Настройка локального репозитория для образов Docker и работа с ним

Обновлено: 10.02.2023 Опубликовано: 02.04.2021

Для хранения собственных образов Docker мы можем использовать облако Docker Hub. Однако, у нас также есть возможность создать собственный репозиторий для внутреннего использования. В данной инструкции рассмотрим процесс создания такого хранилища образов Docker, а также примеры подключения к нему.

Подготовка сервера

Выполним предварительные настройки, необходимые для корректной работы нашего сервера.

Установка Docker

Сам репозиторий является контейнеров Docker, поэтому для его запуска необходимо установить одноименный сервис.

Подробнее, об установке Docker на разные операционные системы семейства Linux читайте на странице Установка Docker на Linux.

Настройка брандмауэра

В официальной документации по настройке хранилища образов Docker приводится пример использования порта 5000. Именно его мы и будем открывать. Рассмотрим примеры использования разных утилит для настройки фаервола.

> В вашей системе не обязательно будет использоваться брандмауэр. Например, он может быть настроен, чтобы пропускать все сетевые пакеты. В таком случае, настройка, рассмотренная в данном пункте не обязательна.

а) Iptables (как правило, используется в системах на базе deb или в старых RPM).

Чтобы открыть нужный нам порт, вводим команду:

```
iptables -I INPUT -p tcp --dport 5000 -j ACCEPT
```

Для сохранения правила используем утилиту iptables-persistent:

```
apt-get install iptables-persistent

netfilter-persistent save
```

б) firewalld (как правило, используется в, относительно, новых системах на базе RPM).

Вводим команду для открытия порта 5000:

```
firewall-cmd --permanent --add-port=5000/tcp
```

Для применения правила вводим:

```
firewall-cmd --reload
```

Репозиторий для localhost

Для начала мы развернем репозиторий, который сможет обслуживать только локальный сервер (о том, как развернуть сетевое хранилище образов Docker будет рассказано ниже).

Чтобы запустить сервис, который сможет принимать запросы docker push и docker pull, вводим команду:

```
docker run -d -p 5000:5000 --restart=always --name registry registry:2
```

** в данном примере мы поднимает контейнер Docker с именем **registry** из образа **registry:2**. Он будет слушать сетевые запросы на порту **5000**. Параметр **--restart=always** позволит автоматически запускаться контейнеру после перезагрузки сервера.*

Проверяем, что на порту 5000 у нас запустился контейнер docker:

```
ss -tunlp | grep :5000
```

Мы должны увидеть что-то на подобие:

```
tcp LISTEN 0 4096 *:5000 *:* users:(("docker-proxy",pid=484238,fd=4))
```

Попробуем проверить работу нашего репозитория. На том же сервере сначала загрузим из облака докер-хаба образ, например, для python:

```
docker pull python:latest
```

Далее мы должны установить новый тег для образа, в начале названии которого должно идти указание на сервер и порт хранения образа:

```
docker tag python:latest localhost:5000/python
```

** в данном примере мы указываем тег **localhost:5000/python**, в названии которого мы видим **localhost:5000** – локальный сервер, порт **5000**.*

Загружаем образ питона на наш сервер:

```
docker push localhost:5000/python
```

Теперь удалим тот образ, который был загружен из облака:

```
docker image remove python:latest
```

... и образ, который мы загрузили в наш локальный репозиторий:

```
docker image remove localhost:5000/python
```

Теперь снова загрузим образ python, но на этот раз из нашего собственного репозитория:

```
docker pull localhost:5000/python
```

Мы должны увидеть процесс загрузки:

```
Using default tag: latest
latest: Pulling from python
8bf9c589d5f9: Pull complete
4c70e46d8b5f: Pull complete
ea848ad42f0d: Pull complete
48fe137f8d26: Pull complete
4b13f6ed9b0c: Extracting [==============>       ]
 56.82MB/192.3MB
ba85279f50e0: Download complete
59a18d8c3680: Download complete
```

```
c610993f70c6: Download complete
a9afc028cd66: Download complete
```

Который должен завершиться загрузкой образа в систему:

```
Digest:
sha256:1e3b89f69bb6ada672153256bd88d74ae60
571f6755703369a108c76595ea00f
Status: Downloaded newer image for
localhost:5000/python:latest
localhost:5000/python:latest
```

Наш репозиторий Docker готов и работает для локального сервера.

