Docker
внутреннее устройство

Презентацию подготовили:

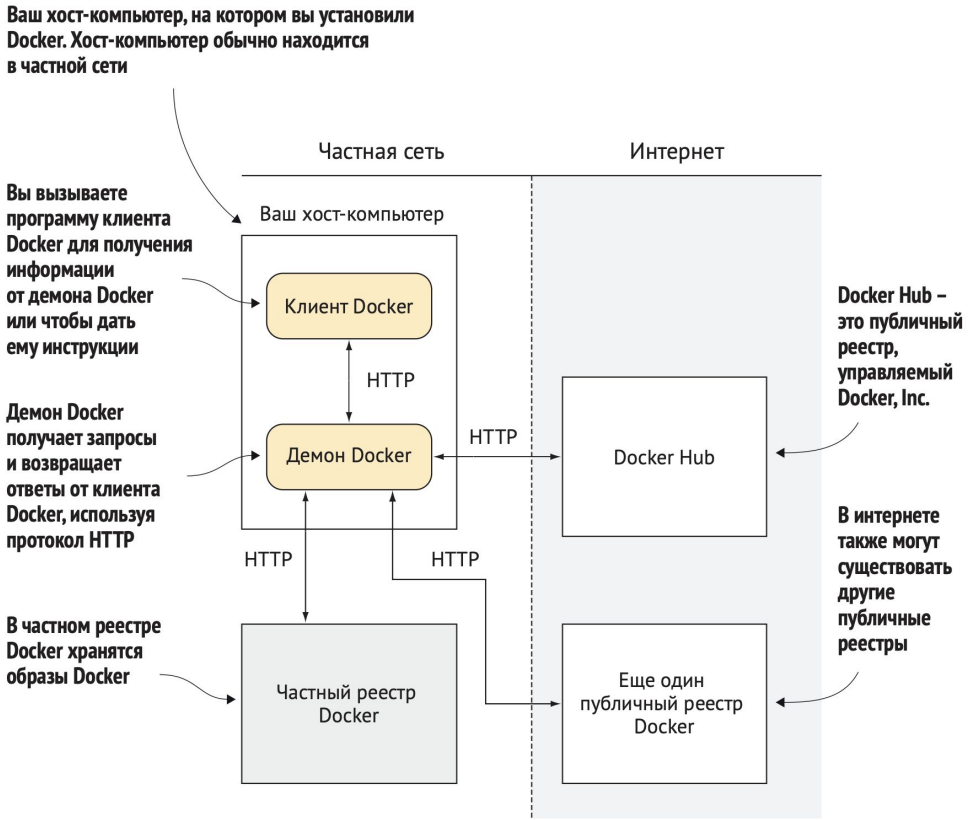
Бабахина Софья
Липс Екатерина

гр. 5040102/50201

Архитектура Docker

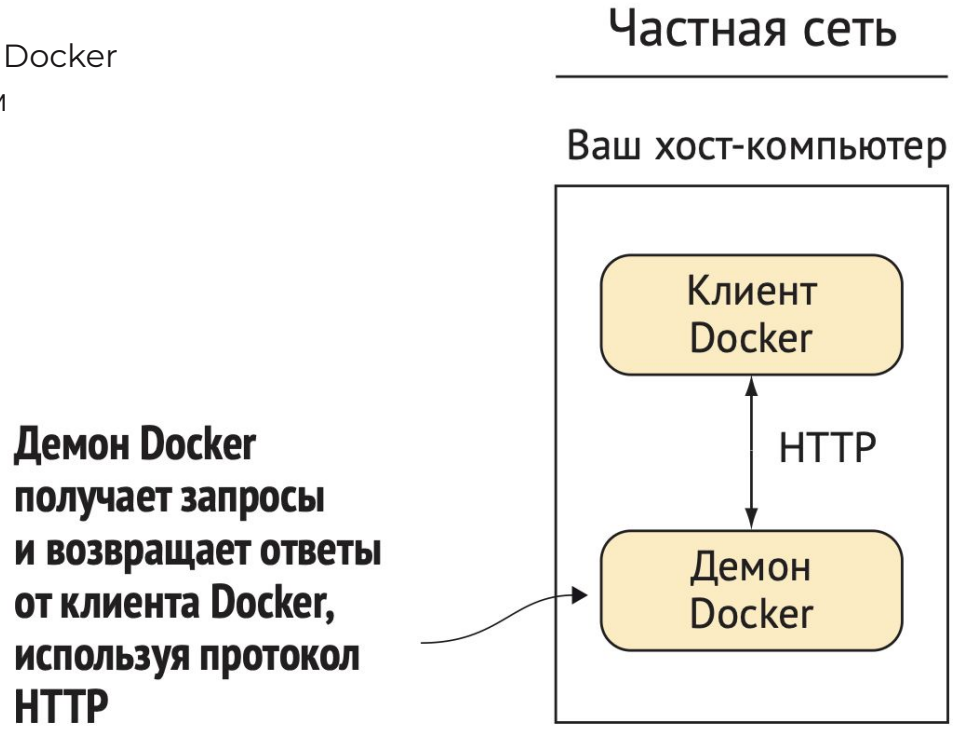
Docker:

