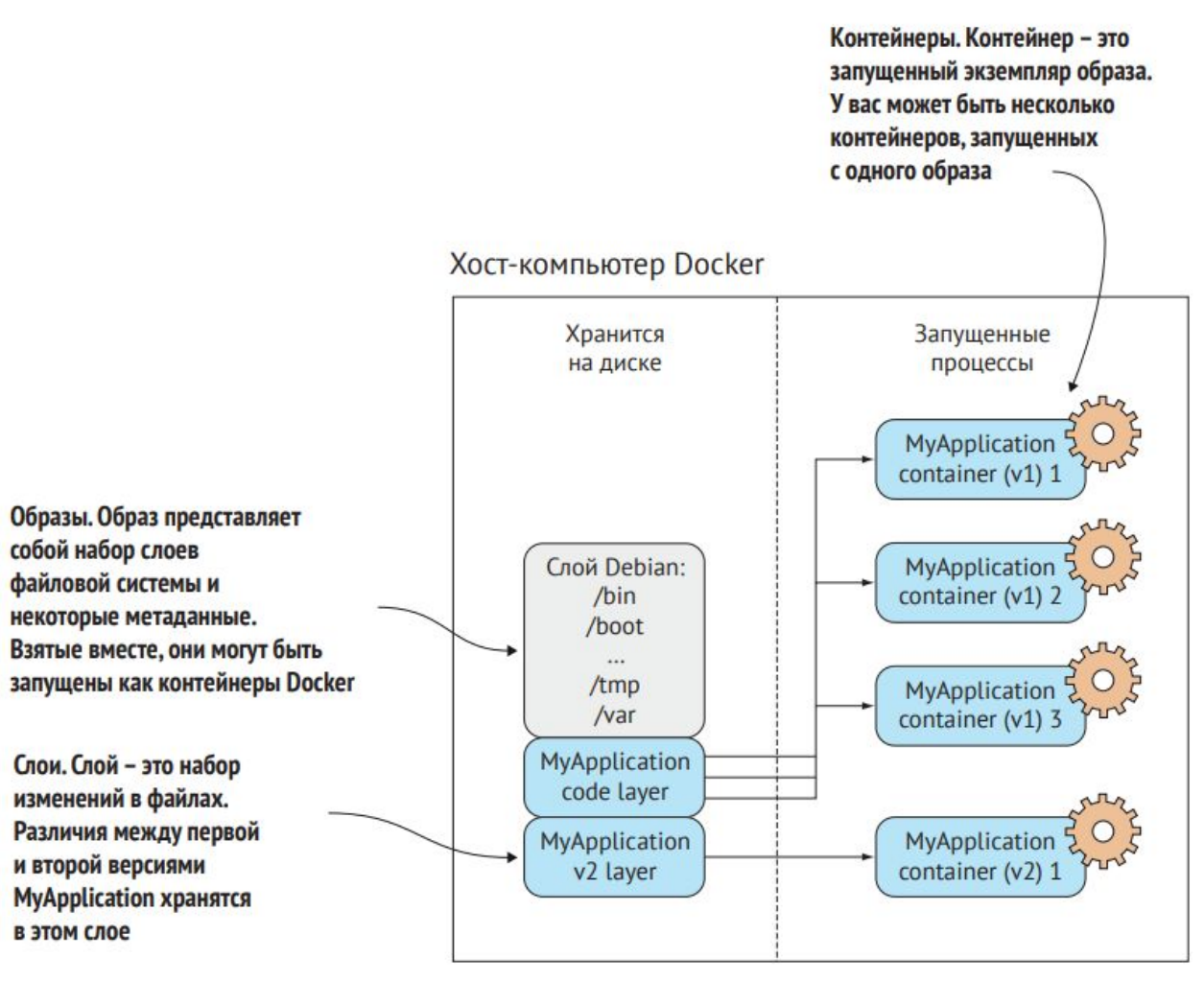
- Замена виртуальных машин;
- Прототипирование программного обеспечения (быстрая «черновая» реализация базовой функциональности будущего продукта/изделия, для анализа работы системы в целом);
- Упаковка программного обеспечения;
- Возможность для архитектуры микросервисов;
- Моделирование сетей;

Чем хорош Docker?

- Возможность производительности полного стека в автономном режиме (сфера деятельности разработчика широкого профиля, который умеет работать с фронтендом (клиентская сторона, пользовательский интерфейс) и бэкендом);
- Сокращение неизбежных расходов на отладку.
- Документирование зависимостей программного обеспечения и точки взаимодействия;
- Возможность непрерывной доставки.

Базовые концепции Docker

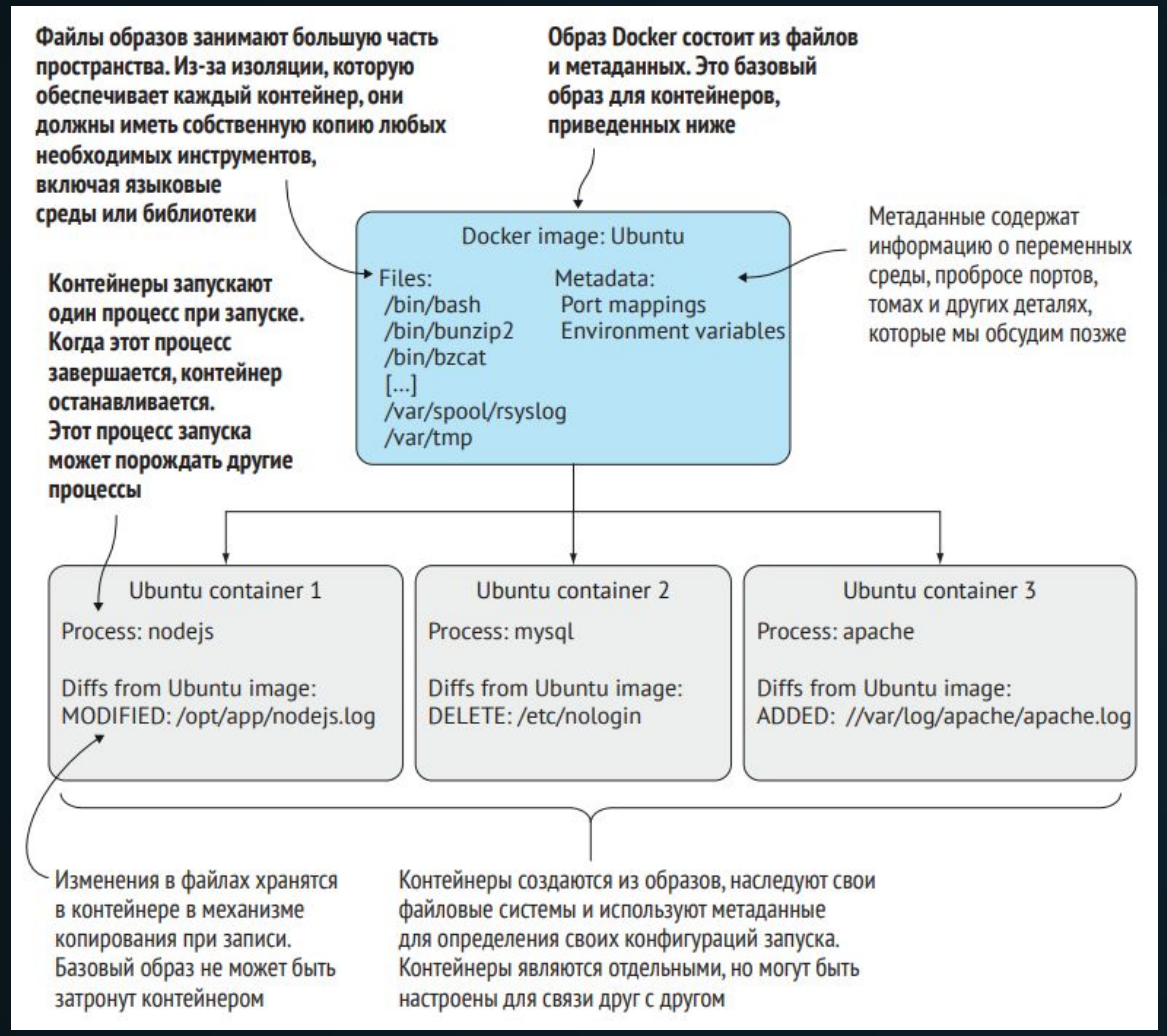
Прежде чем запускать команды Docker, лучше всего разобраться с понятиями образов, контейнеров и слоев. Говоря кратко, контейнеры запускают системы, определенные образами. Эти образы состоят из одного или нескольких слоев (или наборов различий) плюс некоторые метаданные Docker.