Меняем расположение хранилища для образов

Наш репозиторий для докер образов сам является контейнером Docker и данные хранит внутри себя (каталоге /var/lib/registry). Это не является правильной практикой, так как при переустановке контейнера, все данные внутри него уничтожаются. Чтобы избежать потери данных, мы должны внешнее хранилище подключить к контейнеру.

Для начала, создадим каталог на сервере, где будем хранить образы, например:

```
mkdir /dockerrepo
```

Остановим и уничтожим наш контейнер:

```
docker container stop registry && docker container
rm -v registry
```

Запустим его снова, но на этот раз мы подключим созданный каталог внутрь контейнера:

```
docker run -d -p 5000:5000 --restart=always --name
registry -v /dockerrepo:/var/lib/registry registry:2
```

** в данном примере мы используем опцию **-v** для подключения каталога **/dockerrepo** внутрь контейнера в качестве каталога **/var/lib/registry.***

Проверим работу репозитория. На этот раз загрузим другой образ, например, nginx:

```
docker pull nginx:latest
```

Добавим тег:

```
docker tag nginx:latest localhost:5000/nginx
```

Загрузим образ в репозиторий:

```
docker push localhost:5000/nginx
```

Попробуем найти загруженный образ в созданном каталоге /dockerrepo:

```
ls /dockerrepo/docker/registry/v2/repositories
```

В моем случае команда вернула каталог nginx.

Наш сервер готов к настройке доступа по сети.

SSL и подключение по сети

Репозиторий образов Docker не будет работать по сети и обслуживать другие узлы, если мы не зашифруем подключения. Для этого необходимо подключить сертификат безопасности к нашему контейнеру. Рассмотрим варианты его получения и запуска последнего с нужными опциями.

Получение сертификата

Есть 4 стандартных способа получения сертификата, которые нам подойдут. У каждого из них свои преимущества и недостатки.

1. Получение бесплатного сертификата у Let's Encrypt.

Мы можем получить бесплатный легитимный сертификат от Let's Encrypt. Для этого наш сервер должен быть доступен из внешней сети по порту 80 (для выполнения проверки домена) или мы должны перевыпускать сертификат вручную каждые 3 месяца. Подробнее о способах получения в инструкции Получение бесплатного SSL сертификата Let's Encrypt.

2. Покупка.

Если мы хотим получить валидный сертификат, но при этом не можем обеспечить доступность сервера по 80 порту (или не хотим заниматься ручным обновлением сертификата каждые 3 месяца), мы можем купить сертификат. Поставщиков данной услуги, достаточно, много, например REG.RU.

3. Самоподписанный сертификат.

Мы можем сгенерировать самозаверенный сертификат. Это простой способ, но такой сертификат не будет валидным, так как мы получим последовательность от узла, к которому нет доверия у разработчиков программного обеспечения.

Чтобы создать такой сертификат, вводим команды:

```
mkdir -p /etc/ssl/docrepo

openssl req -new -x509 -days 1461 -nodes -out
/etc/ssl/docrepo/public.pem -keyout
/etc/ssl/docrepo/private.key -subj
"/C=RU/ST=SPb/L=SPb/O=Global Security/OU=IT
Department/CN=docrepo.dmosk.local"
```

** первой командой мы создадим каталог, в котором разместим наши сертификаты. Второй – сам сертификат.*

4. Использование внутреннего центра сертификации.

Мы также можем сгенерировать валидный для нашей внутренней инфраструктуры сертификат, используя свой центр сертификации. Подробнее читайте в инструкции Сертификат для Linux в центре сертификации Active Directory Certificate Services.

Запуск внешнего репозитория

После получения сертификата, можно поднять контейнер для обслуживания сетевых запросов. Предполагается, что наши сертификаты находятся в каталоге на сервере **/etc/ssl/docrepo**.

Сначала уничтожим ранее созданный:

```
docker container stop registry && docker container rm -v registry
```

И запустим новый:

```
docker run -d -p 5000:5000 --restart=always --name registry -v /dockerrepo:/var/lib/registry -v /etc/ssl/docrepo:/certs -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/public.pem -e REGISTRY_HTTP_TLS_KEY=/certs/private.key registry:2
```

** к нашей команде добавлены опции:*

- ***-v /etc/ssl/docrepo:/certs** – монтирует каталог на сервере /etc/ssl/docrepo (с сертификатами) в каталог контейнера /certs.*
- ***-e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/public.pem** – создает системную переменную REGISTRY_HTTP_TLS_CERTIFICATE в*

контейнере, которая содержит в качестве значения путь до сертификата с открытым ключом.

