edicas 6 сентября 2017 в 15:49

Основы Docker за X часов и Y дней

Виртуализация, DevOps

Из песочницы

0. Вступление

Цель данной статьи собрать в небольшую кучку основную информацию, минимально достат
с докер на ежедневной основе и удалить с рабочей машины локально установленные apach
memchaced, redis, php5, php7 и весь остальной зоопарк, который мы используем при разраб
конфликтует между собой от версии к версии.

А еще я в автобусе и ближайшие 7 часов мне все равно делать будет нечего. Ну и вдобавок
ссылки и команды, за которыми мне самому периодически приходится лезть в документаци
алиас к локалхосту: `sudo ifconfig lo0 alias 10.200.10.1/24` (зачем это надо будет сказан

Статья называется так, как называется, потому что я не смог более-менее точно подсчитать
поставленную задачу. У меня на это ушло примерно 6 часов в течение 3 дней. Во многом, по
тайм в офисе и осваивал докер, так сказать, без отрыва от производства.

У вас может уйти меньше дней или больше часов, зависит от того, как интенсивно вы будете

Но мое личное мнение – лучше все же не пытаться “ворваться” за один день. Просто потому
дело, то в конце первого дня у вас будет где-то вот такая голова

И лучше в этот момент остановиться, переварить информацию, может даже поспать.

1. Теория

Если Вы ранее имели дело с виртуальными машинами и такими инструментами как virtualbox, vmware, vagrant и подобными штуками – лучше забудьте о них.
Лично моей ошибкой была попытка работать с докер как с виртуальной машиной. Докер – средство виртуализации процессов, а не систем. Важное правило – каждому процессу свой виртуальный контейнер.

Контейнер следует воспринимать как отдельный процесс и наоборот. Например не следует пихать в один контейнер mysql и redis. Или еще хуже всю связку apache+php+mysql.

Основные термины

Image (образ) – собранная подсистема, необходимая для работы процесса, сохраненная в образе.
Container (контейнер) – процесс, инициализированный на базе образа. То есть контейнер существует только когда запущен. Это как экземпляр класса, а образ это типа класс. Ну я думаю идея понятна.
Host (хост) – среда, в которой запускается докер. Проще говоря – ваша локальная машина.
Volume – это дисковое пространство между хостом и контейнером. Проще – это папка на вашей локальной машине примонтированная внутрь контейнера. Меняете тут меняется там, и наоборот, миракл.
Dockerfile – файл с набором инструкций для создания образа будущего контейнера
Service (сервис) – по сути это запущенный образ (один или несколько контейнеров), дополнительно сконфигурированный такими опциями как открытие портов, маппинг папок (volume) и прочее. Обычно это делается при помощи docker-compose.yml файла.
Docker-compose (докер-композ, чаще композер, но не путать с php composer) – тулза, облегчающая сборку и запуск системы состоящей из нескольких контейнеров, связанных между собой.
Build (билд, билдить) – процесс создания образа из набора инструкций в докерфайле, или нескольких докерфайлов, если билд делается с помощью композера
В данной статье позже (завтра) я опишу процесс сборки связки nginx+mysql+php7-fpm с примерами и описаниями dockerfile и docker-compose файлов.

Вкратце о том, как работает билдинг образов

Сначала есть docker hub, это такой репозиторий, куда все желающие самоутвердиться публикуют свои собственные сборки образов. За это некоторым большое спасибо, а некоторым нет, все как и в любой другой свалке пакетов.

Обычно докерфайл начинается с инструкции **FROM**, которая указывает с какого пакета/образа из хаба начать.

Далее обычно идет инструкция maintainer, задача которой увековечить имя создателя очередного гениального творения.

Затем наиболее часто встречающиеся команды:
RUN – выполняет команду внутри образа.
ADD – берет файлы с хоста и кладет внутрь образа.
А также COPY, EXPOSE, ENTRYPOINT, CMD — обо всем этом Вы узнаете в процессе.

