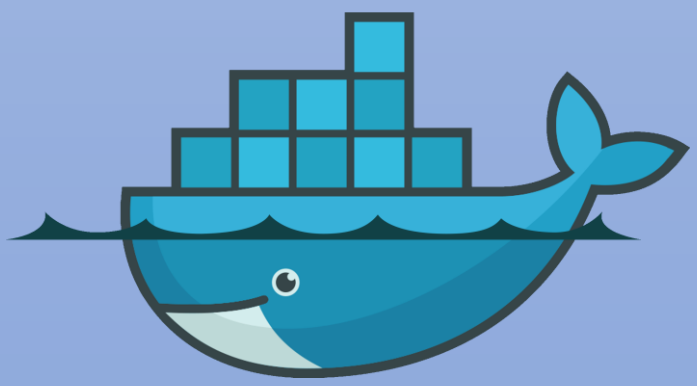
“It's lucky you're going so slowly, because you're going in the wrong direction”.

«Тебе повезло, что ты идешь так медленно, потому что идешь не в том направлении».

Dockerfile, реестры

Лабораторная работа #2

Сергей Станкевич



ЛАБОРАТОРНАЯ РАБОТА #2

Dockerfile, работа с реестрами

Цель работы

Ознакомиться с автоматизацией создания образа в Docker. Работа с реестрами Docker Hub.

Норма времени выполнения: 2 академических часа.
Оценка работы: 3 зачётных единицы.

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ.

Dockerfile

Процедуры Docker не является легко воспроизводимыми – повторение набора действий связано с трудностями и является потенциальным источником ошибок. Решение этой проблемы заключается в автоматизации повторного воспроизведения этих процедур.

Dockerfile – это обычный текстовый файл, содержащий набор операций, которые могут быть использованы для создания Docker-образа (Image).

Создание образов из файлов Dockerfile

Docker-образ создаётся во время сборки, а Docker-контейнер – во время запуска приложения.

Docker-образ – это шаблон для создания Docker-контейнеров. Представляет собой *исполняемый пакет*, содержащий все необходимое для запуска приложения: код, среду выполнения, библиотеки, переменные окружения и файлы конфигурации.

Docker-файл – сердце Docker’а. Он указывает Docker’у как построить образ, который будет использоваться при создании контейнера. Каждый Docker-образ содержит файл с именем Dockerfile (он без расширения).

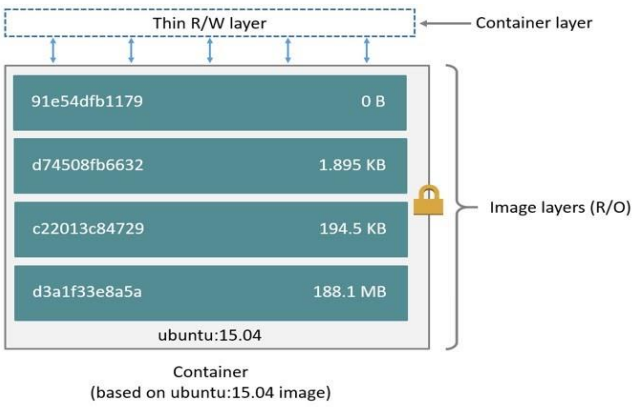
Контейнер состоит из ряда слоёв. Контейнер содержит в себе Docker-образ, который в свою очередь состоит из множества слоёв. Все слои доступны только для чтения, кроме последнего, он располагается над остальными. Docker-файл указывает порядок добавления слоёв. Каждый *слой* – это просто файл с изменением предыдущего слоя. В Unix практически всё является файлом. Базовый слой, его ещё называют родительским, это *начальный слой*.

Образы, контейнеры и файловая система Union File System

Чтобы понять взаимосвязь между образами и контейнерами, необходимо более подробно рассмотреть ключевой элемент технологии, лежащей в основе Docker, – UnionFS («каскадно-объединенное монтирование» (union mount). Файловые системы с каскадно-объединенным монтированием позволяют подключать несколько файловых систем с *перекрытием* (или наложением друг на друга), причем для пользователя они будут выглядеть как одна файловая система. Каталоги могут содержать файлы из нескольких файловых систем, но если двум файлам в точности соответствует один и тот же путь, то файл, смонтированный самым последним, скроет все ранее монтированные файлы.

