Оглавление

1 Указания к заданию 2. Формирование и управление контейнерами в публичных облаках.

Цель: изучить принципы работы платформы Docker, методы создания контейнеров и развертывания контейнеров в публичных облаках.

Критерии оценивания

На локальном компьютере должна быть развернута инфраструктура Docker
В рамках развернутой инфраструктуры Docker, должно быть развернут контейнер с тестовым веб-приложением, к которому должен быть обеспечен доступ с локального компьютера.
В среде AWS необходимо развернуть Docker-контейнер с тестовым веб-приложением. Необходимо продемонстрировать публичную доступность и процесс работы веб-приложения в среде AWS.

Методические указания

1.1 Что такое Докер?

Определение Докера в Википедии звучит так:

программное обеспечение для автоматизации развёртывания и управления приложениями в среде виртуализации на уровне операционной системы; позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, а также предоставляет среду по управлению контейнерами.

Докер это инструмент, который позволяет разработчикам, системными администраторам и другим специалистам развертывать их приложения в песочнице (которые называются *контейнерами*), для запуска на целевой операционной системе, например, Linux. Ключевое преимущество Докера в том, что он позволяет пользователям **упаковать приложение со всеми его зависимостями в стандартизированный модуль** для разработки. В отличие от виртуальных машин, контейнеры не создают такой дополнительной нагрузки, поэтому с ними можно использовать систему и ресурсы более эффективно.

1.1.1 Что такое контейнер?

Стандарт в индустрии на сегодняшний день — это использовать виртуальные машины для запуска приложений. Виртуальные машины запускают приложения внутри гостевой операционной системы, которая работает на виртуальном железе основной операционной системы сервера.

Виртуальные машины отлично подходят для полной изоляции процесса для приложения: почти никакие проблемы основной операционной системы не могут повлиять на софт гостевой ОС, и наоборот. Но за такую изоляцию приходится платить. Существует

значительная вычислительная нагрузка, необходимая для виртуализации железа гостевой ОС.

Контейнеры используют другой подход: они предоставляют схожий с виртуальными машинами уровень изоляции, но благодаря правильному задействованию низкоуровневых механизмов основной операционной системы делают это с в разы меньшей нагрузкой.

1.1.2 Почему я должен использовать контейнеры?

Взлет Докера был по-настоящему эпичным. Несмотря на то, что контейнеры сами по себе — не новая технология, до Докера они не были так распространены и популярны. Докер изменил ситуацию, предоставив стандартный API, который сильно упростил создание и использование контейнеров, и позволил сообществу вместе работать над библиотеками по работе с контейнерами. В статье, опубликованной в The Register в середине 2014 говорится, что Гугл поддерживает больше **двух миллиардов контейнеров в неделю**.

В дополнение к продолжительному росту Докера, компания-разработчик Docker Inc. была оценена в два с лишним миллиарда долларов! Благодаря преимуществам в эффективности и портативности, Докер начал получать все больше поддержки, и сейчас стоит во главе движения по **контейнеризации** (containerization). Как современные разработчики, мы должны понять этот тренд и выяснить, какую пользу мы можем получить из него.

1.1.3 Чему меня научит эта лабораторная работа?

В рамках этой лабораторной работы вы получите опыт по сборке и развертыванию веб-приложений в облаке. Мы будем использовать Amazon Web Services для развертывания контейнеров с веб-приложениями.

1.1.4 Пре-реквизиты

Все, что нужно для прохождения этого пособия — это базовые навыки с командной строкой и текстовым редактором. Опыт разработки веб-приложений будет полезен, но не обязателен. В течение работы мы столкнемся с несколькими облачными сервисами. Вам понадобится создать аккаунт на этих сайтах:

- Amazon Web Services
- Docker Hub

1.2 Настройка компьютера

1.2.1 Докер

Еще несколько релизов назад запуск Докера на OS X и Windows был был проблемным. Но команда разработчиков проделала огромную работу, и сегодня весь процесс — проще некуда. Подробные инструкции по установке на Мак, Linux и Windows.

Проверим, все ли установлено корректно:

```
$ docker run hello-world

Hello from Docker.
This message shows that your installation appears to be working correctly.
...
```

1.2.2 Python

Python обычно предустановлен на OS X и на большинстве дистрибутивов Linux. Если вам нужно установить Питон, то скачайте установщик здесь.