- ***-e REGISTRY_HTTP_TLS_KEY=/certs/private.key*** *– создает системную переменную REGISTRY_HTTP_TLS_KEY в контейнере, которая содержит в качестве значения путь до сертификата с закрытым ключом.*

Подключаемся к другому компьютеру и пробуем загрузить наш образ для nginx:

```
docker pull docrepo.dmosk.local:5000/nginx
```

Если мы используем неподтвержденный сертификат (самоподписанный), то команда вернет нам ошибку:

```
Error response from daemon: Get
https://docrepo.dmosk.local:5000/v2/: x509:
certificate signed by unknown authority
```

Тогда на каждом компьютере, который должен обращаться к общему репозиторию Docker, открываем файл:

```
vi /etc/docker/daemon.json
```

И добавляем к настройке:

```
{
   "insecure-registries" :
["docrepo.dmosk.local:5000"]
}
```

** где **docrepo.dmosk.local:5000** – адрес и порт нашего сервера, к которому мы будем обращаться.*

Перезапускаем сервис докера:

```
systemctl restart docker
```

Можно снова пробовать загрузить образ с центрального Hub-сервера.

Аутентификация

При необходимости, мы можем настроить аутентификацию при использовании репозитория. Сначала сгенерируем сертификат с помощью утилиты htpasswd – установим ее.

** В официальной документации для генерирования файла с паролем рекомендуется применять контейнер, однако, у меня приведенная в пример команда вернула ошибку. Поэтому в данной инструкции я предлагаю универсальное решение.*

а) для Ubuntu / Debian:

```
apt-get install apache2-utils
```

б) для CentOS / Red Hat:

```
yum install httpd-tools
```

Создадим каталог, в котором разместим файл с логинами и паролями:

```
mkdir /etc/docker/auth
```

Создадим пользователя:

```
htpasswd -Bbn repouser password >
/etc/docker/auth/htpasswd
```

** в данном примере мы создадим пользователя **repouser** с паролем **password**. Мы поместим данные в файл **/etc/docker/auth/htpasswd**.*

Уничтожим ранее созданный контейнер:

```
docker container stop registry && docker container
```

```
rm -v registry
```

И запустим его снова с параметрами аутентификации:

```
docker run -d -p 5000:5000 --restart=always --name registry -v /dockerrepo:/var/lib/registry -v /etc/ssl/docrepo:/certs -v /etc/docker/auth:/auth -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/dmosk.local.pem -e REGISTRY_HTTP_TLS_KEY=/certs/dmosk.local.key -e REGISTRY_AUTH=htpasswd -e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd -e "REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm" registry:2
```

** мы добавили:*

- ***-v /etc/docker/auth:/auth** – монтируем каталог с файлом-паролем в каталог контейнера **/auth**.*
- ***-e REGISTRY_AUTH=htpasswd** – указываем системную переменную для указания необходимости способа аутентификации.*
- ***-e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd** – создаем переменную с путем до файла аутентификации.*
- ***-e "REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm"** – имя области аутентификации htpasswd (если я правильно понял, может быть произвольным).*

Репозиторий готов. На компьютере, с которого мы захотим к нему подключиться, вводим:

```
docker login docrepo.dmosk.local:5000
```

Система запросит пароль – вводим тот, что создавали с помощью утилиты htpasswd. Теперь можно работать с нашим репозиторием. Удалим загруженный ранее образ:

```
docker image rm
docrepo.dmosk.local:5000/nginx:latest
```

И загрузим его:

```
docker pull docrepo.dmosk.local:5000/nginx
```

Docker Compose

Мы можем описать наш сервер для репозитория Docker с помощью композера. Для начала, необходимо его установить.