Образы и контейнеры Docker

Один из способов взглянуть на образы и контейнеры – это рассматривать их как программы и процессы.

Если вы знакомы с принципами объектно ориентированного программирования, еще один способ взглянуть на образы и контейнеры – это рассматривать образы как классы, а контейнеры – как объекты.

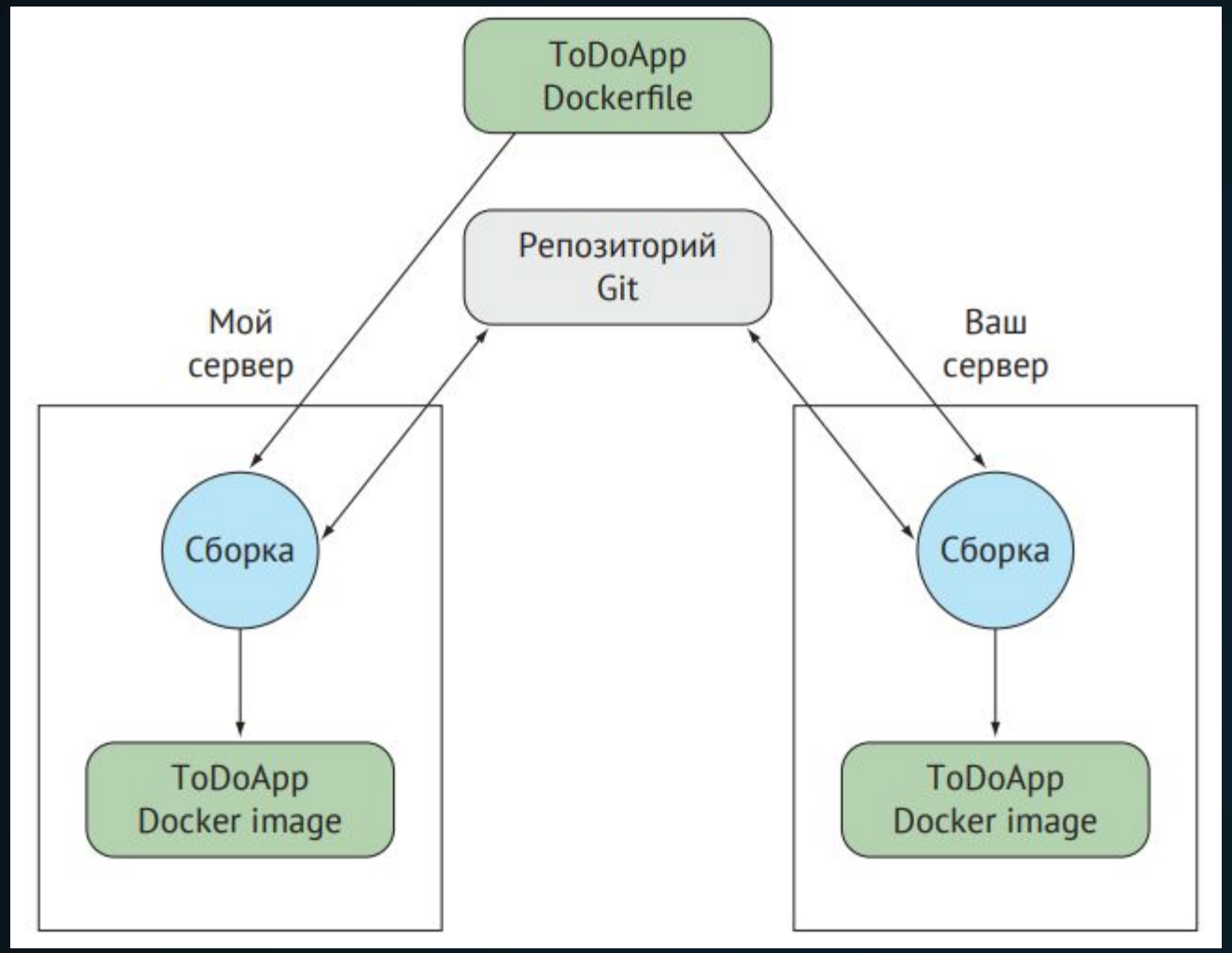


Создание приложения Docker

- ✔ Как создать образ Docker с помощью Dockerfile;
- ✔ Как присвоить тег образу Docker для удобства пользования;
- ✔ Как запустить свой новый образ Docker.

Приложение в формате to-do – это приложение, которое помогает вам отслеживать то, что вы хотите сделать.

Приложение, создаваемое нами, будет хранить и отображать короткие строки информации, которые можно пометить как выполненные, представленные в простом веб-интерфейсе.



Способы создания нового образа Docker

Метод Описание	Метод Описание
Команды Docker/ «От руки»	Запустите контейнер с помощью **docker run** и введите команды для создания образа в командной строке. Создайте новый образ с помощью **docker commit**
Dockerfile	Выполните сборку из известного базового образа и укажите ее с помощью ограниченного набора простых команд.
Dockerfile и инструмент управления конфигурацией	То же самое, что и Dockerfile, но вы передаете контроль над сборкой более сложному инструменту управления конфигурацией
Стереть образ и импортировать набор файлов	Из пустого образа импортируйте файл TAR с необходимыми файлами

Последний вариант строится из нулевого образа путем наложения набора файлов, необходимых для запуска образа. Это полезно, если вы хотите импортировать набор автономных файлов, созданных в другом месте, но этот метод редко встречается при массовом использовании.

Пишем Dockerfile

```
FROM node
LABEL maintainer ian.miell@gmail.com
RUN git clone -q https://github.com/docker-in-practice/todo.git
WORKDIR todo
RUN npm install > /dev/null
EXPOSE 8000
CMD ["npm","start"]
```

Определяет базовый образ

Объявляет автора

Клонирует код приложения

Перемещает в новый клонированный каталог

Запускает команду установки менеджера пакетов (npm)

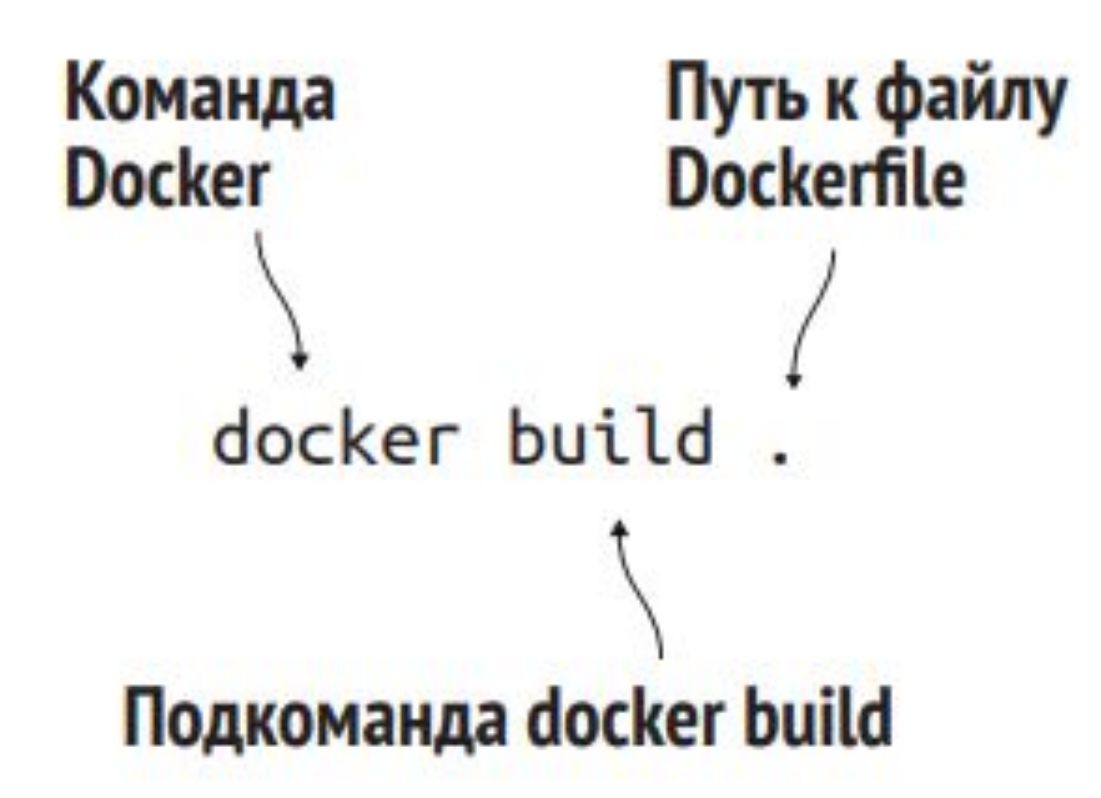
Указывает, какая команда будет запущена при запуске

Указывает, что контейнеры из собранного образа должны прослушивать этот порт

Dockerfile – это текстовый файл, содержащий серию команд.