- демон с прикладным программным интерфейсом RESTful
- клиент, который общается с демоном



Демон Docker

Демон Docker контролирует доступ к Docker на компьютере, управляет состоянием контейнеров и образов, а также взаимодействует с внешним миром.



Метод 1: Сделайте демон Docker доступным

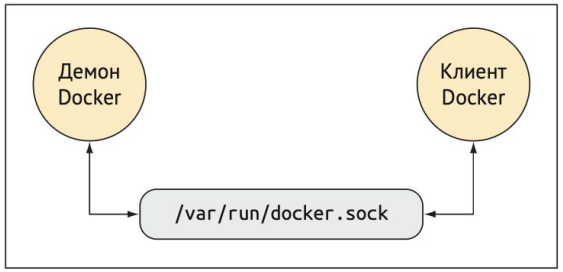
Проблема
Вы хотите открыть свой сервер Docker, чтобы у других был доступ.

Решение
Запустите демон Docker с открытым TCP адресом, для этого:

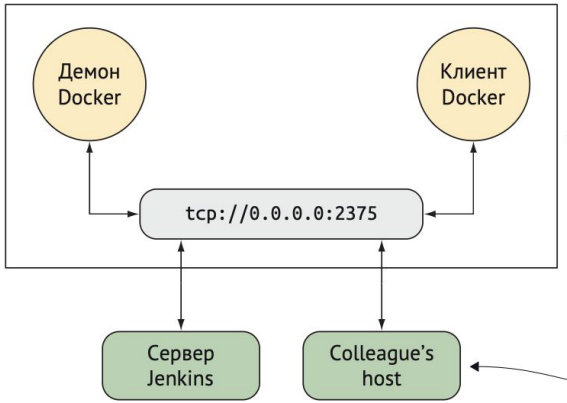
1. Закрыть работающий демон Docker
   $ sudo service docker stop
   или
   $ systemctl stop docker

2. Перезапустить Docker и открыть для внешних пользователей
   $ sudo docker daemon -H tcp://0.0.0.0:2375

3. Подключиться снаружи
   $ docker -H tcp://<your host's ip>:2375 <subcommand>



Конфигурация Docker по умолчанию, доступ к которой ограничен доменным сокетом /var/run/docker.sock. Процессы, внешние по отношению к хосту, не могут получить доступ к Docker



Используя открытый доступ к демону Docker из этого метода, доступ можно получить через TCP-сокет 2375, доступный для всех, кто может подключиться к вашему хосту (что очень небезопасно!)

Третьи стороны могут получить доступ к демону Docker. Сервер Jenkins и хост коллеги подключаются к IP-адресу хоста на порту 2375 и могут читать и писать запросы и ответы, используя этот канал