После уничтожаем созданный контейнер:

```
docker container stop registry && docker container
rm -v registry
```

Создаем каталог – в нем мы создадим файл docker-compose:

```
mkdir -p /opt/docker_repo
```

Переходим в созданный каталог:

```
cd /opt/docker_repo
```

Создаем файл композера:

```
vi docker-compose.yml
```

```
services:

  registry:
    image: registry:2
    container_name: registry
    hostname: registry
    restart: unless-stopped
    environment:
      TZ: "Europe/Moscow"
    ports:
      - 5000:5000
```

```
    environment:
      REGISTRY_HTTP_TLS_CERTIFICATE:
/certs/public.pem
      REGISTRY_HTTP_TLS_KEY: /certs/private.key
      REGISTRY_AUTH: htpasswd
      REGISTRY_AUTH_HTPASSWD_PATH:
/auth/htpasswd
      REGISTRY_AUTH_HTPASSWD_REALM: Registry
Realm
    volumes:
      - /opt/docker_repo/data:/var/lib/registry
      - /opt/docker_repo/ssl:/certs
      - /opt/docker_repo/auth:/auth
```

** в данном примере мы описали запуск контейнера из образа registry:2 со всеми вышеописанными настройками:*

- *Порт 5000.*
- *Использовать сертификат для SSL.*
- *Требовать проходить аутентификацию.*
- *Монтировать некоторые каталоги с хостового сервера внутрь контейнера.*

Создадим каталог для сертификатов и сгенерируем необходимые ключи:

```
mkdir -p ssl

openssl req -new -x509 -days 1461 -nodes -out
ssl/public.pem -keyout ssl/private.key -subj
"/C=RU/ST=SPb/L=SPb/O=Global Security/OU=IT
Department/CN=docrepo.dmosk.local"
```

Создадим каталог для хранения файла аутентификации и сам файл с паролем:

```
mkdir auth

htpasswd -Bbn repouser password > auth/htpasswd
```

** в данном примере логи будет **repouser**, а пароль – **password**.*

Запускаем наш контейнер с помощью композера:

```
docker-compose up -d
```

** так как мы используем готовый образ, нам не нужно делать **docker-compose build**.*

Наш контейнер должен запуститься.

Читайте также

Возможно, вам будут интересны другие инструкции по Docker:

1. Настройка веб-сервера в Docker (NGINX + PHP + MariaDB)

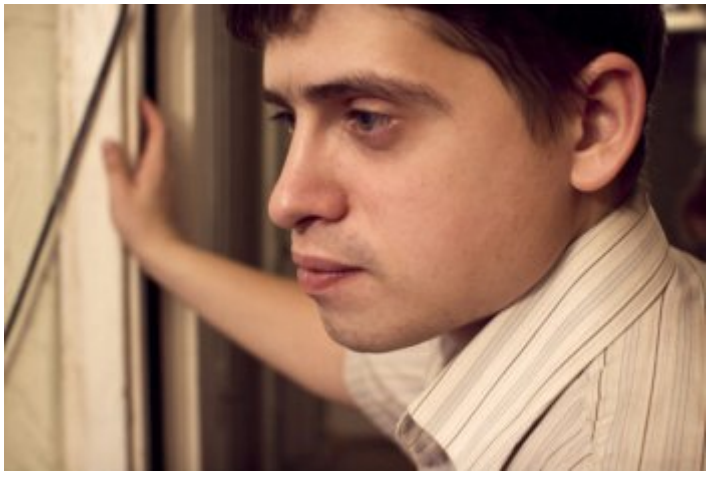
2. Создание собственного образа Docker

DevOps # Виртуализация

Была ли полезна вам эта инструкция?

Да Нет

Нравится 1



Дмитрий Моск — IT-специалист. Настройка серверов, услуги DevOps.

Заказать настройку контейнеризации

Мини-инструкции

Как настроить балансировку http-запросов в веб-сервере NGINX

Как установить, настроить и подключиться к MongoDB на Linux Ubuntu

Как установить и работать с Redis на сервере под управлением Linux Ubuntu

Как настроить свой приватный репозиторий для хранения образов Docker

Как работать с pipeline в Jenkins – подготовка системы, пример Groovy-скрипта

Как установить Jenkins на систему Linux Ubuntu Server

Инструкция по созданию виртуальной машины EC2 на хостинге Amazon Web Services

Весь список ...

Нужна помощь? Пишите:

Что хотите узнать...

Контактная эл. почта

не обязательно, но для более быстрого ответа

Получить инструкцию

Настройка серверов