Собираем образ Docker

Вы определили шаги сборки своего файла Dockerfile. Теперь вы собираетесь создать из него образ Docker.



Собираем образ Docker

Теперь у вас есть образ Docker со своим идентификатором («66c76cea05bb» в предыдущем примере, но ваш идентификатор будет другим). Возможно, к нему неудобно обращаться, поэтому вы можете присвоить ему тег для удобства.

```
Sending build context to Docker daemon  2.048kB
Step 1/7 : FROM node
 ---> 2ca756a6578b
Step 2/7 : LABEL maintainer ian.miell@gmail.com
 ---> Running in bf73f87c88d6
 ---> 5383857304fc
Removing intermediate container bf73f87c88d6
Step 3/7 : RUN git clone -q https://github.com/docker-in-practice/todo.git
 ---> Running in 761baf524cc1
 ---> 4350cb1c977c
Removing intermediate container 761baf524cc1
Step 4/7 : WORKDIR todo
 ---> a1b24710f458
Removing intermediate container 0f8cd22fbe83
Step 5/7 : RUN npm install > /dev/null
 ---> Running in 92a8f9ba530a
npm info it worked if it ends with ok
[...]
npm info ok
 ---> 6ee4d7bba544
Removing intermediate container 92a8f9ba530a
Step 6/7 : EXPOSE 8000
 ---> Running in 8e33c1ded161
 ---> 3ea44544f13c
Removing intermediate container 8e33c1ded161
Step 7/7 : CMD npm start
 ---> Running in ccc076ee38fe
 ---> 66c76cea05bb
Removing intermediate container ccc076ee38fe
Successfully built 66c76cea05bb
```

Каждая команда приводит к созданию нового образа с выводом его идентификатора

Docker загружает файлы и каталоги по пути, предоставленному команде docker build

Каждый шаг сборки последовательно нумеруется, начиная с 1, и выводится командой

Для экономии места каждый промежуточный контейнер удаляется, перед тем как продолжить

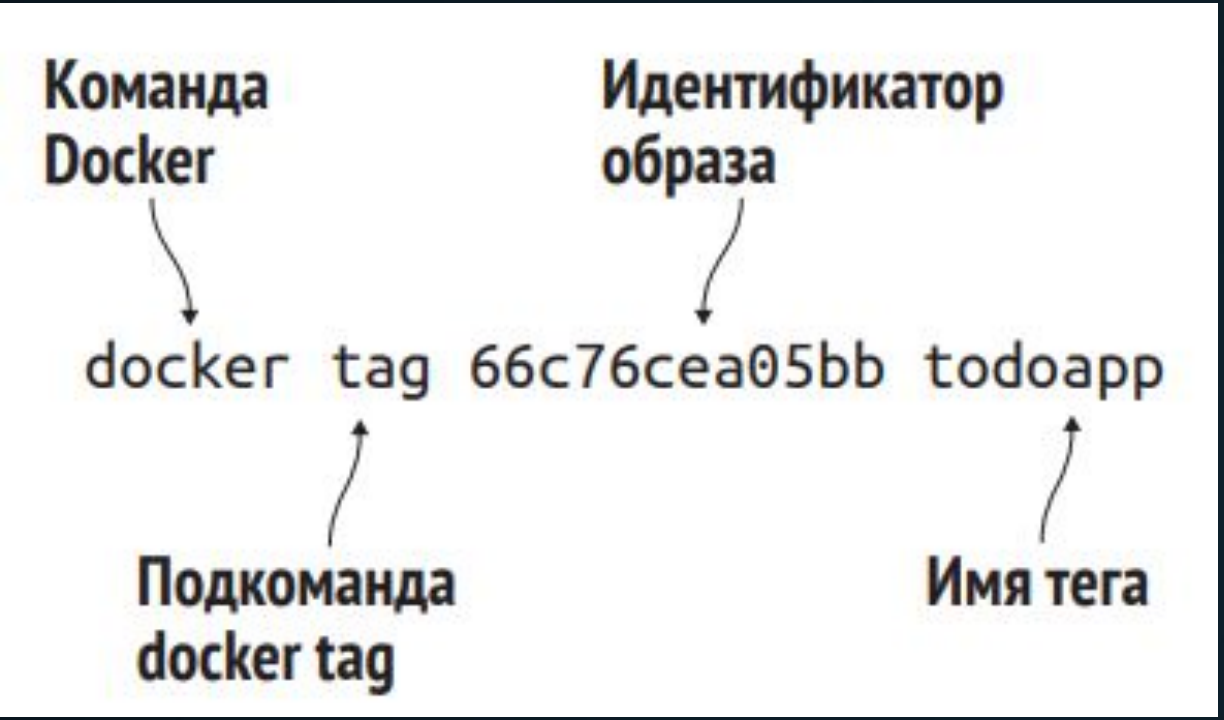
Отладка сборки выводится здесь (и редактируется из этого списка)

Окончательный идентификатор образа для этой сборки, готовый к присвоению тега

Собираем образ Docker

Введите предыдущую команду, заменив 66c76cea05bb сгенерированным для вас идентификатором образа.

Теперь вы можете собрать свою собственную копию образа Docker из файла Dockerfile, воспроизводя среду, определенную кем-то другим!



Запускаем контейнер Docker

```
$ docker run -i -t -p 8000:8000 --name example1 todoapp
npm install
npm info it worked if it ends with ok
npm info using npm@2.14.4
npm info using node@v4.1.1
npm info prestart todomvc-swarm@0.0.1

> todomvc-swarm@0.0.1 prestart /todo
 > make all

npm install
```

Подкоманда docker run запускает контейнер, -p перенаправляет порт контейнера 8000 в порт 8000 на хост-компьютере, --name присваивает контейнеру уникальное имя, а последний аргумент – это образ

Вывод процесса запуска контейнера отправляется на терминал

```
npm info it worked if it ends with ok
npm info using npm@2.14.4
npm info using node@v4.1.1
npm WARN package.json todomvc-swarm@0.0.1 No repository field.
npm WARN package.json todomvc-swarm@0.0.1 license should be a valid SPDX
➥ license expression
npm info preinstall todomvc-swarm@0.0.1
npm info package.json statics@0.1.0 license should be a valid SPDX license
➥ expression
npm info package.json react-tools@0.11.2 No license field.
npm info package.json react@0.11.2 No license field.
npm info package.json node-
     jsx@0.11.0 license should be a valid SPDX license expression
npm info package.json ws@0.4.32 No license field.
```

```
npm info build /todo
npm info linkStuff todomvc-swarm@0.0.1
npm info install todomvc-swarm@0.0.1
npm info postinstall todomvc-swarm@0.0.1
npm info prepublish todomvc-swarm@0.0.1
npm info ok
if [ ! -e dist/ ]; then mkdir dist; fi
cp node_modules/react/dist/react.min.js dist/react.min.js

LocalTodoApp.js:9: // TODO: default english version
LocalTodoApp.js:84: fwdList = this.host.get('/TodoList#'+listId);
 // TODO fn+id sig
TodoApp.js:117: // TODO scroll into view
TodoApp.js:176: if (i>=list.length()) { i=list.length()-1; } // TODO
➥ .length
local.html:30: <!-- TODO 2-split, 3-split -->
model/TodoList.js:29: // TODO one op - repeated spec? long spec?
view/Footer.jsx:61: // TODO: show the entry's metadata
view/Footer.jsx:80: todoList.addObject(new TodoItem()); // TODO
➥ create default
view/Header.jsx:25: // TODO list some meaningful header (apart from the
➥ id)

npm info start todomvc-swarm@0.0.1

> todomvc-swarm@0.0.1 start /todo
> node TodoAppServer.js
```