Проверьте версию:

```
$ python --version
Python 2.7.11
```

Мы будем использовать pip для установки пакетов для нашего приложения. Если pip не установлен, то скачайте версию для своей системы.

Для проверки запустите такую команду:

```
$ pip --version
pip 7.1.2 from /Library/Python/2.7/site-packages/pip-7.1.2-py2.7.egg (python 2.7)
```

1.2.3 Java (не обязательно)

Разрабатываемое нами приложение будет использовать Elasticsearch для хранения и поиска. Для локального запуска Elasticsearch вам понадобится Java. В этом пособии все будет запускаться внутри контейнера, так что локально не обязательно иметь Java. Если Java установлена, то команда java -version должна сгенерировать подобный вывод:

```
$ java -version
java version "1.8.0_60"
Java(TM) SE Runtime Environment (build 1.8.0_60-b27)
Java HotSpot(TM) 64-Bit Server VM (build 25.60-b23, mixed mode)
```

1.3 Играем с Busybox

Теперь, когда все необходимое установлено, пора взяться за работу. В этом разделе мы запустим контейнер Busybox на нашей системе и попробуем запустить docker run.

Для начала, запустите следующую команду:

```
$ docker pull busybox
```

Внимание: в зависимости от того, как вы устанавливали Докер на свою систему, возможно появление сообщения permission denied. Если вы на Маке, то удостоверьтесь, что движок Докер запущен. Если вы на Линуксе, то запустите эту команду с sudo. Или можете создать группу docker чтобы избавиться от этой проблемы.

Команда pull скачивает образ busybox из **регистра Докера** и сохраняет его локально. Можно использовать команду docker images, чтобы посмотреть список образов в системе.

```
$ docker images
REPOSITORY                TAG                 IMAGE ID            CREATED             VI
RTUAL SIZE
busybox                   latest              c51f86c28340        4 weeks ago         1.
109 MB
```

1.3.1 Docker Run

Отлично! Теперь давайте запустим Докер-**контейнер** с этим образом. Для этого используем волшебную команду docker run:

```
$ docker run busybox
$
```

Подождите, ничего не произошло! Это баг? Ну, нет. Под капотом произошло много всего. Докер-клиент нашел образ (в нашем случае, busybox), загрузил контейнер и запустил команду внутри этого контейнера. Мы сделали docker run busybox, но не указали никаких команд, так что контейнер загрузился, запустилась пустая команда и программа завершилась. Ну, да, как-то обидно, так что давайте сделаем что-то поинтереснее.

```
$ docker run busybox echo "hello from busybox"
hello from busybox
```

Ура, наконец-то какой-то вывод. В нашем случае клиент Докера послушно запустил команду echo внутри контейнера, а потом вышел из него. Вы, наверное, заметили, что все произошло очень быстро. А теперь представьте себе, как нужно загружать виртуальную машину, запускать в ней команду и выключать ее. Теперь ясно, почему говорят, что контейнеры быстрые!

Теперь давайте взглянем на команду docker ps. Она выводит на экран список всех запущенных контейнеров.

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
```

Контейнеров сейчас нет, поэтому выводится пустая строка. Не очень полезно, поэтому давайте запустим более полезный вариант: docker ps -a

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
305297d7a235        busybox             "uptime"            11 minutes ago      Exited
(0) 11 minutes ago                       distracted_goldstine
ff0a5c3750b9        busybox             "sh"                12 minutes ago      Exited
(0) 12 minutes ago                       elated_ramanujan
```

Теперь виден список всех контейнеров, которые мы запускали. В колонке STATUS можно заметить, что контейнеры завершили свою работу несколько минут назад.

Вам, наверное, интересно, как запустить больше одной команды в контейнере. Давайте попробуем:

```
$ docker run -it busybox sh
/ # ls
bin   dev   etc   home  proc  root  sys   tmp   usr   var
/ # uptime
 05:45:21 up  5:58,  0 users,  load average: 0.00, 0.01, 0.04
```

Команда run с флагом -it подключает интерактивный tty в контейнер. Теперь можно запускать сколько угодно много команд внутри. Попробуйте.