Метод 2: Запуск контейнеров в качестве демонов

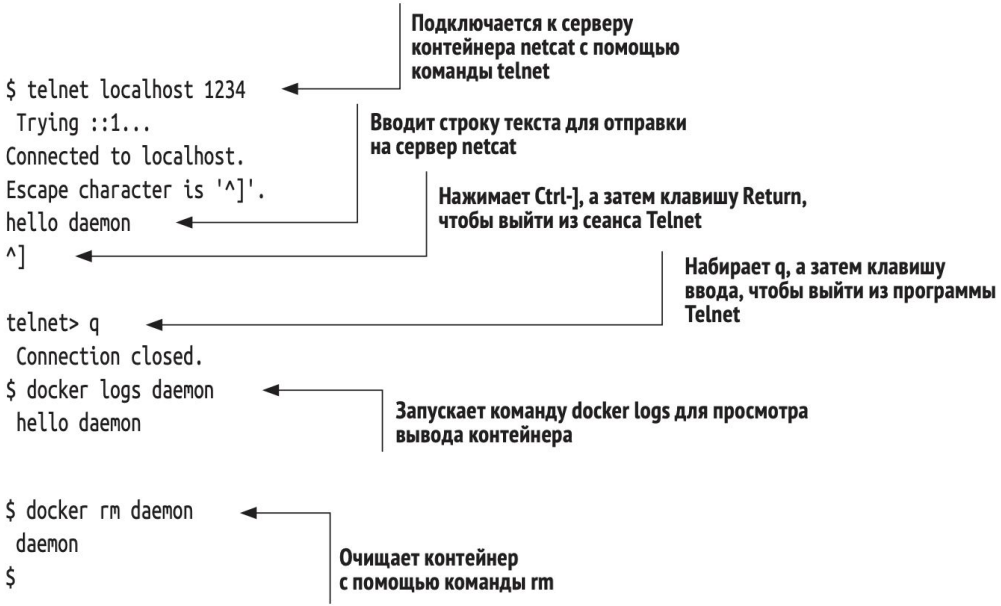
Проблема
Вы хотите запустить контейнер Docker в фоновом режиме как службу.

Решение
Используйте флаг -d в команде docker run и связанные флаги управления контейнерами для определения характеристик службы.

```
$ docker run -d -i -p 1234:1234 --name daemon
ubuntu:14.04 nc -l 1234
```

Чтобы увидеть сообщения:

```
docker logs
```



Метод 2: Запуск контейнеров в качестве демонов

Ключевые флаги

Флаг	Описание
-d	запуск в фоне (демон)
-p	проброс портов
--name	имя контейнера для удобства
--restart	политика перезапуска

Флаги перезапуска (--restart)

Стратегия	Описание
no	Не перезапускать при выходе контейнера
always	Всегда перезапускать при выходе контейнера
unless-stopped	Всегда перезагружать, но помнить о явной остановке
on-failure[:max-retry]	Перезапускать только в случае сбоя

Метод 3: Перемещение Docker в другой раздел

Проблема

Вы хотите перейти туда, где Docker хранит свои данные.

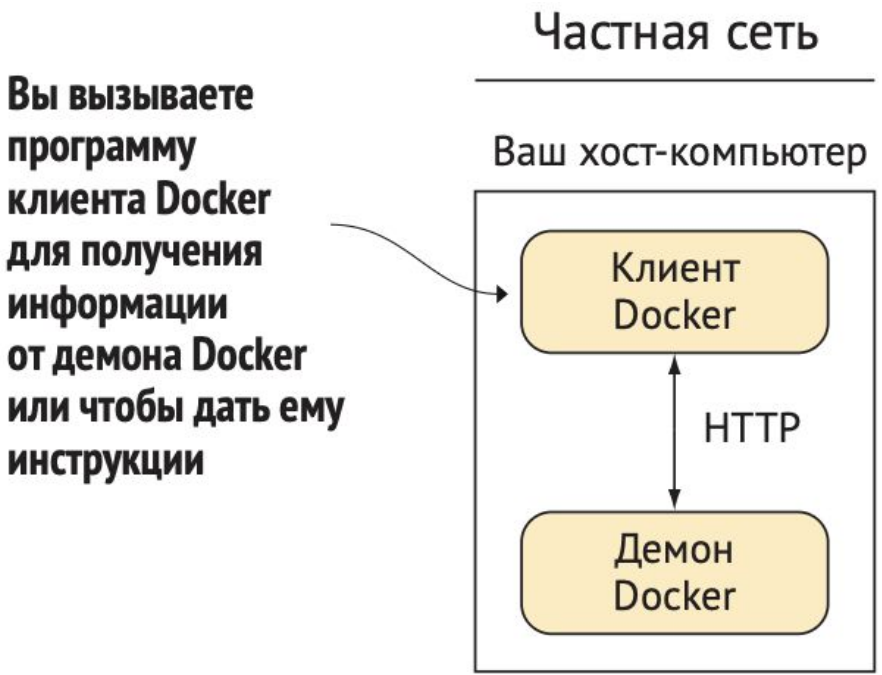
Решение

Остановите и запустите демон Docker, указав новое местоположение с помощью флага -g.

```
dockerd -g /home/dockeruser/mydocker
```