Запускаем контейнер Docker

Как видно, тот факт, что Docker «содержит» вашу среду, означает, что вы способны рассматривать ее как сущность, над которой можно предсказуемо выполнять действия. Это дает Docker широкие возможности – влиять на жизненный цикл программного обеспечения от разработки до эксплуатации и обслуживания.

```
Swarm server started port 8000
^Cshutting down http-server...
closing swarm host...
swarm host closed
npm info lifecycle todomvc-swarm@0.0.1~poststart: todomvc-swarm@0.0.1
npm info ok
$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
b9db5ada0461 todoapp "npm start" 2 minutes ago Exited (0) 2 minutes ago
➥ example1
$ docker start example1
example1
$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS
➥ PORTS NAMES
b9db5ada0461 todoapp "npm start" 8 minutes ago Up 10 seconds
➥ 0.0.0.0:8000->8000/tcp example1
$ docker diff example1
C /root
C /root/.npm
C /root/.npm/_locks
C /root/.npm/anonymous-cli-metrics.json
C /todo
 A /todo/.swarm
 A /todo/.swarm/_log
A /todo/dist
A /todo/dist/LocalTodoApp.app.js
A /todo/dist/TodoApp.app.js
A /todo/dist/react.min.js
C /todo/node_modules
```

Нажмите здесь сочетание клавиш Ctrl-C, чтобы завершить процесс и контейнер

Выполните эту команду, чтобы увидеть контейнеры, которые были запущены и удалены, а также идентификатор и состояние (например, процесс)

Перезапустите контейнер, на этот раз в фоновом режиме

Снова выполняем команду docker ps, чтобы увидеть изменившийся статус

Подкоманда docker diff показывает, какие файлы были затронуты с момента создания экземпляра образа как контейнера

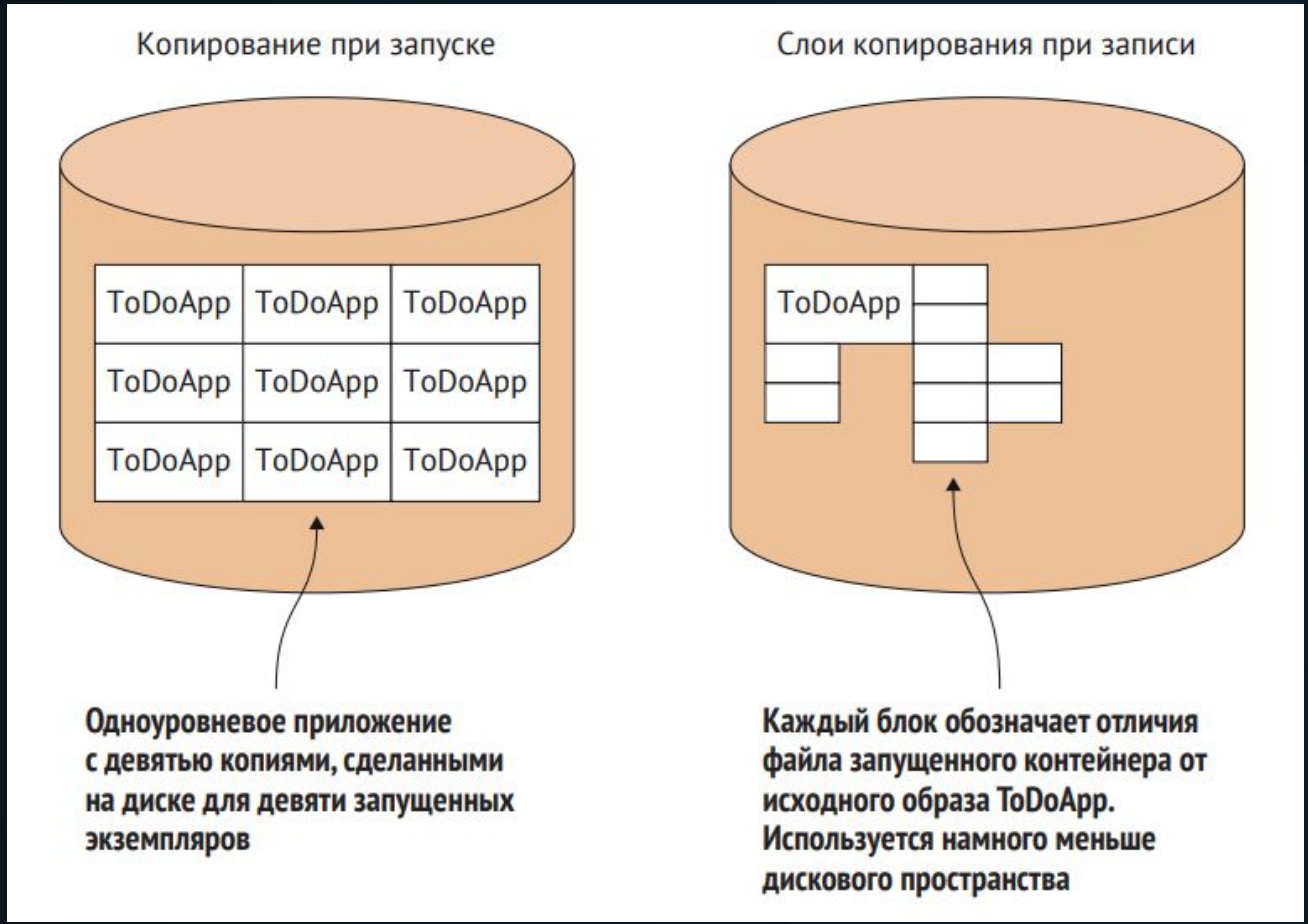
Каталог /todo был изменен (C)

Добавлен каталог /todo/.swarm (A)

Слои Docker

Слои Docker помогают справиться с большой проблемой, которая возникает, когда вы используете контейнеры в широком масштабе. Представьте себе, что произойдет, если вы запустите сотни или даже тысячи приложений, и каждому из них потребуется копия файлов для хранения в каком-либо месте.