Реализацию, используемую в конкретной системе, можно определить командой docker info – смотреть содержимое заголовка «Storage Driver». Файловую систему можно заменить, но это рекомендуется тогда, когда вы точно знаете, что делаете.

Образы Docker состоят из нескольких *уровней* (layers). *Каждый уровень* представляет собой *защищенную от записи файловую систему*. Для каждой инструкции в Dockerfile создается свой уровень, который размещается поверх предыдущих уровней. Во время преобразования образа в контейнер (командой docker run или docker create) механизм Docker выбирает нужный образ и добавляет на самом верхнем уровне файловую систему с возможностью записи (одновременно с этим инициализируются разнообразные параметры настройки, такие как IP-адрес, имя, идентификатор и ограничения ресурсов).



На схеме показано устройство контейнера, запущенного из образа Ubuntu 15.04. Контейнер состоит из пяти слоёв: четыре из них принадлежат образу, и лишь один – самому контейнеру. Слои образа доступны только для чтения, слой контейнера – для чтения и для записи. Если при работе приложения какие-то данные будут изменяться, они попадут в слой

контейнера. Но при уничтожении контейнера слой будет безвозвратно потерян, и все данные вместе с ним.

Контейнер может находиться в одном из следующих состояний: «**создан**» (created), «**перезапуск**» (restarting), «**активен**» или «**работает**» (running), «**приостановлен**» (paused) или «**остановлен**» (exited).

«Созданным» считается контейнер, который был инициализирован командой docker create, но его работа пока еще не началась. Состояние exited в общем случае соответствует состоянию «остановлен» (stopped), когда в данном контейнере нет активно выполняющихся процессов (их нет и в «созданном» контейнере, но остановленный контейнер уже запускался, по крайней мере один раз).

Контейнер существует, пока существует его основной процесс. Остановленный контейнер можно перезапустить командой docker start. *Остановленный контейнер* – это не то же самое, что исходный образ. Остановленный контейнер сохраняет все изменения в его параметрах настройки, метаданных и файловой системе, в том числе и параметры конфигурации времени выполнения, например IP-адрес, которые не хранятся в образах. Состояние перезапуска на практике встречается редко и возникает в тех случаях, когда механизм Docker пытается повторно запустить контейнер после неудачной первой попытки.

Инструкции Docker-файла

Инструкция Docker-файла – слово, обозначающее как правило какую ни будь команду, за которой следует аргумент. Инструкция не учитывает регистр. Тем не менее, соглашение заключается в том, чтобы они были прописными, чтобы легче отличать их от аргументов.

Каждая строка в Docker-файле может содержать инструкцию, все они обрабатываются (интерпретируется) сверху вниз. Однако первой инструкцией должна быть команда FROM. Она может быть после директив парсера, комментариев и глобальных ограничений ARG.

Docker-файл – это скрипт, текст которого может выглядеть так:

```
FROM debian:buster
RUN apt-get update && apt-get install -y cowsay fortune
```

Только инструкции FROM, RUN, COPY и ADD создают слои в конечном образе. Другие инструкции производят настройку, добавляют метаданные или же просто говорят Docker’у сделать что-либо во время запуска (например, открыть порт или выполнить команду).

Переменные окружения (Environment variables) поддерживаются списком инструкций в Dockerfile, которые можно посмотреть здесь:

https://docs.docker.com/engine/reference/builder/

Реестры, репозитории, образы и теги

Для хранения собственных образов Docker можно использовать облако Docker Hub. Также из этого хранилища можно скачивать нужные нам доступные образы. Для хранения образов применяется иерархическая система. При этом используются следующие термины:

Реестр (registry) – сервис, отвечающий за хранение и распространение образов. По умолчанию используется реестр Docker Hub.

Репозиторий (repository) – набор взаимосвязанных образов (обычно представляющих различные версии одного приложения или сервиса).

Тег (tag) – алфавитно-цифровой идентификатор, присваиваемый образам внутри репозитория (например, 14.04 или stable).