Клиент Docker

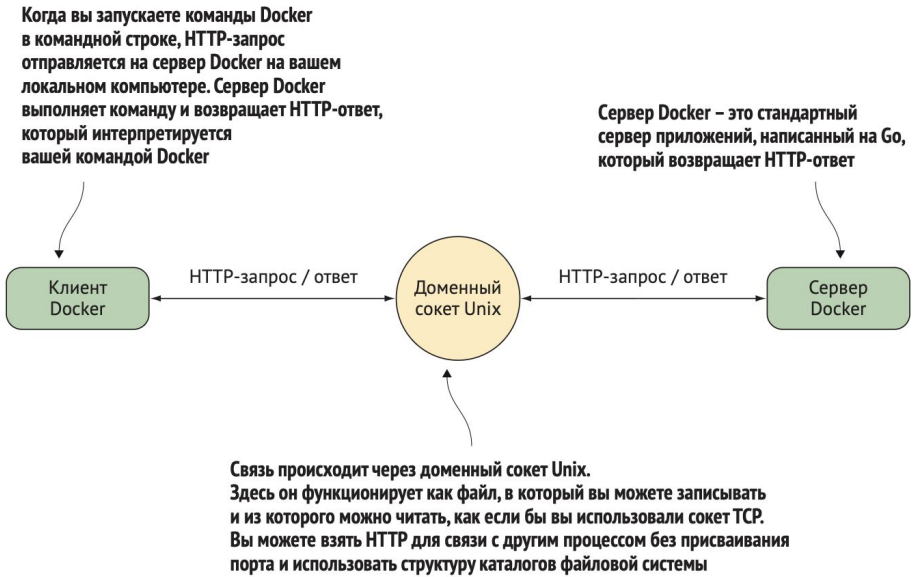
Клиент Docker взаимодействует с демоном Docker посредством HTTPзапросов.



Метод 4: Мониторинг API через socat

Проблема: Вы хотите отладить проблему с помощью команды Docker.

Решение: Используйте программу для контроля сетевого трафика, чтобы проверять API вызовы и создавать свои собственные.



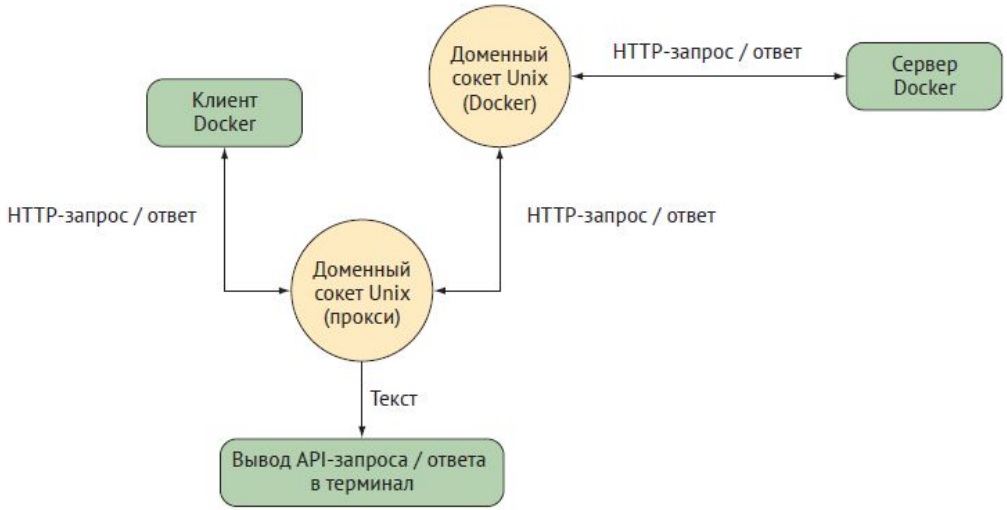
Метод 4: Мониторинг API через socat

```
# Терминал 1: Запускаем socat-прокси (нужен sudo)
sudo socat -v UNIX-LISTEN:/tmp/dockerapi.sock,fork \
UNIX-CONNECT:/var/run/docker.sock

# Терминал 2: Отправляем команду через прокси
docker -H unix:///tmp/dockerapi.sock ps

# В первом терминале видим детальный HTTP-трафик
```

Метод 4: Мониторинг API через socat

Проблема: Вы хотите отладить проблему с помощью команды Docker.