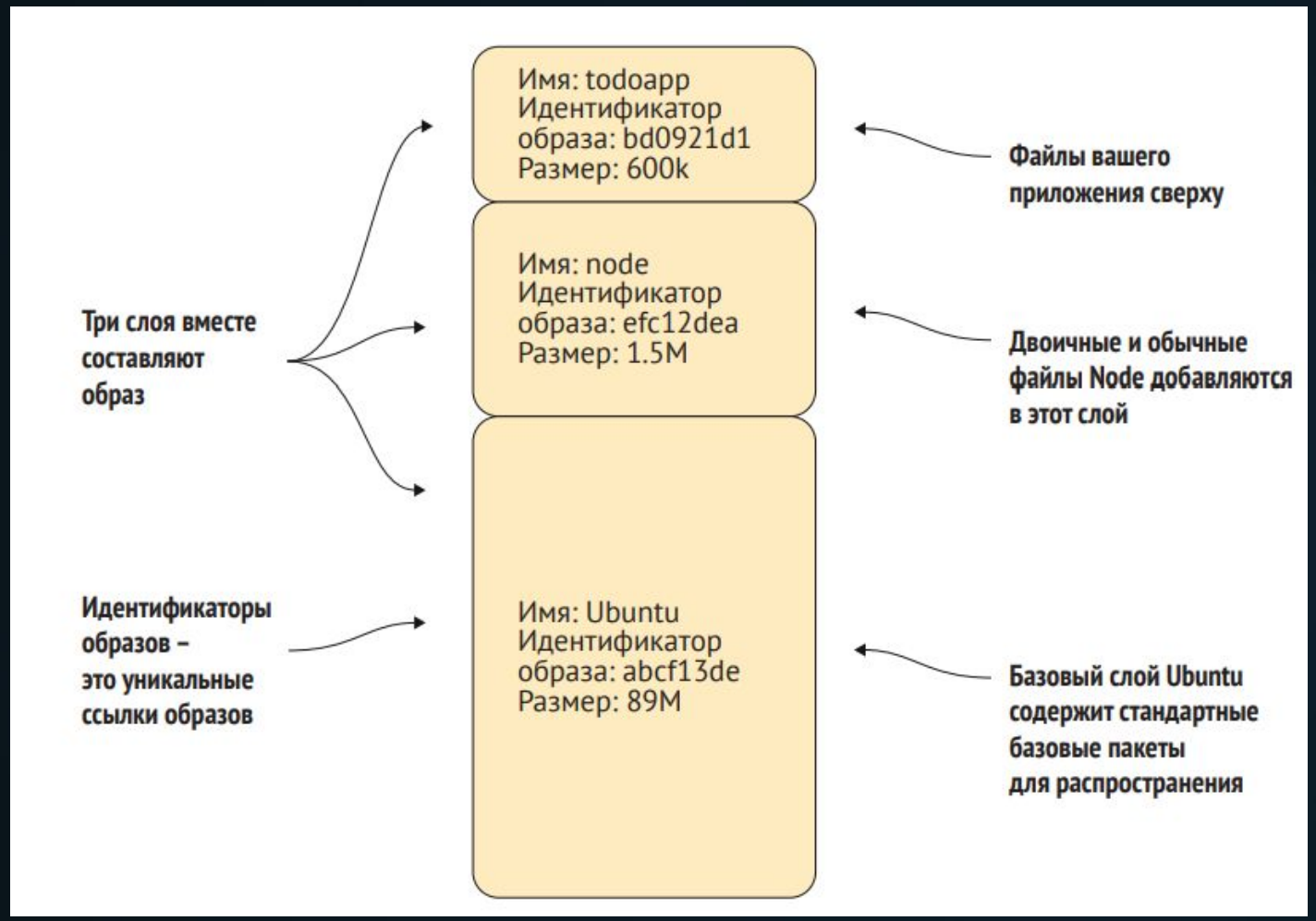
Как вы можете себе представить, дисковое пространство закончится довольно быстро! По умолчанию Docker внутренне использует механизм копирования при записи, чтобы уменьшить объем требуемого дискового пространства.



Концепция слоев в файловой системе ToDoApp в Docker

Слои статичны, поэтому, если вам нужно что-то изменить в более высоком слое, можно просто выполнить сборку поверх образа, который вы хотите взять в качестве ссылки. В своем приложении вы создали общедоступный node-образ и многоуровневые изменения сверху.

Все три слоя могут совместно использоваться несколькими запущенными контейнерами, так же как общая библиотека может совместно использоваться в памяти несколькими запущенными процессами. Это жизненно важная функция для операций, позволяющая запускать многочисленные контейнеры на основе разных образов на хост-компьютерах, не испытывая нехватки дискового пространства.



Резюме

- Вы можете создавать и запускать приложение Docker из файла Dockerfile, используя команды docker build и docker run.
- Образ Docker – это шаблон для работающего контейнера.
- Изменения в запущенных контейнерах можно сохранять и тегировать (классификация или разметка данных по какому-то признаку) как новые образы.
- Образы создаются из многоуровневой файловой системы, что уменьшает пространство, используемое образами Docker на вашем хосте.

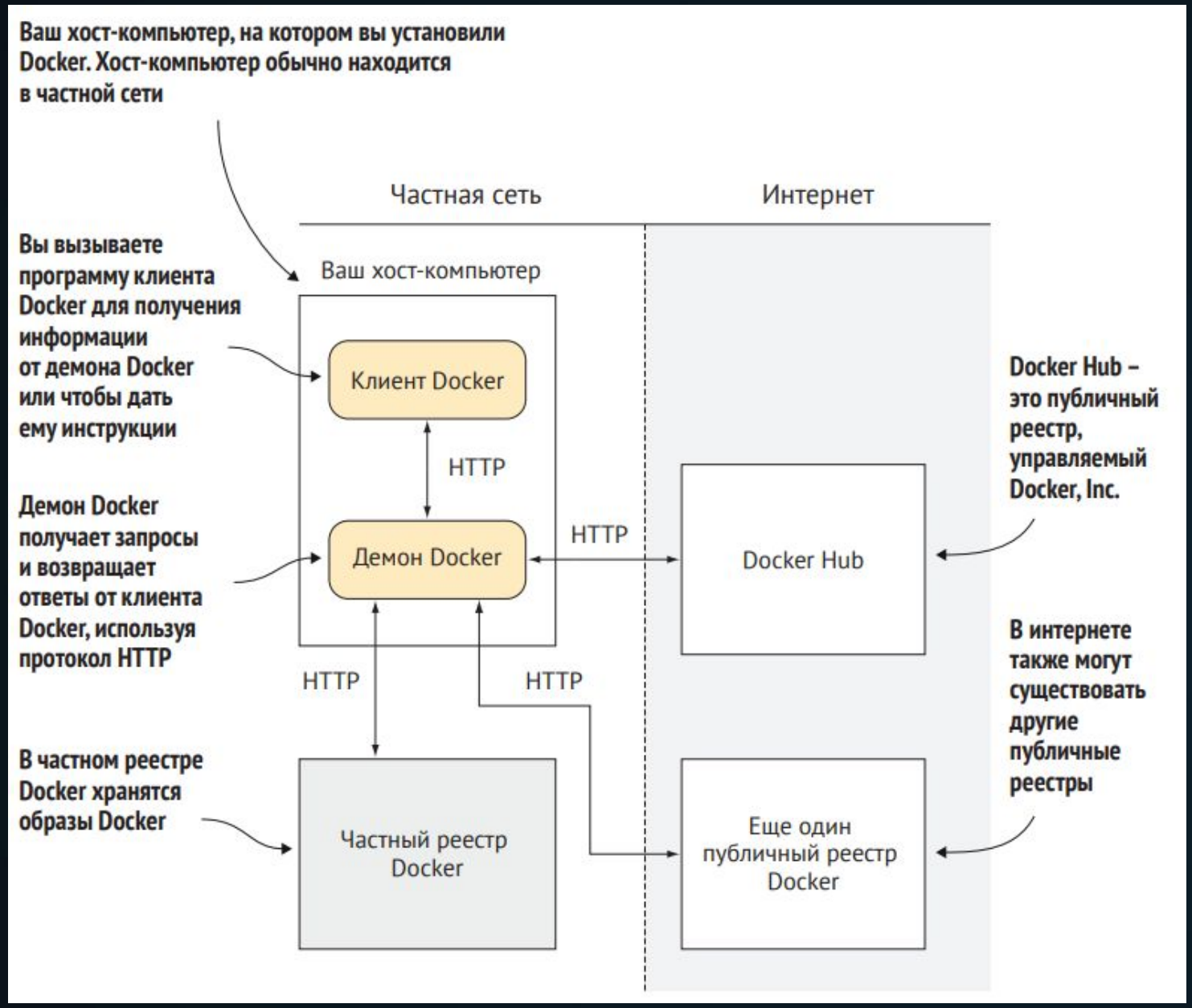
Архитектура Docker

Docker на вашем хост-компьютере разделен на две части: демон с прикладным программным интерфейсом RESTful и клиент, который общается с демоном.

Вы вызываете Docker клиент, чтобы получить информацию или дать инструкции демону. Демон – это сервер, который получает запросы и возвращает ответы от клиента по протоколу HTTP. В свою очередь, он будет отправлять запросы в другие службы для отправки и получения образов, также используя протокол HTTP. Сервер будет принимать запросы от клиента командной строки или любого, кто авторизован для подключения. Демон также отвечает за заботу о ваших образах и контейнерах.

Частный реестр Docker – это сервис, который хранит образы Docker. Их можно запросить у любого демона Docker, у которого есть соответствующий доступ. Этот реестр находится во внутренней сети и не является общедоступным, поэтому считается закрытым.

