docker info - Информация обо всём в установленном Docker
docker version - Версия Docker
docker tag - Дать не образу локальное имя и версию
docker login - Залогиниться в repo
docker search - Поиск образа в repo
docker pull - Скачивание образа с repo себе на хост
docker push - Отправить локальный образ в repo

Управление контейнерами

docker ps -a - Посмотреть все контейнеры
docker start container name - Запустить контейнер
docker run container-name - создает новый контейнер и сразу включает его. **(подробнее примеры)**
docker kill/stop container name - Убить (SIGKILL)/Остановить (SIGTERM) контейнер
docker logs -tail 100 container name - Вывести logs контейнера, последние 100 строк
docker inspect container-name - Все info контейнера в JP
docker rm container-name - Удалить контейнер (нельзя удалить живой образ Docker'ов)
docker rm -f $(docker ps -aq) - Удалить все запущенные и остановленные контейнеры
docker events container-name
docker port container-name - Показать публичный порт контейнера
docker top container-name - Отобразить процессы в контейнере
docker stats container-name - Статистика использования ресурсов в контейнере
docker diff container-name - Изменения в ФС контейнера

Управление образами

docker build -t my_app . - Билд контейнера в текущей папке, Dockerfile должен быть в папке запуска образа
docker images / docker image ls - Показать все образы и сумму
docker image rm / docker rm image - Удалить image
docker commit <container-name/ID> repo/dockerlabel:tag-version - Создает образ из контейнера
docker insert URL -c scriptname dpkn ps URL и выполнение скрипта dpkn.ps, загруженного по URL
docker exec -it container-name /bin/bash (exit для выхода) - Интерактивно подключиться к контейнеру, запустить bash, чтобы выйти
docker attach container-name - Подключиться к контейнеру чтобы мониторить shell/logs или в realtime

Подробное "docker run"

Синтаксис операций: **docker run [опции] название образа [команда/аргументы]**.
docker run -d -p 80:80 -p 22:22 debian:11.1:latest /usr/sbin/sshd - запустить контейнер с Debian 11.1 в фоновом режиме, пробросить 80 и 22 порты, запустить в контейнере sshd.
Контейнер интерактивно или как демон(flag/detach) (-d), Пусть: слева хостовая система, справа контейнер, пробросить порт 80 хоста на порт 80 контейнера, пробросить порт 22, запустить образ Debian 11 в режиме бесконечной сплящки.
docker run -d -p 80:80 -p 22:22 -v /home/test:/var/www/html debian:11.1:latest /usr/sbin/sshd - то же, но с пробросом папки /home/test в /var/www/html.
docker exec -it container-name /bin/bash (exit для выхода) - Интерактивно подключиться к контейнеру, запустить bash, чтобы выйти
docker attach container-name - Подключиться к контейнеру чтобы мониторить shell/logs или в realtime

Управление разделами [Volumes]

docker cp <container>:/file . - Скопировать в корень контейнера file в Docker
docker cp <container>:/file . - Скопировать file из корня контейнера в текущую директорию хостовой стороны
docker volume create todo-db - Создать volume для постоянного хранения файлов
docker run -d -p 3000:3000 --name todo -v todo-db:/etc/todos container-name - Запустить наш volume todo-db и контейнер (том ок когда мы не замораживаем где конкретно хранит данные)
docker run -ap 3000:3000 --name=todo --mount source=todo-db,target=/etc/todos container-name - Запустить наш volume todo-db и контейнер
docker volume ls - Отобразить список всех volume'ов
docker volume inspect - Инспекция volume'ов
docker volume rm - Удалить volume

Управление сетью [Network]

docker network create todo-app - Создать сеть
docker network ls - Удалить сеть
docker network rm - Удалить сеть
docker network inspect - Все информация о сети
docker network connect - Соединиться с сетью
docker network disconnect - Отсоединиться от сети

Пример 1: Hello world

Пусть можем запустить ваш первый контейнер:

```
docker run -d -p 80:80 docker/getting-started
```

Вывод консоли:

```
Unable to find image 'docker/getting-started:latest' locally
latest: Pulling from docker/getting-started
4b7ed9a4f909: Pull complete
d7d0a0a0a0a0: Pull complete
0a0a0a0a0a0a: Pull complete
0b0b0b0b0b0b: Pull complete
Digest: sha256:d6f0f0d0f0d0f0d0f0d0f0d0f0d0f0d0f0d0f0d0f0d0f0d0f0d0f0d0f0d0f0
Status: Downloaded newer image for docker/getting-started:latest
d1e2f3a4b5c6d7e8f9a0b1c2d3e4f5a6b7c8d9e0f1a2b3c4d5e6f7a8b9c0d1
```