Решение: Используйте программу для контроля сетевого трафика, чтобы проверять API вызовы и создавать свои собственные.



Метод 5: Docker в браузере

Идея: Управление Docker через веб-интерфейс, используя тот же REST API.

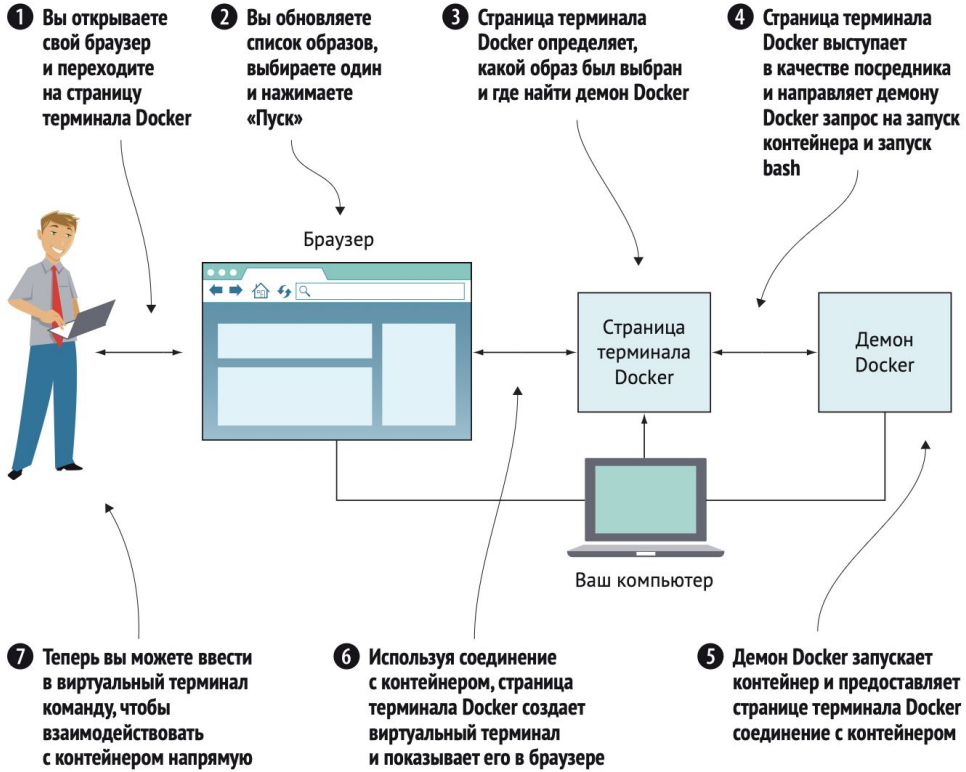
Компоненты:

- Демон с открытым портом и CORS
- Веб-страница, которая делает AJAX-вызовы к API

Результат: Открываем http://localhost:8000 и управляем контейнерами из браузера

```
# Запуск демона с поддержкой CORS
dockerd -H tcp://0.0.0.0:2375 --api-enable-cors=true

# Запуск веб-интерфейса
git clone https://github.com/aidanhs/Docker-Terminal.git
cd Docker-Terminal
python3 -m http.server 8000

# Для Windows
docker run -d -p 8000:9000 --name portainer -v //var/run/docker.sock:/var/run/docker.sock portainer/portainer-ce
```

Метод 5: Docker в браузере



Проброс портов - основа сетевого доступа

Проблема: Как получить доступ к службе внутри контейнера с хост-машины?

Решение: Флаг -p <host_port>:<container_port>

- <host_port> - порт на хост-машине
- <container_port> - порт внутри контейнера

Пример: -p 10001:80 - подключение к localhost:10001 попадет на порт 80 контейнера.

Демонстрация 6: Мульти-сервисы на одном хосте

Цель: Запустить несколько веб-серверов на разных портах хоста

```
# Запускаем два контейнера nginx на разных портах
docker run -d -p 10001:80 --name web1 nginx:alpine
docker run -d -p 10002:80 --name web2 nginx:alpine

# Проверяем проброс портов
docker ps
# Должны увидеть:
# 0.0.0.0:10001->80/tcp
# 0.0.0.0:10002->80/tcp

# Проверяем в браузере или curl
curl http://localhost:10001
curl http://localhost:10002

# Убираем за собой
docker rm -f web1 web2
```

Пользовательские сети: изоляция и DNS

Проблема: Как обеспечить безопасное общение контейнеров друг с другом?