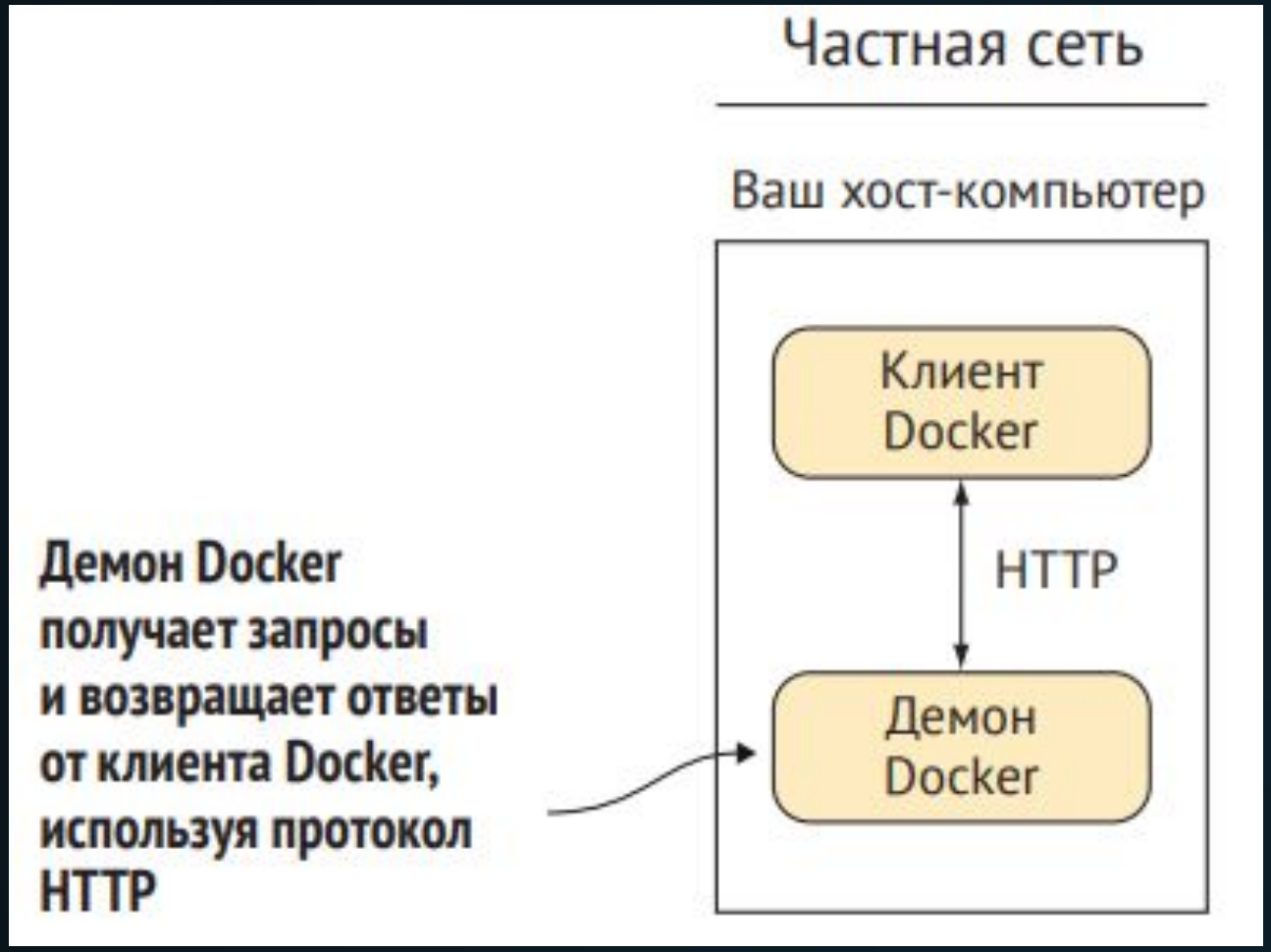
Docker Hub – это общедоступный реестр, управляемый Docker Inc. В интернете также могут существовать другие публичные реестры, и ваш демон Docker способен взаимодействовать с ними.



Демон Docker

Демон Docker – это центр ваших взаимодействий с Docker, и поэтому он является лучшим местом, где вы можете начать понимать все соответствующие элементы. Он контролирует доступ к Docker на вашем компьютере, управляет состоянием контейнеров и образов, а также взаимодействует с внешним миром.

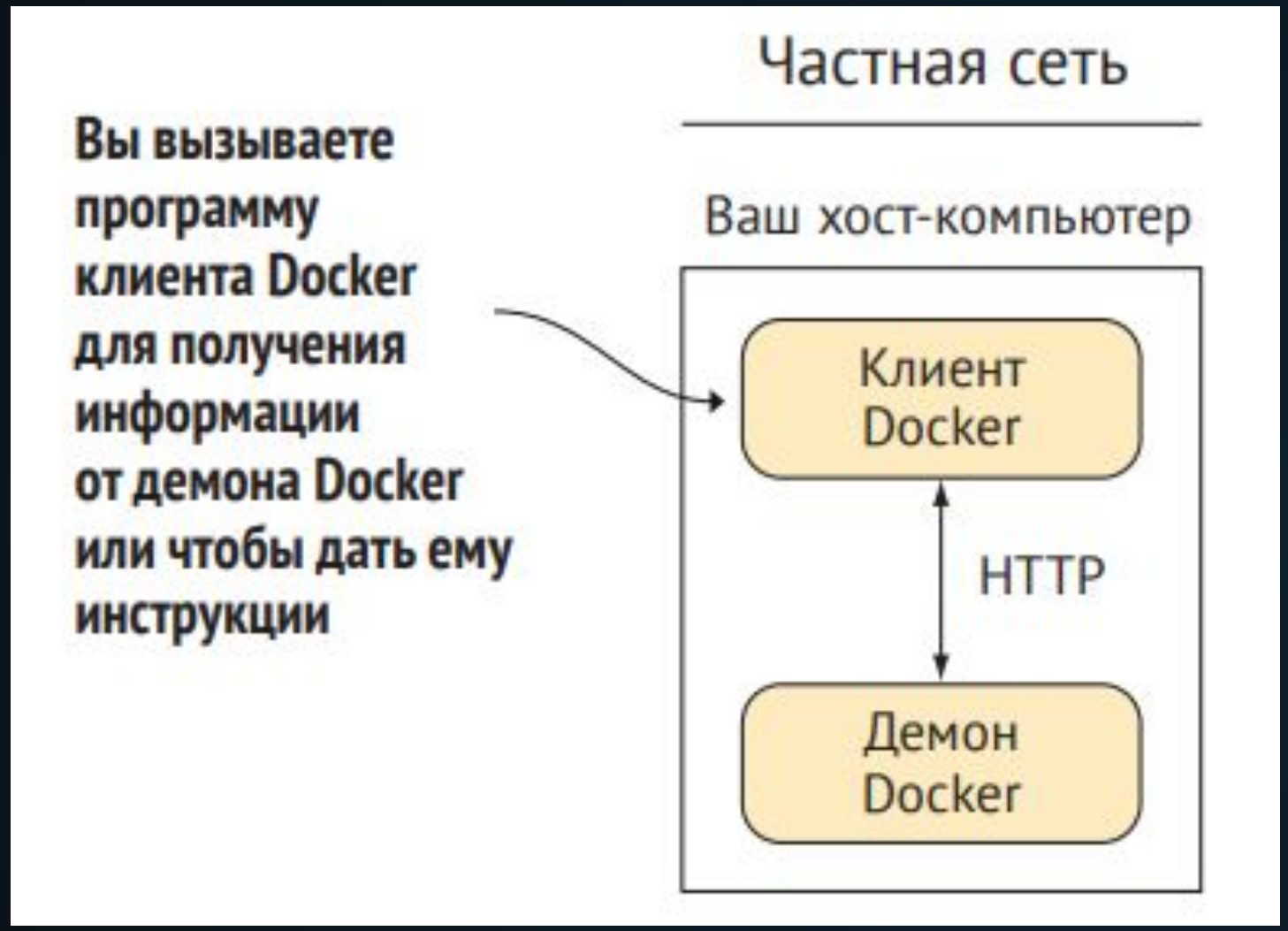
Демон – это процесс, который выполняется в фоновом режиме, а не под непосредственным контролем пользователя. Сервер – процесс, который принимает запросы от клиента и осуществляет действия, необходимые для выполнения запросов. Демоны часто также являются серверами, принимающими запросы от клиентов для выполнения действий для них. Команда docker – это клиент, а демон Docker выступает в качестве сервера, выполняющего обработку ваших контейнеров и образов Docker.