Вывод docker run - команда, запускающая контейнер.
docker -d - Запустить контейнер в фоновом режиме (демонизирует его).
-p 80:80 - Пусть слева хостовая система, а вот есть на первый номер:

```
docker ps -a
```

Вывод:

```
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS          PORTS     NAMES
d1e2f3a4b5c6   docker/getting-started   "docker-entrypoint.s"   7 minutes ago   Up 7 minutes    80/tcp    gracious_hawking
d4e5f6a7b8c9   docker:latest            "docker-entrypoint.s"   2 days ago      Exited (137) 3 days ago   gifted_nash
```

Контейнер остановлен. Вы можем запустить его заново:

```
docker start daemon
```

Удостоверимся, что он работает:

```
docker ps -a
```

Вывод:

```
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS          PORTS     NAMES
d1e2f3a4b5c6   docker/getting-started   "docker-entrypoint.s"   7 minutes ago   Up 1 minute     80/tcp    gracious_hawking
d4e5f6a7b8c9   docker:latest            "docker-entrypoint.s"   2 days ago      Exited (0) 7 minutes ago  gifted_nash
```

Теперь остановим его опять и удалим все контейнеры вручную:

```
docker stop daemon
docker rm -ryuor first container name
docker rm daemon
```

Чтобы удалить все контейнеры, мы можем использовать следующую команду:

```
docker rm -f $(docker ps -aq)
```

docker rm -f - команда для удаления контейнера;
-f — флаг для rm, который останавливает контейнер, если он работает (принудительное удаление);
-q — флаг для ps, который выводит только ID контейнеров.

Пример 2: Nginx

Настроим запуск создания и запуска более полезный контейнер типа Nginx.

Синтаксис на этот случай:

```
docker run -d --name test-nginx -p 80:80 -v $(pwd):/var/www/html/ngx/html/ngx:latest
```

Вывод консоли:

```
Unable to find image 'ngx/html/ngx:latest' locally
latest: Pulling from library/ngx
a2abf6c4d29d: Pull complete
6dc0c0c0c0c1: Pull complete
c3d0f0f0e0d0: Pull complete
Digest: sha256:b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0
Status: Downloaded newer image for ngx/html/ngx:latest
43a6d0227c4d4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b
docker: Error response from daemon: Bind mount failed: '/home/test/ngx' does not exist.
See 'docker run --help'.
```

Важно: команда run принимает только абсолютные пути. В нашем примере мы использовали $(pwd), чтобы использовать абсолютный путь текущей директории.

Теперь мы можем открыть localhost в нашем браузере.

```
docker stop test-nginx
docker rm -f test-nginx
mkdir -p ~/var/www/html/ngx && touch ~/var/www/html/ngx/index.html
docker run -d --name test-nginx -p 80:80 -v ~/var/www/html/ngx:/usr/share/nginx/html:ro nginx:latest
```

Вывод консоли:

```
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
a2abf6c4d29d: Pull complete
6dc0c0c0c0c1: Pull complete
c3d0f0f0e0d0: Pull complete
Digest: sha256:b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0
Status: Downloaded newer image for nginx:latest
43a6d0227c4d4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b4b
docker: Error response from daemon: Bind mount failed: '~/var/www/html/ngx' does not exist.
See 'docker run --help'.
```

Важно: команда run принимает только абсолютные пути. В нашем примере мы использовали $(pwd), чтобы использовать абсолютный путь текущей директории.

Пример 3: написание Dockerfile

Чтобы создать Docker-образ, нам необходимо создать **Dockerfile**. Это просто текстовый файл с инструкциями и аргументами. Для описания инструкций, которые мы будем использовать в нашем следующем примере:

FROM — устанавливает базовый образ.
ENV — выставляет переменную среды.
RUN — выполняет команды в текущем окружении.
WORKDIR — устанавливает рабочий директорию.
COLLAPSE — создает новую контейнерную для того.
CMD — выполняется при запуске контейнера.