Решение: Создание пользовательских сетей Docker

Преимущества:

- DNS - автоматическое разрешение имен по **--name**
- Изоляция - контейнеры в разных сетях не видят друг друга
- Безопасность - не нужно открывать порты на хосте

Команды:

- **docker network create** - создать сеть
- **docker network connect** - подключить контейнер к сети
- **--network** - запустить контейнер в определенной сети

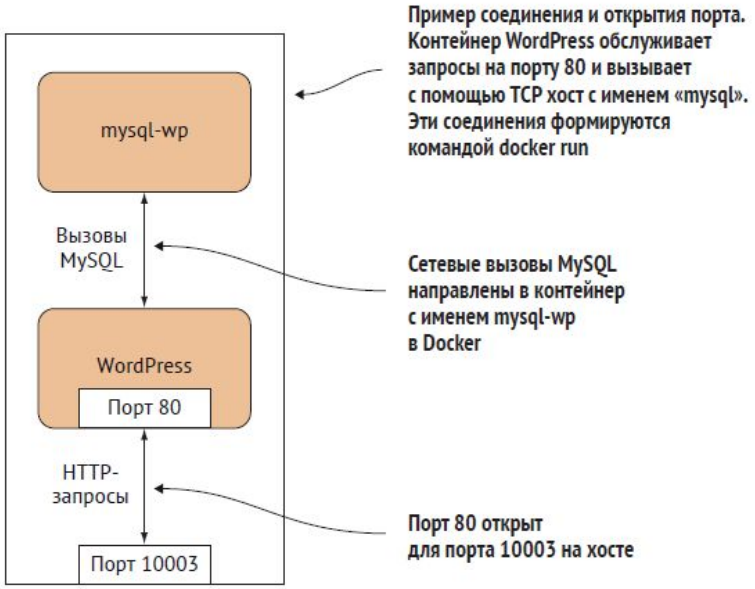


Рис. 2.8 ❖ Настройка WordPress с соединенными контейнерами

Демонстрация 7: Сети и DNS в действии

Цель: Показать, как контейнеры в одной сети общаются по именам

```
# Создаем пользовательскую сеть
docker network create my_app_net

# Запускаем веб-сервер в этой сети
docker run -d --name web-server --network my_app_net nginx:alpine

# Запускаем клиентский контейнер в той же сети
docker run -it --network my_app_net alpine sh

# ВНУТРИ контейнера выполняем:
# apk add curl  # если curl не установлен
# curl http://web-server
# exit
```

Реестры Docker - хранилища образов

Что такое реестр Docker?

- Хранилище образов Docker (аналог Git-репозитория)
- Docker Hub - публичный реестр по умолчанию
- Частные реестры - можно развернуть самостоятельно

Аналогия:

- Docker Hub ≈ GitHub (публичные репозитории)
- Частный реестр ≈ GitLab (приватные репозитории)

Основные операции:

- **docker pull** - загрузить образ
- **docker push** - отправить образ
- **docker search** - найти образ