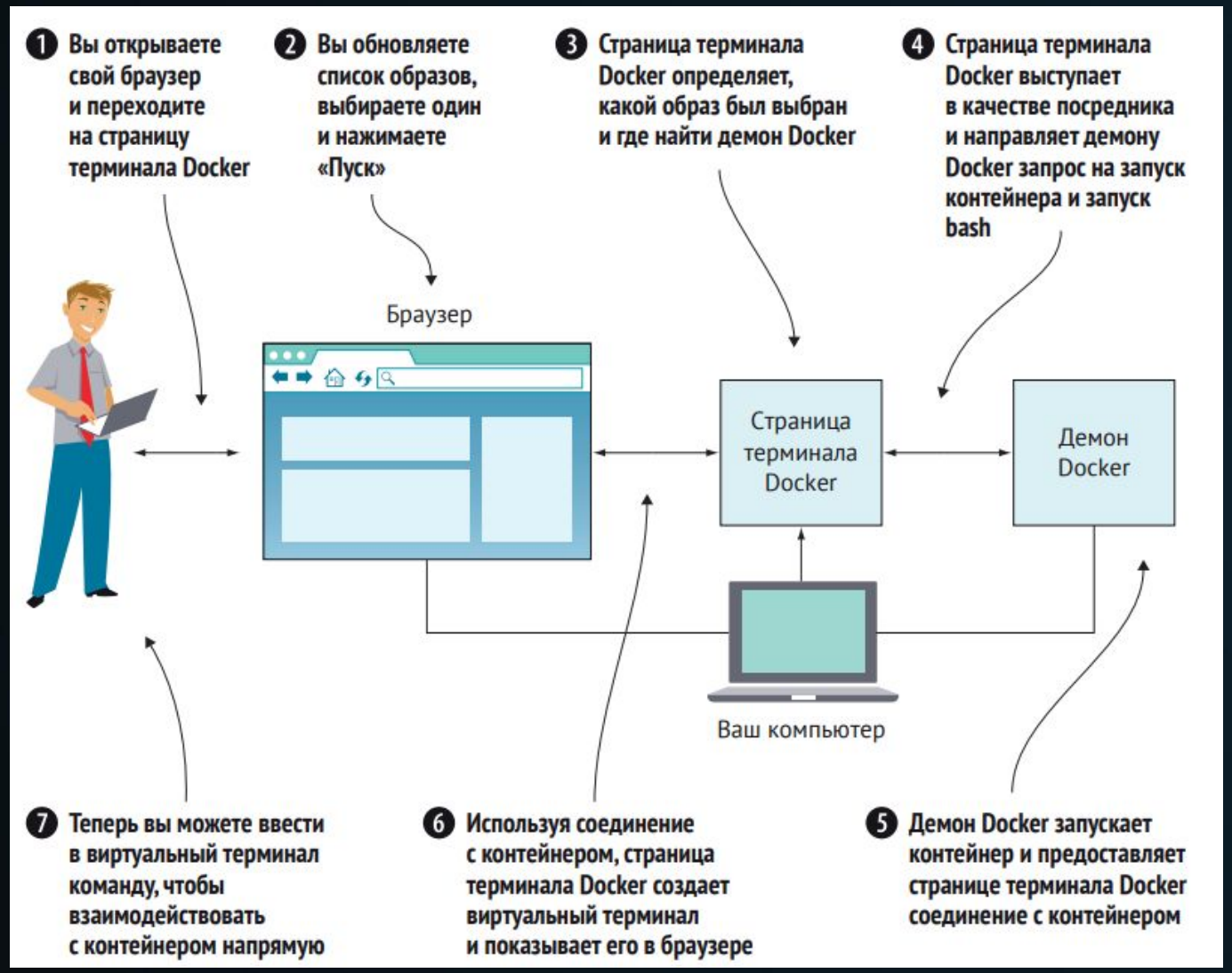
Клиент Docker

Клиент Docker – самый простой компонент в архитектуре Docker.

Это то, что вы запускаете, когда набираете такие команды, как **docker run** или **docker pull** на своем компьютере. Его задача – взаимодействовать с демоном Docker посредством HTTP-запросов.