Мы можем использовать эти инструкции для сборки нашего Dockerfile:

```
mkdir -p ~/docker/test && cd ~/docker/test && touch Dockerfile
```

Перейдем в example/test и выполним следующую команду:

```
vim Dockerfile && ls -lah Dockerfile
```

Вывод консоли:

```
Reading package lists... Done
Building dependency tree...
Reading state information...
curl is already the newest version (7.74.0-1.3+deb11u7).
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
  libcap2-bin libpam-cap libpcre2-8-0 libpcre2-posix0 libsession-ini-perl libtext-wrapi18n-perl libtime-duration-perl libtime-moment-perl libtime-parseatom-perl libtime-piece-perl libtimedate-perl libtinfo6 libtry-tiny-perl libunbound8 libuuid-tiny-perl libxml2 libxslt1.1 libzstd1 lsb-release mailcap media-types netbase perl perl-base perl-modules perl5.32.1-libnet-ssleay-perl perl5.32.1-libwhich-perl publicsuffix sensible-utf8 ucf unzip xz-utils zip zlib1g zsh zsh-common zsh-doc
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
Need to get 0 B/1,000 kB of archives.
After 0 B of additional disk space will be used.
Setting up libcap2-bin (1:2.44-1) ...
Setting up libpam-cap:amd64 (1.5.2-6) ...
Setting up libpcre2-8-0:amd64 (10.39-3) ...
Setting up libpcre2-posix0:amd64 (10.39-3) ...
Setting up libsession-ini-perl (0.02-1) ...
Setting up libtext-wrapi18n-perl (0.06-7) ...
Setting up libtime-duration-perl (1.21-1) ...
Setting up libtime-moment-perl (0.44-1+b1) ...
Setting up libtime-parseatom-perl (0.01-1) ...
Setting up libtime-piece-perl (1.33-1) ...
Setting up libtimedate-perl (2.3300-2) ...
Setting up libtinfo6:amd64 (6.2+20201114-2) ...
Setting up libtry-tiny-perl (0.31-2) ...
Setting up libunbound8:amd64 (1.13.1-1+deb11u5) ...
Setting up libuuid-tiny-perl (0.03-3) ...
Setting up libxml2:amd64 (2.9.10+dfsg-6.7+deb11u4) ...
Setting up libxslt1.1:amd64 (1.1.34-4+deb11u1) ...
Setting up libzstd1:amd64 (1.4.8+dfsg-2.1) ...
Setting up lsb-release (11.1.0) ...
Setting up mailcap (3.69) ...
Setting up media-types (4.0.1) ...
Setting up netbase (6.3) ...
Setting up perl-modules-5.32 (5.32.1-4+deb11u3) ...
Setting up perl-base (5.32.1-4+deb11u3) ...
Setting up perl (5.32.1-4+deb11u3) ...
Setting up perl5.32.1-libnet-ssleay-perl (1.92-2+b1) ...
Setting up perl5.32.1-libwhich-perl (1.40-1) ...
Setting up publicsuffix (20210518.0000-1) ...
Setting up sensible-utf8 (0.0.1) ...
Setting up ucf (3.0043+nmu1+deb11u1) ...
Setting up unzip (6.0-26+deb11u1) ...
Setting up xz-utils (5.2.5-2.1+deb11u1) ...
Setting up zip (3.0-12+b1) ...
Setting up zlib1g:amd64 (1:1.2.11.dfsg-2+deb11u2) ...
Setting up zsh (5.8-6+deb11u1) ...
Setting up zsh-common (5.8-6+deb11u1) ...
Setting up zsh-doc (5.8-6+deb11u1) ...
Processing triggers for libc-bin (2.31-13+deb11u5) ...
Processing triggers for mime-support (3.69) ...
Processing triggers for man-db (2.9.4-2) ...
Processing triggers for libc-bin (2.31-13+deb11u5) ...
Processing triggers for ca-certificates (20210119) ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Processing triggers for ca-certificates-java (20190909) ...
done.
-rw-r--r-- 1 root root 1.1K Dec 15 17:00 /usr/share/doc/curl/TLS-SRP-HTTPS-FTPS-SCP-SFTP-LDAP-LDAPS-DICT-TELNET-FILE-IMAP-SMTP-POP3-RTSP-RTMP-RTMPS
```

Теперь build — создает новый контейнер:

```
-t — устанавливает именную метку для образа.
```

Теперь у нас есть новый образ, и мы можем посмотреть список существующих:

```
docker images
```

Вывод консоли:

```
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
test-curl    latest    b4c9d2e7f1a3   37 minutes ago   188 MB
<none>       <none>    a1b2c3d4e5f6   2 days ago       188 MB
debian       latest    a1b2c3d4e5f6   2 days ago       188 MB
```

Мы можем создать и запустить контейнер из образа. Попробуем сделать это со следующим параметром:

```
docker run --rm -v ~/pwd/test/data/:/w wordlist
```