Например, команда **docker pull amouat/revealjs:latest** загрузит образ с тегом **latest** в репозиторий **amouat/revealjs** из реестра Docker Hub.

Вот что написано на официальном сайте Docker: «Docker Hub – ведущий в мире сервис для поиска образов контейнеров и обмена ими с вашей командой и сообществом Docker. Для тех, кто экспериментирует с Docker, это отправная точка для изучения контейнеров Docker. Начните изучать миллионы образов, доступных в сообществе и у проверенных издателей».

Два способа доступа к *реестру* Docker Hub:

- из командной строки, с помощью с помощью специальной команды поиска Docker;
- через официальный сайт Docker: http://registry.hub.docker.com.

Docker сам обеспечивает доступ к сервисам Docker Hub через команды docker search, pull, login и push.

Каждый уважающий себя Docker-программист должен иметь собственную учетную запись в Docker Hub. Создайте свой аккаунт.

Если вы не согласны предоставить доступ к своему образу любому желающему, можно выбрать один из двух вариантов. Воспользуйтесь платным защищенным репозиторием (на сайте Docker Hub или аналогичным сервисом, например quay.io) или создайте собственный реестр.

Пространства имен для образов

Выгружаемые Docker-образы могут принадлежать одному из трех пространств имен, определяемых по имени самого образа:

- имена, начинающиеся с *текстовой строки и слеша* (/), такие как amouat/revealjs, принадлежат пространству имен «**user**», это образы, выгруженные конкретным пользователем;
- имена, *подобные debian и ubuntu*, без префиксов и слешей, принадлежат пространству имен «**root**», управляемому компанией Docker Inc.;
- имена с *префиксами в виде имени хоста или IP-адреса* представляют образы, хранящиеся в сторонних реестрах (не в Docker Hub).

Такое распределение по пространствам имен *помогает пользователю не запутаться* при определении происхождения различных образов: используя образ debian, вы уверены, что это официальный образ из реестра Docker Hub, а не какая-то случайная версия из другого реестра.

Например, упомянутый выше amouat/revealjs – это образ revealjs, выгруженный пользователем amouat. Можно *свободно и бесплатно* выгружать общедоступные образы в репозиторий Docker Hub, который уже содержит тысячи образов.

Образы, контролируемые компанией Docker, сопровождаются в основном третьими сторонами, обычно авторами и поставщиками соответствующего ПО (например, образ nginx сопровождается компанией nginx). Для большинства широко распространенных пакетов ПО здесь представлены официальные образы, на которые следует обратить внимание в первую очередь при поиске контейнеров для производственных целей.

Образы, хранящиеся в сторонних реестрах (не в Docker Hub), могут быть собственными реестрами различных организаций, а также реестры конкурентов Hub, таких как quay.io. Например, имя localhost:5000/wordpress определяет образ WordPress, расположенный в локальном реестре.

Хранение данных в Docker

Важная характеристика Docker-контейнеров – эфемерность. В любой момент контейнер может рестартовать: закончить работу и вновь запуститься из образа. При этом все накопленные в нём данные будут потеряны. Но как в таком случае запускать в Docker приложения, которые должны сохранять информацию о своём состоянии? Поэтому нужны способы сделать так, чтобы важные изменяемые данные не зависели от эфемерности контейнеров и были доступными сразу из нескольких мест.

Остался один вопрос: как *сохранить* наши *данные* и создать их *резервную копию*? Для этого не следует использовать *стандартную файловую систему контейнера*, необходима возможность простого совместного использования хранилища данных контейнером и хостом или другими контейнерами. Docker предоставляет такую возможность посредством реализации концепции томов.

Тома (volumes) – это файлы или каталоги, которые смонтированы непосредственно на хосте и не являются частью каскадно-объединенной файловой системы. Это означает, что другие контейнеры могут совместно использовать их, и все изменения будут сразу же фиксироваться в файловой системе хоста.

Существуют два способа объявления каталога как тома с одинаковым результатом: использование инструкции VOLUME в Dockerfile или включение флага -v в команду docker run.