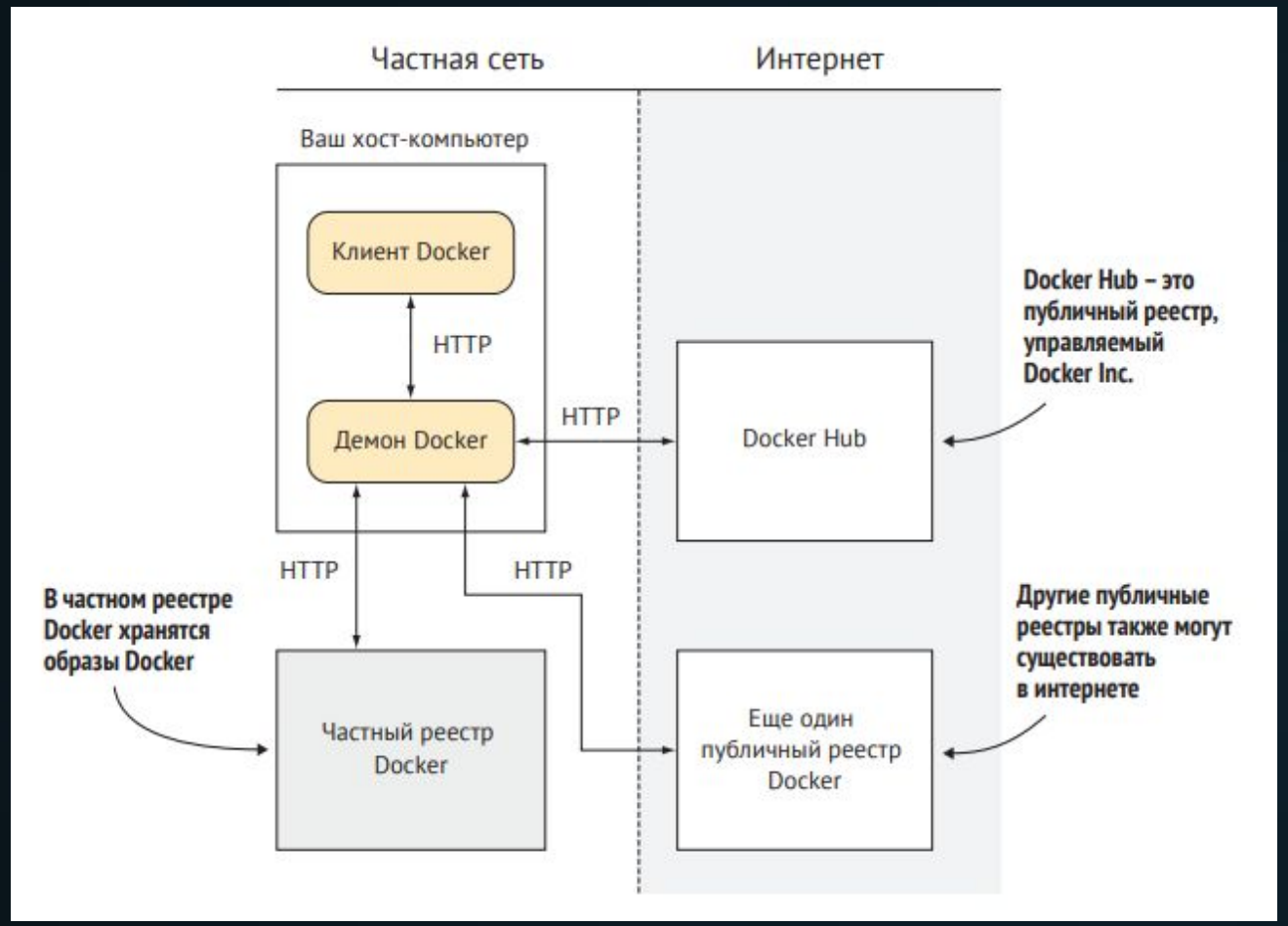


Как работает терминал Docker



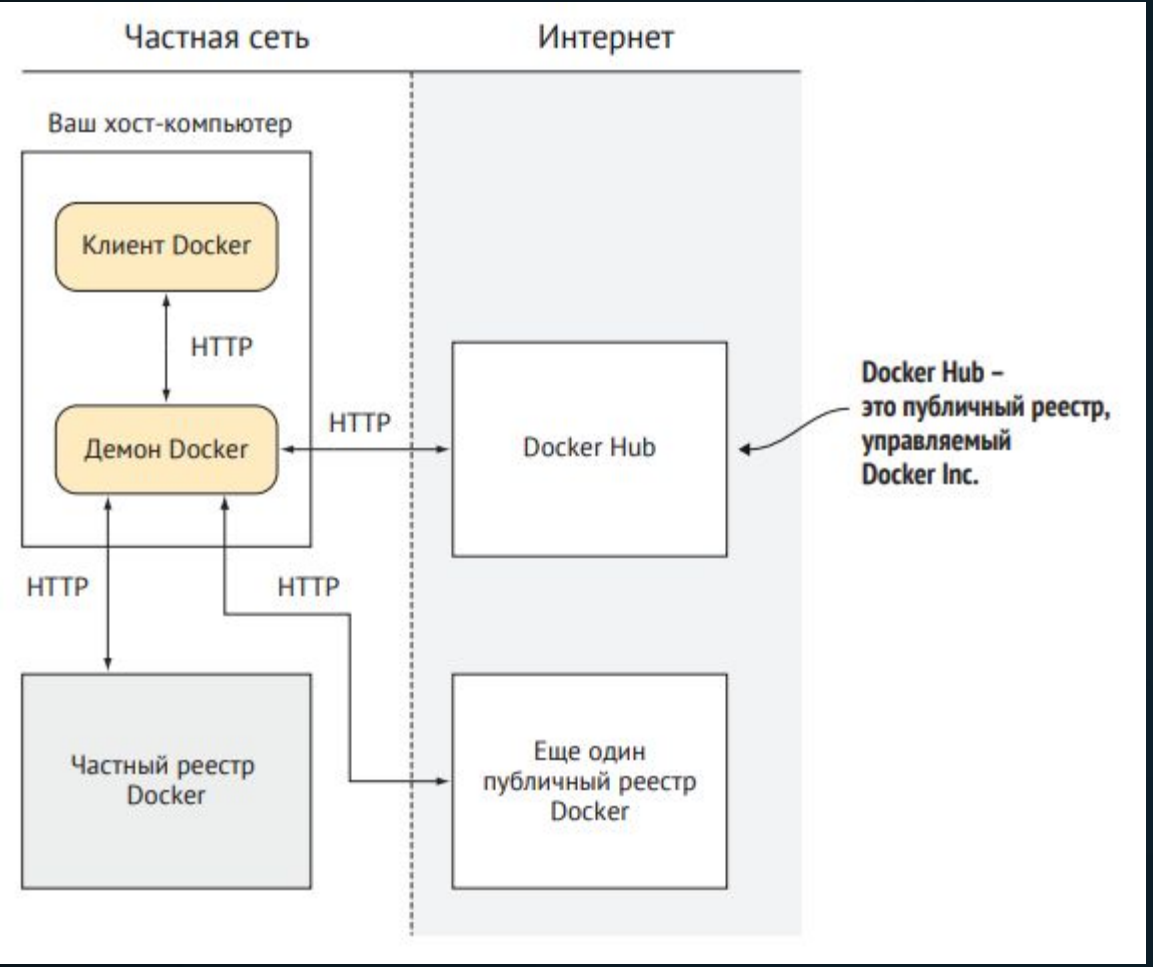
Реестры Docker

Три реестра на рисунке различаются по своей доступности. Один находится в частной сети, один открыт во внешней сети, а другой общедоступен, но только для тех, кто зарегистрирован в Docker. Все они выполняют одну и ту же функцию с одним и тем же API, и вот откуда демон Docker знает, как обмениваться с ними информацией на взаимозаменяемой основе. Код реестра, как и сам Docker, – это ПО с открытым исходным кодом.



Docker Hub

Docker Hub – это реестр, поддерживаемый Docker Inc. Он содержит десятки тысяч образов, готовых к загрузке и запуску. Любой пользователь Docker может создать бесплатную учетную запись и хранить там общедоступные образы Docker. В дополнение к предоставленным пользователями официальные образы поддерживаются для справочных целей.



Команды

Команды	Значение
sudo snap install docker	Установка Docker для ОС Linux
sudo docker images	Просмотр образов
sudo docker image rm *IMAGE_ID* или *NAMES*	Удалить образ
sudo docker ps -a	Просмотреть контейнеры (ключ -a показывать остановленные контейнеры)
sudo docker container rm *CONTAINER_ID* или *NAMES*	Удалите контейнер
sudo docker pull *NAMES*	Скачайте образ
sudo docker run -it *NAMES*	Запускает образ в интерактивном режиме
sudo docker stop *CONTAINER_ID* или *NAMES*	Остановка контейнера

Пример создания собственного образа в Linux Ubuntu

Создание собственного образа в Linux Ubuntu может быть полезным, если вы хотите создать настраиваемый образ для развертывания виртуальных машин или для установки на несколько компьютеров. Вот шаги, которые необходимо выполнить для создания собственного образа в Linux Ubuntu:

1. Установите необходимое ПО. Для создания образа вам нужно установить программы debootstrap и qemu-utils, которые позволяют создавать и работать с образами.

sql Copy code

```sql
sudo apt-get update
sudo apt-get install debootstrap qemu-utils
```

2. Создайте папку, где будет храниться образ.

perl Copy code

```perl
mkdir my-image
```

3. Создайте базовый образ Ubuntu внутри папки, используя команду debootstrap. В качестве параметра указывается версия Ubuntu, которую вы хотите использовать (например, 18.04).

cpp Copy code

```cpp
sudo debootstrap --variant=minbase bionic my-image
```

4. Подключите виртуальный диск к созданному образу.

css Copy code

```css
sudo qemu-img create -f qcow2 my-image.img 2G
```

5. Создайте loop-устройство, чтобы можно было монтировать образ и копировать в него файлы.

bash Copy code

```bash
sudo losetup /dev/loop0 my-image.img
```

Пример создания собственного образа в Linux Ubuntu

6. Создайте файловую систему в созданном виртуальном диске.

bash Copy code

```bash
sudo mkfs.ext4 /dev/loop0
```

7. Смонтируйте виртуальный диск и скопируйте содержимое базового образа Ubuntu в созданный образ.

perl Copy code

```perl
sudo mount /dev/loop0 my-image
sudo cp -a my-image.bionic/* my-image/
```

8. Отмонтируйте образ.

perl Copy code

```perl
sudo umount my-image
```

9. Отключите loop-устройство.

bash Copy code

```bash
sudo losetup -d /dev/loop0
```

10. Теперь вы можете использовать созданный образ для установки Ubuntu на другие компьютеры или виртуальные машины.

perl Copy code

```perl
sudo qemu-system-x86_64 -hda my-image.img -m 512 -boot c
```

Примеры созданных образов Docker Linux

1. Образ Nginx

sql Copy code

```
# Dockerfile
FROM ubuntu:latest
RUN apt-get update && apt-get install -y nginx
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

2. Образ Node.js

sql Copy code

```
# Dockerfile
FROM ubuntu:latest
RUN apt-get update && apt-get install -y curl
RUN curl -sL https://deb.nodesource.com/setup_14.x | bash -
RUN apt-get install -y nodejs
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

Это только несколько примеров созданных образов Docker. Образы могут быть созданы для различных инструментов, технологий и приложений, включая базы данных, веб-серверы, инструменты разработки и т.д.

Примеры созданных образов Docker Linux

3. Образ PostgreSQL

sql Copy code

```sql
# Dockerfile
FROM ubuntu:latest
RUN apt-get update && apt-get install -y postgresql
USER postgres
RUN /etc/init.d/postgresql start && \
    psql --command "CREATE USER docker WITH SUPERUSER PASSWORD 'docker';" && \
    createdb -O docker docker
EXPOSE 5432
CMD ["postgres"]
```

4. Образ Python Flask

sql Copy code

```sql
# Dockerfile
FROM ubuntu:latest
RUN apt-get update && apt-get install -y python3-pip
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt
COPY . .
ENV FLASK_APP=app.py
EXPOSE 5000
CMD ["flask", "run", "--host=0.0.0.0"]
```