Сейчас внимание. Докер выполняет инструкции из докерфайла последовательно поверх предыдущего результата. Таким образом организовано хранение кеша.
Не поняли? Сейчас покажу. Вот простейший докерфайл:

```
FROM ubuntu:latest
MAINTAINER igor
RUN apt-get update
RUN apt-get install nginx
ADD ./nginx.conf /etc/nginx/
EXPOSE 80
CMD [nginx]
```

Как докер его билдит:

1. качаем образ ubuntu с тегом latest, сохраняем его с ID=aaa
2. берем образ aaa, прописываем ему maintainer=igor, сохраняем его с ID=aab
3. берем образ aab, запускаем контейнер и выполняем внутри команду “apt-get update”, останавливаем контейнер, получившийся в результате образ сохраняем с ID=aac
4. берем образ aac, запускаем контейнер и выполняем внутри команду “apt-get install nginx”, останавливаем контейнер, получившийся в результате образ сохраняем с ID=aad
5. берем образ aad, запускаем контейнер и копируем файл ./nginx.conf (путь указывается относительно папки в которой находится dockerfile) внутрь контейнера по пути /etc/nginx/, останавливаем контейнер, получившийся в результате образ сохраняем с ID=aae
….

уже понятнее?

ID я тут написал условные, но важно помнить, что идентификаторы этих “промежуточных” образов напрямую связаны с самими инструкциями, с файлами которые добавляются инструкцией ADD и с ID родительского образа. То есть фактически перед выполнением каждого шага сначала вычисляется ID (хеш) образа, поиск такого ID в локальном кеше, и только если такого ID в кеше нет, тогда выполняется шаг и сохраняется в кеш, иначе – используется образ из кеша.

А также это значит, что если вы решите поменять команду например `run apt-get install nginx` на другую, то Хеш(ID) инструкции изменится и весь дальнейший кеш после этого использоваться не будет. Потому не удивляйтесь если после изменения одной буквы в имени maintainer’а у вас вся сборка будет пересобираться от самого начала.

Также исходя из описанного сценария выполнения команд становится понятно, почему не имеет смысла в инструкциях выполнять команды ничего не сохраняющие после своего выполнения – частый вопрос на stackoverflow — “я запустил что-то, а в следующей инструкции оно не запущено”. Например кто-то хочет активировать source env/bin/activate и в следующей инструкции выполнить pip install

```
RUN source /app/env/bin/activate
RUN pip install something
```

или другой пример – запустить монгодб и в следующей инструкции создать юзера/базу или выполнить импорт базы из файла (есть причины почему лучше так не делать, но сейчас не об этом)

```
RUN service mongodb start
RUN mongo db --eval 'db.createUser({user:"dbuser",pwd:"dbpass",roles:["readWrite","dbAdmin"]})'
```

Как видно из процесса сборки каждый следующий контейнер ничего не знает о том что там запускалось в предыдущем шаге, поэтому такие инструкции нужно объединять в одну через &&:

```
RUN source /app/env/bin/activate \
&& pip install something

RUN service mongodb start \
&& RUN mongo db --eval 'db.createUser({user:"",...})'
```

но вообще, в связи с тем, что контейнеры изолированы, я не вижу большого смысла в использовании внутри средств типа nvm, virtualenv, rbenv and similar stuff. Просто ставьте что нужно и все.

Думаю для начала работы этой теории вполне достаточно.

2. Практика

Перед началом думаю следует пойти немного передохнуть и сделать себе чайку.
А когда вернетесь, сначала установите себе Docker и Docker Compose.

Небольшое отступление для тех, кто читает это под виндой.

Вы серьезно?

Нет, это конечно же возможно и там на сайте есть установочные файлы и инструкции. И лично я ничего не имею против windows, когда речь о домашнем использовании в качестве медиастанции. Но если вы разрабатываете под виндой, то я одновременно и восхищаюсь вами и соболезную. Скажу откровенно, я сам под виндой докер не поднимал, только под убунту и под мак. Однако я слышал стоны коллеги из соседнего кабинета, который пытался, и у него даже получилось запустить сборку, но когда дело дошло до симлинок внутри volume, то все пошло прахом. Потому, если вы попали сюда в поиске рецептов, как работать с Docker под Windows, то у меня плохие новости, здесь вы их не найдете.

Итак, на данном этапе у вас уже установлен докер, и в панели задач радостно побулькивает блоками #синийкит (извините, не удержался). Теперь сходите вот по этой ссылке и пройдите туториал Get Started.

Теперь давайте представим, что мы разрабатываем веб сайт на php и будем его публиковать связкой nginx+php7-fpm+mysql.