Первый способ, это определение каталога /data как тома внутри контейнера:

```
VOLUME /data
```

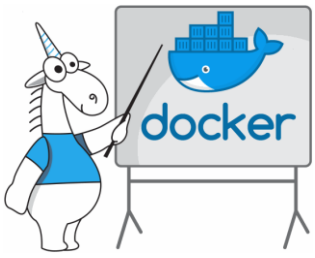
Пример второго способа с помощью консоли:

```
$ docker run -v /data test/webserver
```

По умолчанию заданный каталог или файл будет смонтирован на хосте внутри каталога, в котором был установлен Docker (обычно это каталог /var/lib/docker/). В качестве точки монтирования можно определить любой другой каталог хоста в команде docker run (например, docker run -d -v /host/dir:/container/dir test/webserver).

В файле Dockerfile определить каталог хоста как точку монтирования невозможно по причинам, связанным с обеспечением переносимости и безопасности (заданный файл или каталог может отсутствовать на других системах, а контейнерам нельзя предоставлять возможность монтирования критически важных файлов, подобных /etc/passwd, без явно определенных прав доступа к ним).

Best of LUCK with it, and remember to HAVE FUN
while you're learning :)



УПРАЖНЕНИЯ

Упражнение 1

Создание образов из файлов Dockerfile

Сначала создадим новый каталог и в нем Dockerfile:

```
$ mkdir cowsay
$ cd cowsay
$ touch Dockerfile
```

При работе с WSL могут возникнуть трудности с заполнением докер-файла. В WSL/Linux используется предустановлены текстовые терминалы nano или vim. Работа с ними требует филигранной техники управления клавиатурой, технология «copypast» здесь не работает. Чтобы упростить себе жизнь используйте такой «life hack». В консоли WSL наберите команду:

```
$ notepad.exe Dockerfile
```

И для вас откроется текстовый редактор для Windos – «Блокнот». В нем привычно редактируйте, копируйте, вставляйте.

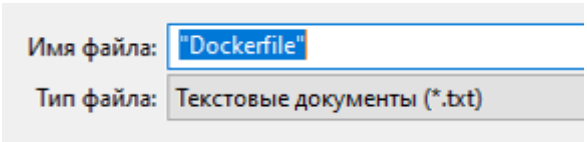
В созданный Dockerfile добавим следующее содержимое:

```
FROM debian:wheezy
RUN apt-get update && apt-get install -y cowsay fortune
```

Это упражнение предполагает использование Unix Docker-образа. Вы, конечно, можете использовать и Windows Docker-образ, но он работает медленнее, менее удобный и, вообще, его не часто применяют. Так что, пользуйтесь Unix по возможности.

Как создать “Dockerfile” в Windows?

Dockerfile должен быть создан без расширения. Чтобы сделать это в Windows, создайте файл в выбранном вами редакторе, а затем сохраните его с обозначением «Dockerfile» (включая кавычки).



Инструкция FROM определяет базовый образ ОС (здесь это, debian, но сейчас мы уточнили, что необходимо воспользоваться конкретной версией «wheezy»). Инструкция FROM является строго обязательной для всех файлов Dockerfile как самая первая незакомментированная инструкция. Инструкции RUN определяют команды, выполняемые в командной оболочке внутри данного образа. В нашем случае это команда установки пакетов cowsay и fortune, которая ранее была выполнена вручную.