Опасно!: Если хочется острых ощущений, то можете попробовать rm -rf bin в контейнере. Но удостоверьтесь, что запускаете ее внутри контейнера, а **не снаружи**. Если сделаете это снаружи, на своем компьютере, то будет очень плохо, и команды вроде ls, echo перестанут работать. Когда внутри контейнера все перестанет работать, просто выйдете и запустите его заново командой docker run -it busybox sh. Докер создает новый контейнер при запуске, поэтому все заработает снова.

На этом захватывающий тур по возможностям команды docker run закончен. Скорее всего, вы будете использовать эту команду довольно часто. Так что важно, чтобы мы поняли как с ней обращаться. Чтобы узнать больше о run, используйте docker run --help, и увидите полный список поддерживаемых флагов. Скоро мы увидим еще несколько способов использования docker run.

Перед тем, как продолжать, давайте вкратце рассмотрим удаление контейнеров. Мы видели выше, что с помощью команды docker ps -a все еще можно увидеть остатки завершенных контейнеров. На протяжении этого пособия, вы будете запускать docker run несколько раз, и оставшиеся, бездомные контейнеры будут съедать дисковое пространство. Так что я взял за правило удалять контейнеры после завершения работы с ними. Для этого используется команда docker rm. Просто скопируйте ID (можно несколько) из вывода выше и передайте параметрами в команду.

```
$ docker rm 305297d7a235 ff0a5c3750b9
305297d7a235
ff0a5c3750b9
```

При удалении идентификаторы будут снова выведены на экран. Если нужно удалить много контейнеров, то вместо ручного копирования и вставления можно сделать так:

```
$ docker rm $(docker ps -a -q -f status=exited)
```

Эта команда удаляет все контейнеры, у которых статус exited. Флаг -q возвращает только численные ID, а флаг -f фильтрует вывод на основе предоставленных условий. Последняя полезная деталь — команде docker run можно передать флаг --rm, тогда контейнер будет автоматически удаляться при завершении. Это очень полезно для разовых запусков и экспериментов с Докером.

Также можно удалять ненужные образы командой docker rmi.

1.3.2 Терминология

В предыдущем разделе мы использовали много специфичного для Докера жаргона, и многих это может запутать. Перед тем, как продолжать, давайте разберем некоторые термины, которые часто используются в экосистеме Докера.

- *Images* (образы) - Схемы нашего приложения, которые являются основой контейнеров. В примере выше мы использовали команду docker pull чтобы скачать образ **busybox**.
- *Containers* (контейнеры) - Создаются на основе образа и запускают само приложение. Мы создали контейнер командой docker run, и использовали образ busybox, скачанный ранее. Список запущенных контейнеров можно увидеть с помощью команды docker ps.
- *Docker Daemon* (демон Докера) - Фоновый сервис, запущенный на хост-машине, который отвечает за создание, запуск и уничтожение Докер-контейнеров. Демон — это процесс, который запущен на операционной системе, с которой взаимодействует клиент.

- *Docker Client* (клиент Докера) - Утилита командной строки, которая позволяет пользователю взаимодействовать с демоном. Существуют другие формы клиента, например, Kitematic, с графическим интерфейсом.
- *Docker Hub* - Регистр Докер-образов. Грубо говоря, архив всех доступных образов. Если нужно, то можно содержать собственный регистр и использовать его для получения образов.

1.4 Веб-приложения и Докер

Супер! Теперь мы научились работать с docker run, поиграли с несколькими контейнерами и разобрались в терминологии. Вооруженные этими знаниями, мы готовы переходить к реальным штукам: деплою веб-приложений с Докером!

1.4.1 Статические сайты

Давайте начнем с малого. Вначале рассмотрим самый простой статический веб-сайт. Скачаем образ из Docker Hub, запустим контейнер и посмотрим, насколько легко будет запустить веб-сервер.

Поехали. Для одностраничного сайта нам понадобится образ, который был заранее создан для этого пособия и размещен в регистре - prakhar1989/static-site. Можно скачать образ напрямую командой docker run.

```
$ docker run prakhar1989/static-site
```

Так как образа не существует локально, клиент сначала скачает образ из регистра, а потом запустит его. Если все без проблем, то вы увидите сообщение Nginx is running... в терминале. Теперь сервер запущен. Как увидеть сайт в действии? На каком порту работает сервер? И, что самое важное, как напрямую достучаться до контейнера из хост-контейнера?