Вот очень примитивный dockerfile для php сервиса:

```
FROM php:7-fpm
# Install modules
RUN apt-get update && apt-get install -y \
libfreetype6-dev \
libjpeg62-turbo-dev \
libmcrypt-dev \
libpng12-dev \
libicu-dev \
--no-install-recommends
RUN docker-php-ext-install mcrypt zip intl mbstring pdo_mysql exif \
&& docker-php-ext-configure gd --with-freetype-dir=/usr/include/ --with-jpeg-dir=/usr/include/ \
&& docker-php-ext-install gd

ADD ./php.ini /usr/local/etc/php/
ADD ./www.conf /usr/local/etc/php/

RUN apt-get purge -y g++ \
&& apt-get autoremove -y \
&& rm -r /var/lib/apt/lists/* \
&& rm -rf /tmp/*

EXPOSE 9000
CMD ["php-fpm"]
```

Вкратце человеческим языком:

- за основу берем образ php:7-fpm из хаба
- обновляем убунту, доставляем нужные нам php-extensions и инструменты, копируем (добавляем) конфиги php внутрь образа, удаляем все ненужное и открываем порт 9000, который будет слушать php-fpm
- единственная незнакомая деталь сейчас это конструкция ADD но она очевидна: взять файл согласно первого аргумента и положить внутрь контейнера по пути указанному во втором аргументе. В первом аргументе может быть абсолютный путь, относительный путь, http или ftp url, а также этот путь может указывать на папку или архив. В случае архива он будет распакован. Подробнее смотрите в документации ADD и COPY.

С php разобрались, теперь нам нужно обозначить образы для сервисов nginx и mysql, а также собрать все сервисы в целостную систему.

В случае с nginx и mysql нам даже не нужно писать свои dockerfile, так как никаких дополнительных расширений нам ставить не нужно. Вот как будет выглядеть docker-compose.yml нашего проекта

```
app:
build: docker/php/Dockerfile
working_dir: /app
volumes:
- ./:/app
expose:
- 9000
links:
- db
nginx:
image: nginx:latest
ports:
- "80:80"
volumes:
- ./:/app
- ./docker/nginx/vhost.conf:/etc/nginx/conf.d/vhost.conf
links:
- app
db:
image: mysql:5.7
volumes:
- /var/lib/mysql
ports:
- "3306:3306"
environment:
MYSQL_ROOT_PASSWORD: root
MYSQL_DATABASE: dbname
MYSQL_USER: dbuser
MYSQL_PASSWORD: dbpassword
```

Здесь объявлены сервисы app, nginx, db. app собирается из нашего докерфайла, а остальные просто используют образы из хаба.

Директива volumes монтирует папки из хостмашины внурть контейнера, таким образом осуществляется конфигурирование nginx и сохранение данных бд при перезапуске.
Директива links связывает сервисы между собой, app связан с db, это значит что после запуска внутри контейнера app будет доступен хост “db”, и он будет указывать на соответствующий контейнер.

Все просто (ирония).

Есть довольно интересный темплейт yii2-starter-kit, в коробке которого можно найти неплохую реализацию описанной сборки php7-fpm nginx mysql а также mailcatcher.

Тем кому, как и мне, больше по душе python и django можно вообще не париться и все сделать по официальному туториалу от Docker — docs.docker.com/compose/django
плюс, после того как уже пришло понимание как это все работает, не составит особого труда переработать любую понравившуюся

сборку под свои нужды.

Pitfalls

— MacOS. Доступ к сервису на хосте (к примеру mongo или mysql) из контейнера.
Из-за ограничений “Docker for Mac networking stack” нельзя “просто так взять и подключиться” к локалхост. Но есть два обходных пути:

а) официальный и простой (доступен в версии Docker начиная с 17.06) — использовать для подключения специальный DNS хост (доступно только в Docker for Mac) docker.for.mac.localhost. Источник.

б) добавить алиас IP к сетевому устройству lo0:
`sudo ifconfig lo0 alias 10.200.10.1/24`
и использовать этот адрес для подключения

— MongoDB. На маке нельзя монтировать внешний диск для данных. Причины описаны здесь
WARNING (Windows & OS X): The default Docker setup on Windows and OS X uses a VirtualBox VM to host the Docker daemon. Unfortunately, the mechanism VirtualBox uses to share folders between the host system and the Docker container is not compatible with the memory mapped files used by MongoDB

— Под Windows не работают симлинки в примонтированных томах. Ожидаемо, но очень неприятно узнавать об этом уже после того как все остальное заработало.