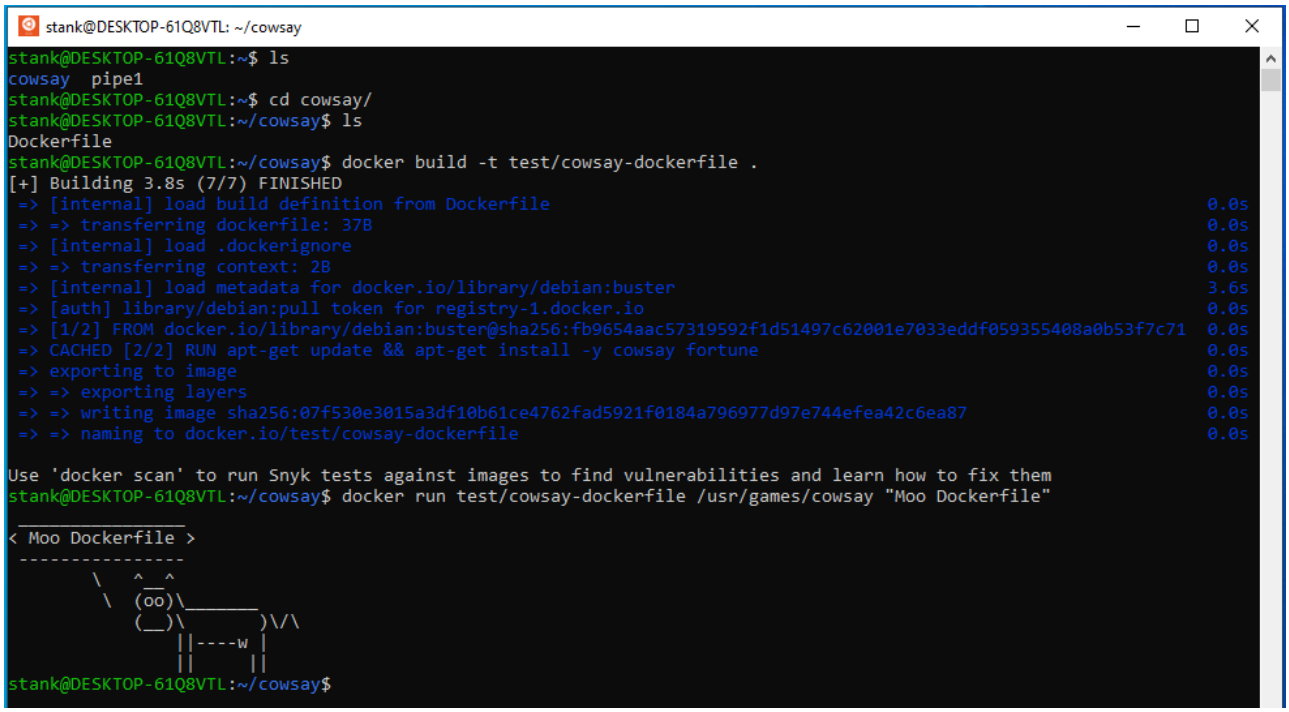
Теперь мы можем создать образ, выполнив команду docker build в том же каталоге, где расположен наш Dockerfile:

```
$ docker build -t test/cowsay-dockerfile .
```

Здесь *обратите внимание на точку*, которая стоит в конце строки. Скриптовая техника написания кода требует особого внимания на такие мелочи.

После этого мы можем запускать образ точно таким же способом, как раньше:

```
$ docker run test/cowsay-dockerfile /usr/games/cowsay "Moo"
```



На рисунке 1 представлена схема процесса, который мы выполнили по созданию образа с помощью Dockerfile.

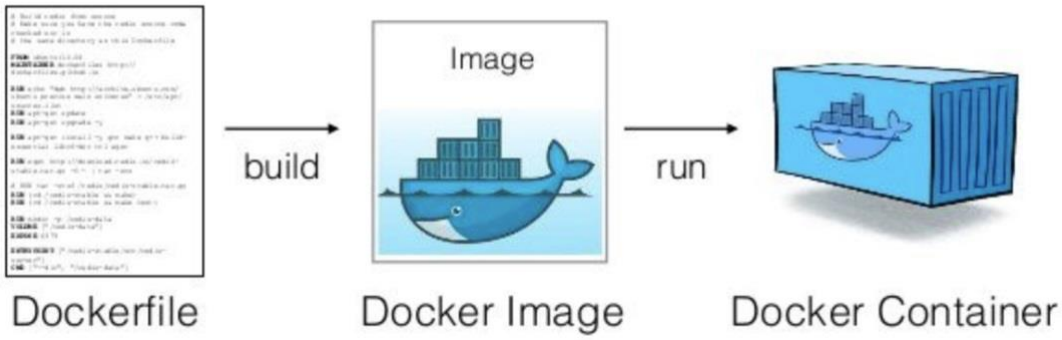


Рисунок 1 – Процесс создания образа с помощью Dockerfile

Выполнение этой задачи можно немного упростить, воспользовавшись преимуществами инструкции для файлов Dockerfile ENTRYPOINT. Эта инструкция позволяет определить выполняемый файл, который будет вызываться для обработки любых аргументов, переданных в команду docker run.