В нашем случае клиент не открывает никакие порты, так что нужно будет перезапустить команду docker run чтобы сделать порты публичными. Заодно давайте сделаем так, чтобы терминал не был прикреплен к запущенному контейнеру. В таком случае можно будет спокойно закрыть терминал, а контейнер продолжит работу. Это называется **detached** mode.

```
$ docker run -d -P --name static-site prakhar1989/static-site
e61d12292d69556eabe2a44c16cbd54486b2527e2ce4f95438e504afb7b02810
```

Флаг -d открепит (detach) терминал, флаг -P сделает все открытые порты публичными и случайными, и, наконец, флаг --name это имя, которое мы хотим дать контейнеру. Теперь можно увидеть порты с помощью команды docker port [CONTAINER].

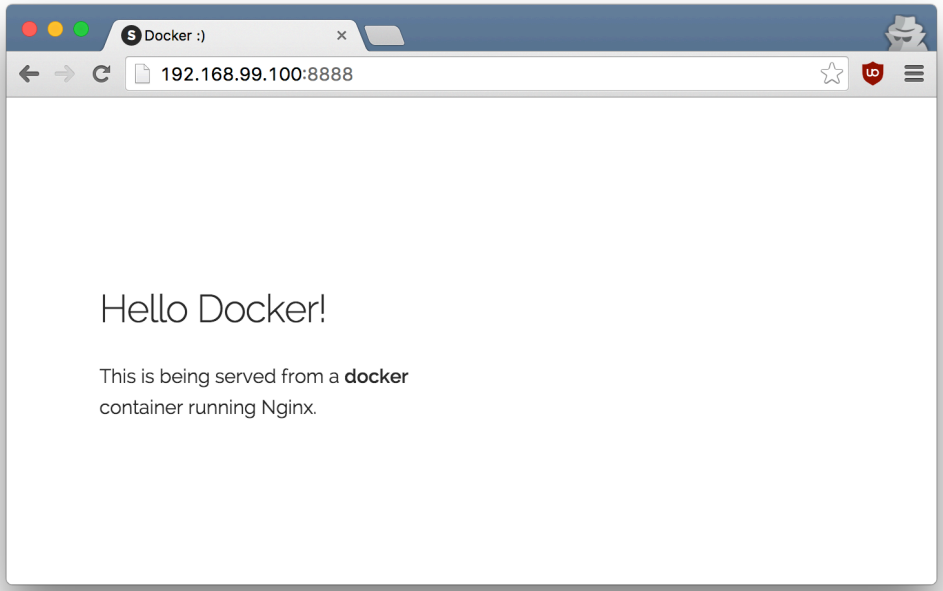
```
$ docker port static-site
80/tcp -> 0.0.0.0:32769
443/tcp -> 0.0.0.0:32768
```

Откройте http://localhost:32769 в своем браузере.

Замечание: Если вы используете docker-toolbox, то, возможно, нужно будет использовать docker-machine ip default чтобы получить IP-адрес.

Также можете обозначить свой порт. Клиент будет перенаправлять соединения на него.

```
$ docker run -p 8888:80 prakhar1989/static-site
Nginx is running...
```



Чтобы остановить контейнер запустите docker stop и укажите идентификатор (ID) контейнера.

Согласитесь, все было очень просто. Чтобы задеплоить это на реальный сервер, нужно просто установить Докер и запустить команду выше. Теперь, когда вы увидели, как запускать веб-сервер внутри образа, вам, наверное, интересно — а как создать свой Докер-образ? Мы будем изучать эту тему в следующем разделе.

1.4.2 Образы

Мы касались образов ранее, но в этом разделе мы заглянем глубже: что такое Докер-образы и как создавать собственные образы. Наконец, мы используем собственный образ чтобы запустить приложение локально, а потом задеплоим его на AWS, чтобы показать друзьям. Круто? Круто! Давайте начнем.