— Отличия entrypoint и command — вот тут подробно и понятно описана разница между entrypoint и command.

— **UPD** дополнение от @saskasa: На маке скорость записи из контейнера на диск хоста (добавленный как VOLUME) очень медленная, для понимания масштабов — примерно в 50-100 раз.

Теги: docker, docker-compose

Комментарии 16

Almer 6 сентября 2017 в 18:43 +1

> The default Docker setup on Windows and OS X uses a VirtualBox VM to host the Docker daemon.

Это было актуально во времена Docker Toolbox и уже давно не соответствует действительности: docs.docker.com/docker-for-mac/docker-toolbox.

С выходом нативного Docker for Mac на HyperKit никаких проблем не возникает.

ecto 6 сентября 2017 в 22:47 0

С сетевым стеком были проблемы несколько раз. Лучше чем Toolbox но не идеально. При рестарте проекта в композе внешние порты остались заняты для системы, и проект не подымался пока не перезагрузили машину.

saggid 6 сентября 2017 в 20:07 +1

Здравствуйте) Можно глупый вопрос, который меня давно гложит?

Вот подняли мы всё это чудо из нескольких связанных образов на продакшене. А потом у нас возникла необходимость обновить какой-то из образов. Как сделать всё это без остановки всего продакшена?

mogaika 6 сентября 2017 в 21:33 0

Сразу пришло в голову haproxy и скрипт, который будет аккуратно перезапускать контейнеры.
В случае с compose можно делать так (решение не сильно отличается): github.com/docker/compose/issues/734#issuecomment-167392770
Для себя закинул еще одно очко в копилку «docker — для удобной/быстрой разработки/тестирования, но не для продакшена»

Dreyk 6 сентября 2017 в 21:35 0

нормально крутил докер на продакшне, не надо страшилок

ну да, наш админ немного патчей там накатил, чтоб оно с Macvlan работало нормально, но это уже подробности =)

Dreyk 6 сентября 2017 в 21:34 +2

обычно на продакшне это все обмазано docker-swarm либо чем-то подобным (kubernetes и т.д.). В этом случае после обновления образа и пуша его в registry, на swarm-ноде делается update — и оно заменяет контейнеры один за другим с zero downtime

saskasa 11 сентября 2017 в 13:33 +2

Забыли в Pitfails добавить, что на маке скорость записи из контейнера на диск хоста (добавленный как VOLUME) очень медленная, для понимания масштабов — примерно в 50-100 раз.
Вот пример записи данных на 102MB (на диск контейнера — 0.3сек, на диск хоста — 44сек.):

```
root@cdb2179ff:/var/www/temp# time dd if=/dev/zero of=test.dat bs=1024 count=100000
102400000 bytes (102 MB, 98 MiB) copied, 44.3007 s, 2.3 MB/s
real    0m44.310s
user    0m0.080s
sys     0m3.060s

root@cdb2179ff:/tmp# time dd if=/dev/zero of=test.dat bs=1024 count=100000
102400000 bytes (102 MB, 98 MiB) copied, 0.375994 s, 272 MB/s
real    0m0.381s
user    0m0.030s
sys     0m0.340s
```

На официальном форуме развернулась огромная дискуссия на эту тему и судя по ней прогнозы так себе.
Поэтому разработка может стать неприятной когда работа идет с процессом связанным с записью данных из контейнера на диск хоста.
У меня, например, на одном проекте обновление страницы сайта запущенного в docker занимает около 1.5 секунды (из-за записи данных в mysql базу, которая лежит на хосте и монтируется через VOLUME), если тот же сайт запускать на локальном сервере, обновление страницы происходит моментально. Все это очень мешает при отладке когда часто приходится обновлять страницу в браузере.

edicas 11 сентября 2017 в 13:37 0

спасибо за дополнение, добавил в список

helpik94 11 сентября 2017 в 13:37 0

Кстати использование && еще и неплохо уменьшает размер докерфайла.

Akrillis 11 сентября 2017 в 13:37 0

Согласен с автором во всем, сам въезжал в docker примерно 3 дня с параллельными попытками написания Dockerfile для сборки нужного мне image. Однако советую сразу пристальное внимание обратить на инструкцию CMD, без правильного написания которой запущенный контейнер будет сразу останавливаться, а вместо deprecated MAINTAINER использовать все разнообразие инструкции LABEL.

Rastler 18 сентября 2017 в 15:08 0

Вторую часть дождёмся?