Добавим в конец нашего Dockerfile следующую строку:

```
ENTRYPOINT ["/usr/games/cowsay"]
```

Теперь нужно заново создать образ, и можно запускать его без указания команды cowsay (не забудьте точку):

```
$ docker build -t test/cowsay-dockerfile .
...
$ docker run test/cowsay-dockerfile "Moo"
...
```

Запуск стал проще. Но при этом мы лишились возможности использования внутри контейнера команды fortune в качестве генератора потока ввода для программы cowsay. Это можно исправить, если написать специальный скрипт для инструкции ENTRYPOINT – обычное дело при создании Dockerfile. В том же каталоге, где расположен наш Dockerfile, создадим скриптовый файл с именем entrypoint.sh и введем в него следующее содержимое:

```
#!/bin/bash
if [ $# -eq 0 ]; then
        /usr/games/fortune | /usr/games/cowsay
    else
        /usr/games/cowsay "$@"
fi
```

После сохранения необходимо сделать этот файл сценария оболочки shell (bash) выполняемым при помощи команды **chmod +x entrypoint.sh**.

```
stank@DESKTOP-61Q8VTL:~/cowsay$ notepad.exe entrypoint.sh
stank@DESKTOP-61Q8VTL:~/cowsay$ ls
Dockerfile  entrypoint.sh
stank@DESKTOP-61Q8VTL:~/cowsay$ chmod +x entrypoint.sh
stank@DESKTOP-61Q8VTL:~/cowsay$ ls -l
total 8
-rw-r--r-- 1 stank stank 116 Aug 20 09:21 Dockerfile
-rwxr-xr-x 1 stank stank 116 Aug 20 09:36 entrypoint.sh
stank@DESKTOP-61Q8VTL:~/cowsay$
```

Скриптовый файл стал исполняемый, здесь об этом говорит зеленый цвет имени файла и измененные атрибуты файла rwx.

При вызове без аргументов скрипт создает программный канал для передачи потока вывода программы fortune в поток ввода программы cowsay, в противном случае вызывается программа cowsay с передачей ей заданных аргументов.

Теперь необходимо изменить Dockerfile: добавить только что созданный файл в образ и вызвать скрипт в инструкции ENTRYPOINT.

Отредактированный Dockerfile должен выглядеть так:

```
FROM debian
RUN apt-get update && apt-get install -y cowsay fortune
COPY entrypoint.sh /
ENTRYPOINT ["/enrtypoint.sh"]
```

Инструкция COPY просто копирует файл из файловой системы хоста в файловую систему образа, где первый аргумент определяет файл хоста, а второй – целевой путь, таким образом, эта инструкция очень похожа на обычную команду **cp**.

```
$ docker build -t test/cowsay-dockerfile .
...
$ docker run test/cowsay-dockerfile
```

Упражнение 2
Работа с реестрами

Для выгрузки в реестр любого созданного вами образа необходимо зарегистрироваться (создать учетную запись) в реестре Docker Hub (в онлайн-режиме на сайте или с помощью команды docker login). После этого достаточно связать образ с соответствующим репозиторием и выполнить команду docker push для выгрузки помеченного образа в Docker Hub.