Образы это основы для контейнеров. В прошлом примере мы скачали (**pull**) образ под названием *Busybox* из регистра, и попросили клиент Докера запустить контейнер, **основанный** на этом образе. Чтобы увидеть список доступных локально образов, используйте команду docker images.

```
$ docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED
VIRTUAL SIZE
prakhar1989/catnip                  latest              c7ffb5626a50        2 hours ago
697.9 MB
prakhar1989/static-site             latest              b270625a1631        21 hours ago
133.9 MB
python                              3-onbuild           cf4002b2c383        5 days ago
688.8 MB
martin/docker-cleanup-volumes       latest              b42990daaca2        7 weeks ago
22.14 MB
ubuntu                              latest              e9ae3c220b23        7 weeks ago
187.9 MB
busybox                             latest              c51f86c28340        9 weeks ago
1.109 MB
hello-world                         latest              0a6ba66e537a        11 weeks ago
960 B
```

Это список образов, которые я скачал из регистра, а также тех, что я сделал сам (скоро увидим, как это делать). TAG — это конкретный снимок или снэпшот (snapshot) образа, а IMAGE ID — это соответствующий уникальный идентификатор образа.

Для простоты, можно относиться к образу как к git-репозиторию. Образы можно коммитить с изменениями, и можно иметь несколько версий. Если не указывать конкретную версию, то клиент по умолчанию использует latest. Например, можно скачать определенную версию образа ubuntu:

```
$ docker pull ubuntu:12.04
```

Чтобы получить новый Докер-образ, можно скачать его из регистра (такого, как Docker Hub) или создать собственный. На Docker Hub есть десятки тысяч образов. Можно искать напрямую из командной строки с помощью docker search.

Важно понимать разницу между базовыми и дочерними образами:

- **Base images** (базовые образы) — это образы, которые не имеют родительского образа. Обычно это образы с операционной системой, такие как ubuntu, busybox или debian.
- **Child images** (дочерние образы) — это образы, построенные на базовых образах и обладающие дополнительной функциональностью.

Существуют официальные и пользовательские образы, и любые из них могут быть базовыми и дочерними.

- **Официальные образы** — это образы, которые официально поддерживаются командой Docker. Обычно в их названии одно слово. В списке выше python, ubuntu, busybox и hello-world — базовые образы.
- **Пользовательские образы** — образы, созданные простыми пользователями вроде меня и вас. Они построены на базовых образах. Обычно, они называются по формату user/image-name.

1.4.3 Наш первый образ

Теперь, когда мы лучше понимаем, что такое образы и какие они бывают, самое время создать собственный образ. Цель этого раздела — создать образ с простым приложением на Flask. Для этого пособия я сделал маленькое приложение, которое выводит случайную гифку с кошкой. Ну, потому что, кто не любит кошек? Склонируйте этот репозиторий к себе на локальную машину.

Вначале давайте проверим, что приложение работает локально. Войдите в директорию flask-apprкомандой cd и установите зависимости.

```
$ cd flask-app
$ pip install -r requirements.txt
$ python app.py
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Если все хорошо, то вы увидите вывод как в примере выше. Зайдите на http://localhost:5000 чтобы увидеть приложение в действии.

Замечание: Если команда pip install падает с ошибками "permission denied", то попробуйте запустить ее с sudo. Если не хотите устанавливать пользовательские пакеты на уровне системы, то используйте команду pip install --user -r requirements.txt.

Выглядит отлично, правда? Теперь нужно создать образ с приложением. Как говорилось выше, все пользовательские образы основаны на базовом образе. Так как наше приложение написано на Питоне, нам нужен базовый образ Python 3. В частности, нам нужна версия python:3-onbuild базового образа с Питоном.

Что за версия onbuild, спросите вы?

Эти образы включают несколько триггеров ONBUILD, которых обычно достаточно чтобы быстро развернуть приложение. При сборке будет скопирован файл requirements.txt, будет запущен pip install с этим файлом, а потом текущая директория будет скопирована в /usr/src/app.

Другими словами, версия onbuild включает хелперы, которые автоматизируют скучные процессы запуска приложения. Вместо того, чтобы вручную выполнять эти задачи (или писать скрипты), образы делают все за вас. Теперь у нас есть все ингредиенты для создания своего образа: работающее веб-приложение и базовый образ. Как это сделать? Ответ: использовать **Dockerfile**.

1.4.4 Dockerfile

Dockerfile — это простой текстовый файл, в котором содержится список команд Докер-клиента. Это простой способ автоматизировать процесс создания образа. Самое классное, что команды в Dockerfile *почти* идентичны своим аналогам в Linux. Это значит, что в принципе не нужно изучать никакой новый синтаксис чтобы начать работать с докерфайлами.

В директории с приложением есть Dockerfile, но так как мы делаем все впервые, нам нужно создать его с нуля. Создайте новый пустой файл в любимом текстовом редакторе, и сохраните его в **той же** директории, где находится flask-приложение. Назовите файл Dockerfile.

Для начала укажем базовый образ. Для этого нужно использовать ключевое слово FROM.