Чтобы разобраться, сохраненные в файле, выполним команду:

```
cat ~/wd/results
```

Попробуем с localhost.com:

```
docker run --rm -v $(TR_URL)/results:/w $(pwd)/wd/data/:/w test curl
```

И снова посмотрим результаты:

```
cat ~/wd/results
```

Пример 4: Развертывание PROXY / ZABBIX / AGENT в docker контейнере

Docker Compose - это материал, который позволяет создавать и запускать несколько связанных контейнеров Docker как одно приложение. Он основан на файле docker-compose.yaml в формате YAML, который описывает различные сервисы, сети и тома, необходимые для вашего приложения. Docker Compose упрощает управление несколькими контейнерами, делая их запуск, остановку и масштабирование в файле "docker-compose.yml". Я этим файлом мы создаем образы Docker'а, сети, тома, переменные среды и определяем, как контейнеры взаимодействуют друг с другом, что делает развертывание сложных приложений гораздо более управляемым.

Docker Compose предоставляет простой способ определения и управления несколькими контейнерами, которые могут быть запущены на одном хосте. Он обеспечивает изолированную среду для каждого приложения, гарантируя, что контейнеры могут взаимодействовать друг с другом, не влияя на другие приложения, работающие на хосте. Он также предоставляет такие функции, как переменные среды, тома и сети, которые можно использовать для настройки поведения контейнеров и обеспечения их взаимодействия друг с другом. Это делает его ценным инструментом для разработчиков, которые хотят упростить развертывание и управление контейнерными приложениями.

Docker Compose также предоставляет такие функции, как масштабирование, которое позволяет запускать несколько экземпляров определенного контейнера, что может быть полезно для обеспечения высокой доступности и баланса нагрузки. Он также предоставляет такие функции, как проверки работоспособности, которые можно использовать для мониторинга состояния контейнеров и обеспечения их правильной работы. Docker Compose также предоставляет такие функции, как ведение журнала и мониторинг, которые можно использовать для отслеживания производительности и устранения неполадок ваших контейнеров.

Что касается Docker Compose, то можно сказать, что это мощный инструмент, который упрощает управление несколькими контейнерными приложениями. Он предоставляет простой способ определения и управления несколькими контейнерами, обеспечивает изолированную среду для каждого приложения, гарантирует, что контейнеры могут взаимодействовать друг с другом, предоставляет такие функции, как переменные среды, тома и сети, которые можно использовать для настройки поведения контейнеров и обеспечения их взаимодействия друг с другом, предоставляет такие функции, как масштабирование, проверки работоспособности, ведение журнала и мониторинг, которые можно использовать для обеспечения высокой доступности, баланса нагрузки, мониторинга состояния контейнеров и устранения неполадок ваших контейнеров. В целом, Docker Compose - это ценный инструмент для разработчиков, которые хотят упростить развертывание и управление контейнерными приложениями.

Docker Compose - это инструмент, который позволяет определять и запускать несколько связанных контейнеров как одно приложение. Он использует файл docker-compose.yaml в формате YAML, который описывает различные сервисы, сети и тома, необходимые для вашего приложения. Docker Compose упрощает управление несколькими контейнерами, делая их запуск, остановку и масштабирование в файле "docker-compose.yml". Я этим файлом мы создаем образы Docker'а, сети, тома, переменные среды и определяем, как контейнеры взаимодействуют друг с другом, что делает развертывание сложных приложений гораздо более управляемым.

Docker Compose предоставляет простой способ определения и управления несколькими контейнерами, которые могут быть запущены на одном хосте. Он обеспечивает изолированную среду для каждого приложения, гарантируя, что контейнеры могут взаимодействовать друг с другом, не влияя на другие приложения, работающие на хосте. Он также предоставляет такие функции, как переменные среды, тома и сети, которые можно использовать для настройки поведения контейнеров и обеспечения их взаимодействия друг с другом. Это делает его ценным инструментом для разработчиков, которые хотят упростить развертывание и управление контейнерными приложениями.

Docker Compose предоставляет такие функции, как масштабирование, которое позволяет запускать несколько экземпляров определенного контейнера, что может быть полезно для обеспечения высокой доступности и баланса нагрузки. Он также предоставляет такие функции, как проверки работоспособности, которые можно использовать для мониторинга состояния контейнеров и обеспечения их правильной работы. Docker Compose также предоставляет такие функции, как ведение журнала и мониторинг, которые можно использовать для отслеживания производительности