Но сначала добавим в Dockerfile инструкцию MAINTAINER, которая просто определяет информацию, позволяющую связаться с автором данного образа:

```
FROM debian
MAINTAINER John Smith <john@smith.com>
RUN apt-get update && apt-get install -y cowsay fortune
COPY entrypoint.sh /
ENTRYPOINT ["/entrypoint.sh"]
```

Теперь заново сгенерируем образ и выгрузим его в Docker Hub. В этот раз нужно будет использовать имя репозитория, начинающееся с вашего пользовательского имени в Docker Hub (в случае примера это amouat), за которым следуют слеш (/) и имя выгружаемого образа. Например:

```
$ docker build -t amouat/cowsay .
...
$ docker push amouat/cowsay
The push refers to a repository [docker.io/amouat/cowsay] (len: 1)
(Команда push выполняется для репозитория [docker.io/amouat/cowsay] (длина: 1))
e8728c722290: Image successfully pushed (Образ выгружен успешно)
5427ac510fe6: Image successfully pushed
...
latest: digest: sha256:bfd17b7c5977520211cecb202ad73c3ca14acde6878d9
ffc81d95...
```

Так как не указан тег после имени репозитория, образу автоматически присвоен тег latest. Чтобы определить свой тег, необходимо добавить его после имени репозитория и символа двоеточия (например, docker build -t amouat/cowsay:stable).

После успешного окончания выгрузки любой пользователь может загрузить этот образ командой docker pull amouat/cowsay.

Упражнение 3
Использование официального образа Redis

Из *официального* репозитория Docker, возьмем образ для Redis, широко распространенной системы хранения данных в виде пар «ключ-значение». Загрузим образ с помощью консоли:

```
$ docker pull redis
```

Запустим контейнер Redis, но при этом используем аргумент -d:

```
$ docker run --name myredis -d redis
```

Аргумент -d сообщает Docker, что контейнер нужно запустить в фоновом режиме. Docker начинает работу контейнера как обычно, но вместо вывода результатов из контейнера возвращает только его идентификатор и выполняет выход. Контейнер продолжает работу в фоновом режиме, а чтобы увидеть вывод результатов из контейнера, можно воспользоваться командой docker logs.

Необходимо каким-то образом установить связь с базой данных. Для этого используем инструмент командной строки redis-cli. Его можно установить прямо на хост, но проще, и полезнее в учебных целях создать новый контейнер для запуска в нем redis-cli и *установить соединение* между двумя контейнерами:

```
stank@DESKTOP-61Q8VTL:~/cowsay$ docker run --rm -it --link myredis:redis redis /bin/bash
root@187686105b2c:/data# redis-cli -h redis -p 6379
redis:6379> ping
PONG
redis:6379> set "abc" 123
OK
redis:6379> get "abc"
"123"
redis:6379> exit
root@187686105b2c:/data# exit
exit
stank@DESKTOP-61Q8VTL:~/cowsay$
```

Итак, мы только что установили соединение между двумя контейнерами и добавили данные в СУБД Redis буквально за несколько секунд.

Установление соединения определяется аргументом --link myredis:redis в команде docker run. Docker получает информацию о том, что нам нужно установить соединение между новым контейнером и существующим контейнером myredis, и в новом контейнере ссылка на существующий должна быть обозначена именем redis. Для этого Docker создает в файле нового контейнера /etc/hosts запись redis, указывающую на IP-адрес контейнера myredis. Это позволяет нам пользоваться именем хоста redis непосредственно в командной строке redis-cli, без дополнительного определения пути к нему или поиска IP-адреса контейнера Redis.

После этого выполняется команда СУБД Redis ping для проверки правильности установленного соединения с Redis-сервером перед добавлением и извлечением каких-либо данных с помощью команд set и put.

Создадим резервные копии содержимого контейнера Redis (предполагается, что контейнер myredis продолжает работу):

```
$ docker run --rm -it --link myredis:redis redis /bin/bash
root@a1c4abf81f:/data# redis-cli -h redis -p 6379
redis:6379> set "persistence" "test"
OK
redis:6379> save
```

```
OK
redis:6379> exit
root@a1c4abf81f:/data# exit
exit
$ docker run --rm --volumes-from myredis -v $(pwd)/backup:/backup debian cp
        /data/dump.rdb /backup/
$ ls backup
dump.rdb
```

Обратите внимание, аргумент -v использован для монтирования известного нам существующего каталога хоста, а аргумент --volumes-from – для установления соединения между новым контейнером и каталогом базы данных Redis.