```
FROM python:3-onbuild
```

Дальше обычно указывают команды для копирования файлов и установки зависимостей. Но к счастью, onbuild-версия базового образа берет эти задачи на себя. Дальше нам нужно указать порт, который следует открыть. Наше приложение работает на порту 5000, поэтому укажем его:

```
EXPOSE 5000
```

Последний шаг — указать команду для запуска приложения. Это просто python ./app.py. Для этого используем команду CMD:

```
CMD ["python", "./app.py"]
```

Главное предназначение CMD — это сообщить контейнеру какие команды нужно выполнить при старте. Теперь наш Dockerfile готов. Вот как он выглядит:

```
# our base image
FROM python:3-onbuild

# specify the port number the container should expose
EXPOSE 5000

# run the application
CMD ["python", "./app.py"]
```

Теперь можно создать образ. Команда docker build занимается сложной задачей создания образа на основе Dockerfile.

Листинг ниже демонстрирует процесс. Перед тем, как запустите команду сами (не забудьте точку в конце), проверьте, чтобы там был ваш username вместо моего. Username должен соответствовать тому, что использовался при регистрации на Docker hub. Если вы еще не регистрировались, то сделайте это до выполнения команды. Команда docker build довольно проста: она принимает опциональный тег с флагом -t и путь до директории, в которой лежит Dockerfile.

```
$ docker build -t prakhar1989/catnip .
Sending build context to Docker daemon 8.704 kB
```

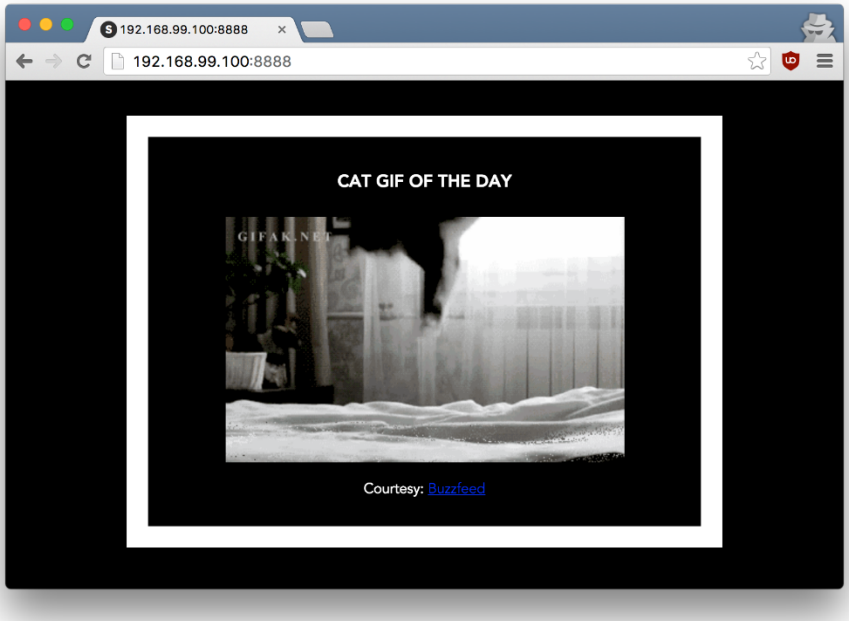
```
Step 1 : FROM python:3-onbuild
# Executing 3 build triggers...
Step 1 : COPY requirements.txt /usr/src/app/
 ---> Using cache
Step 1 : RUN pip install --no-cache-dir -r requirements.txt
 ---> Using cache
Step 1 : COPY . /usr/src/app
 ---> 1d61f639ef9e
Removing intermediate container 4de6ddf5528c
Step 2 : EXPOSE 5000
 ---> Running in 12cfcf6d67ee
 ---> f423c2f179d1
Removing intermediate container 12cfcf6d67ee
Step 3 : CMD python ./app.py
 ---> Running in f01401a5ace9
 ---> 13e87ed1fbc2
Removing intermediate container f01401a5ace9
Successfully built 13e87ed1fbc2
```

Если у вас нет образа python:3-onbuild, то клиент сначала скачает его, а потом возьмется за создание вашего образа. Так что, вывод на экран может отличаться от моего. Посмотрите внимательно, и найдете триггеры onbuild. Если все прошло хорошо, то образ готов! Запустите docker images и увидите свой образ в списке.

Последний шаг — запустить образ и проверить его работоспособность (замените username на свой):

```
$ docker run -p 8888:5000 prakhar1989/catnip
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Зайдите на указанный URL и увидите приложение в работе.