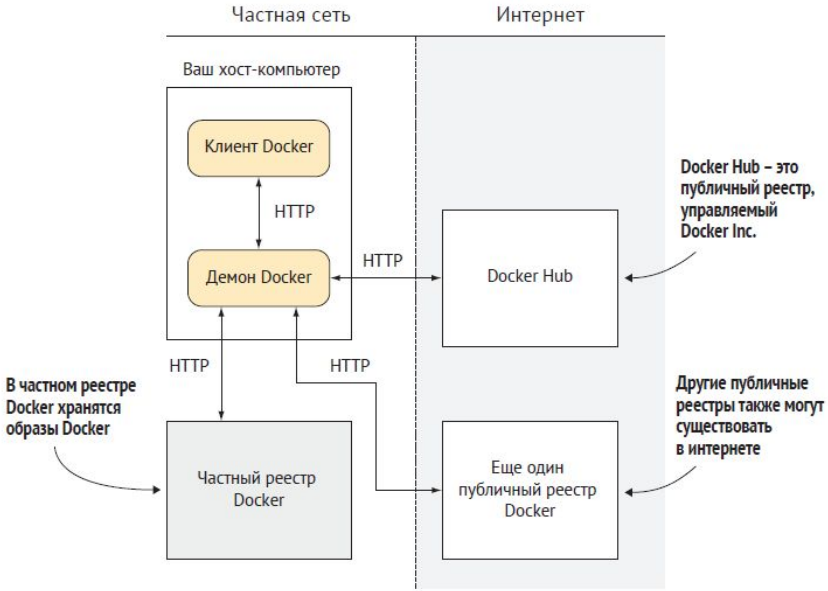


Рис. 2.9 ❖ Реестр Docker

Метод 9: Собственный реестр за 60 секунд

Проблема: Нужно хранить образы локально, приватно и быстро
Решение: Запуск официального образа registry:2

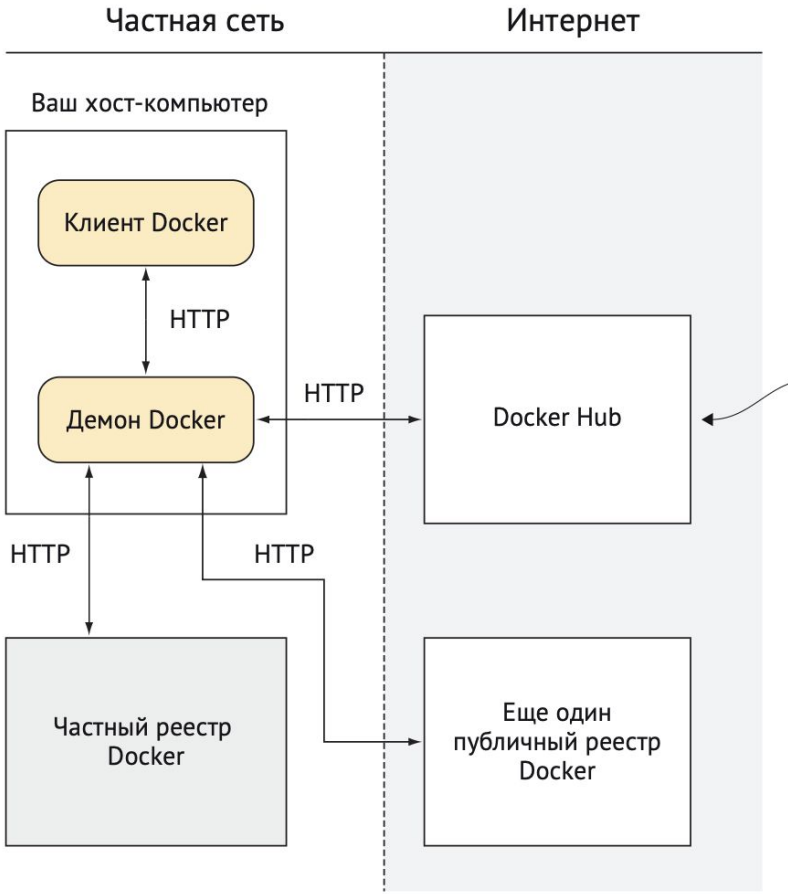
```
# Запускаем локальный реестр на порту 5000
docker run -d -p 5000:5000 --name my-registry registry:2

# Тегируем существующий образ для нашего реестра
docker tag nginx:alpine localhost:5000/my-nginx

# Пушим образ в локальный реестр
docker push localhost:5000/my-nginx

# Удаляем локальные образы
docker image remove nginx:alpine localhost:5000/my-nginx

# И пуляем обратно из нашего реестра!
docker pull localhost:5000/my-nginx
```

Docker Hub

Docker Hub - это это реестр, поддерживаемый Docker Inc. Он содержит десятки тысяч образов, готовых к загрузке и запуску.

Основные команды:

- **docker search <имя>** - поиск образов
- **docker pull <имя>** - загрузка образа
- **docker login** - аутентификация для push



Демонстрация 10: Поиск и тестирование образов

```
# Ищем образы, связанные с Node.js
docker search node

# Скачиваем официальный образ node
docker pull node

# Запускаем в интерактивном режиме  Node.js
docker run -t -i node /bin/bash
```

Литература

1. Docker Documentation: https://docs.docker.com/
2. Docker Hub: https://hub.docker.com/
3. Awesome Docker: https://github.com/veggiemonk/awesome-docker