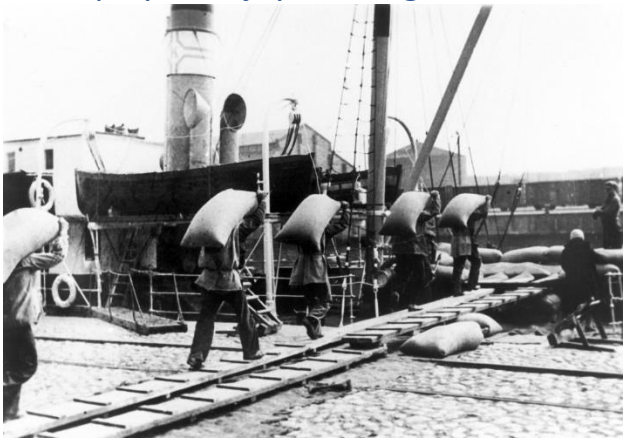
После завершения работы с контейнером myredis можно остановить и удалить его:

```
stank@DESKTOP-61Q8VTL:~/cowsay$ docker stop myredis
myredis
stank@DESKTOP-61Q8VTL:~/cowsay$ docker rm -v myredis
myredis
```

Все оставшиеся вспомогательные контейнеры можно удалить с помощью команды:

```
stank@DESKTOP-61Q8VTL:~/cowsay$ docker rm $(docker ps -aq)
1f7aa1feaccc
3bb3eee7afe2
d383904f0687
c9a860f3dd7b
90a6ebeb830f
cdb4519d09c3
8dc15baa734e
2884b491e929
432649c8b3ee
ca62135de59d
d4e49f8e041e
eb6562aedc27
2f698dd14b52
stank@DESKTOP-61Q8VTL:~/cowsay$
```

We hope you enjoy working with Docker!



ЗАДАНИЯ

Задание 1

Выполните упражнение №1 по созданию образа из Dockerfile. Вместо приложения cowsay может быть любое выбранное вами приложение. Уникальность приветствуется!

Задание 2

Зарегистрируйтесь, создайте собственную учетную запись в реестре Docker Hub. Не забудте пароль и тем более имя своей учетной записи! С помощью Dockerfile выгрузите и загрузите в репозиторий созданный вами образ.

Задание 3

Выполните упражнение №3. Дайте информацию, что такое Redis. В качестве пары «ключ-значение» используйте фамилии членов вашей команды и номера из телефонов (можно не настоящие). Результат подтвердите скрншотом.

«Easy things should be easy and hard things should be possible»
«Простые вещи должны быть простыми, а сложные вещи должны быть возможными»

Контрольные вопросы:

1) Что такое официальные репозитории Docker ?
2) Что такое реестры, репозитории, образы и теги?
3) Какое существует соглашение по определию имен для образов, и какие пространства имен для хранения образов вы знаете?
4) Как осуществляется хранение данных и создание резервных копий в Docker?
5) Что такое файловая система UnionFS и что такое многослойность контейнера?

ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ

Литература

Моуэт Э. **Использование Docker** / пер. с англ. А.В. Снастина; науч. ред. А.А. Маркелов. – М.: ДМК Пресс, 2017. – 354 с.: ил. [ ]

Иан Милл, Эйдан Хобсон Сейерс **Docker на практике** / пер. с англ. Д.А. Беликов. – М.: ДМК Пресс, 2020. – 516 с.: ил. [ ]

Шоттс У. Ш80 **Командная строка Linux. Полное руководство**. — СПб.: Питер, 2017. — 480 с.: ил. — (Серия «Для профессионалов»). [ ]

Интернет-источники

https://tproger.ru/translations/docker-instuction/
Tproger Основные инструкции Docker
Сергей Ринг 11 марта 2019 в 00:37 72962

https://dker.ru/docs/docker-engine/get-started-with-docker/build-your-own-image/ - dker.ru: Шаг 1: Создание Dockerfile. Шаг 2: Создание образа из Dockerfile. Шаг 3: Узнайте больше о процессе сборки. Шаг 4: Запуск нового образа.

Хранение данных в Docker / Хабр (habr.com)
Rekken
23 января 2021 в 10:14 Хранение данных в Docker

https://cloud.yandex.ru/docs/container-registry/concepts/docker-image Yandex Container Registry