Поздравляю! Вы успешно создали свой первый образ Докера!

1.4.5 Docker на AWS

В этом разделе мы научимся развертывать и обеспечивать доступ к нашему приложению из облака. Будем использовать виртуальные машины AWS EC2.

1.4.5.1 Docker push

Первое, что нужно сделать перед деплоем на AWS это опубликовать наш образ в регистре, чтобы можно было скачивать его из AWS. Есть несколько Docker-регистров (или можно создать собственный). Для начала, давайте используем Docker Hub. Не забудьте заменить название образа на свое. Очень важно сохранить формат username/image_name, чтобы клиент понимал, куда публиковать образ. Просто выполните:

```
$ docker push user_name/catnip
```

Если это ваша первая публикация, то клиент попросит вас залогиниться. Введите те же данные, что используете для входа в Docker Hub.

```
$ docker login
Username: user_name
WARNING: login credentials saved in /Users/prakhar/.docker/config.json
Login Succeeded
```

После этого можете посмотреть на свой образ на Docker Hub. Например, вот страница моего образа.

Замечание: один важный момент, который стоит прояснить перед тем, как продолжить — **не обязательно** хранить образ в публичном регистре (или в любом другом регистре вообще) чтобы деплоить на AWS. Если вы пишете код для следующего многомиллионного стартапа-единорога, то можно пропустить этот шаг. Мы публикуем свой образ чтобы упростить деплой, пропустив несколько конфигурационных шагов.

Теперь наш образ онлайн, и любой докер-клиент может поиграться с ним с помощью простой команды:

```
$ docker run -p 8888:5000 prakhar1989/catnip
```

Если в прошлом вы мучались с установкой локального рабочего окружения и попытками поделиться своей конфигурацией с коллегами, то понимаете, как круто это звучит. Вот почему Докер — это сила!

1.4.5.2 Развертывание Docker на базе экземпляра ВМ в EC2

Для развертывания образа Docker на базе экземпляра виртуальной машины в Amazon EC2, необходимо подготовить и запустить образ виртуальной машины в среде Amazon EC2. Процесс запуска и организации доступа к удаленной виртуальной машине подробно рассмотрен в предыдущей лабораторной работе.

После подключения к удаленной виртуальной машине, обновите пакеты при помощи команды

```
sudo yum update -y
```

Установка последней версии Docker на Amazon Linux обеспечивается следующей командой:

```
sudo amazon-linux-extras install docker
```

Запустите сервис Docker

```
sudo service docker start
```

Для того, чтобы использовать Docker без необходимости использования sudo необходимо добавить пользователя `ec2-user` в группу `docker` .

```
sudo usermod -a -G docker ec2-user
```

Выйдете из системы и зарегистрируйтесь заново для получения прав группы `docker` . Вы можете просто закрыть окно вашего SSH клиента и подключиться к консоли заново. Новая сессия SSH будет иметь необходимые для запуска права группы `docker`.

Проверьте, что пользователь `ec2-user` может выполнять команды без `sudo`.

```
docker info
```

Если сервис Docker не запущен, и на выполнение этой команды возвращается ошибка, то можно запустить сервис Docker вручную с помощью команды

```
sudo service docker start
```

После того, как Docker успешно запущен в виртуальной машине, можно запускать наше приложение. Это можно сделать с помощью команды (не забудьте заменить user_name на ваше имя пользователя в Docker Hub)

```
docker run -p 80:5000 user_name/catnip
```

Данная команда загрузит образ catnip из вашего репозитория в Docker Hub, развернет его на машине и сделает пор 5000 доступным по 80 порту вашей виртуальной машины AWS. Если в сетевых настройках доступа к вашей машине прописана возможность доступа по 80 порту, то теперь вы можете зайти на ip-адрес (или URL) вашей виртуальной машины в браузере и увидите приложение во все красе. Пошлите адрес своим друзьям, чтобы все могли насладиться гифками с кошками.