

1 ПРАКТИЧЕСКАЯ РАБОТА №3

1.1 Теоретическое введение

Docker — это программная платформа для разработки, доставки и запуска контейнерных приложений. Он позволяет создавать контейнеры, автоматизировать их запуск и развертывание, управляет жизненным циклом. С помощью Docker можно запускать множество контейнеров на одной хост-машине.

Контейнеризация — это способ упаковки приложения и всех его зависимостей в один образ, который запускается в изолированной среде, не влияющей на основную операционную систему.

Контейнер — базовая единица программного обеспечения, покрывающая код и все его зависимости для обеспечения запуска приложения прозрачно, быстро и надежно независимо от окружения. Контейнер Docker может быть создан с использованием образа Docker. Это исполняемый пакет программного обеспечения, содержащий все необходимое для запуска приложения, например, системные программы, библиотеки, код, среды исполнения и настройки.

Docker-образ — 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-образ состоит из слоев. Каждое изменение записывается в новый слой.

При загрузке или скачивании Docker-образа, операции производятся только с теми слоями, которые были изменены. Слои исходного Docker-образа являются общими между всеми его версиями и не дублируются.

Томы Docker — это способ создания постоянного хранилища для контейнеров Docker. Томы Docker не привязаны к времени жизни контейнера, поэтому сделанные в них записи не исчезнут, как это произойдет с контейнером.

Они также могут быть повторно подключены к одному или к нескольким контейнерам, чтобы можно было обмениваться данными и подключать новые контейнеры к существующему хранилищу. Тома Docker работают путем создания каталога на главной машине и последующего монтирования этого каталога в контейнер (или в несколько контейнеров). Этот каталог существует вне многослойного образа, который обычно содержит контейнер Docker, поэтому он не подчиняется тем же правилам (только для чтения и т. д.).

Часто используемые команды Docker:

- `docker push`: Закачать репозиторий или образ в Registry;
- `docker run`: Запустить команду в новом контейнере;
- `docker pull`: Скачать репозиторий или образ из Registry;
- `docker start`: Запустить один или несколько контейнеров;
- `docker stop`: Остановить один или несколько контейнеров;
- `docker search`: Поиск образа на DockerHub;
- `docker commit`: Сохранить изменения в новый образ.
- `docker -version`: узнать установленную версию Docker;
- `docker ps`: перечислить все запущенные контейнеры вместе с дополнительной информацией о них;
- `docker ps -a`: перечислить все контейнеры, включая остановленные, вместе с дополнительной информацией о них;
- `docker exec`: войти в контейнер и выполнить в нем команду;
- `docker build`: собрать образ из Dockerfile;
- `docker rm`: удалить контейнер с указанным идентификатором;
- `docker rmi`: удалить образ с указанным идентификатором;
- `docker info`: получить расширенную информацию об установленном Docker, например, сколько запущено контейнеров, образов, версию ядра, доступную оперативную память и т.п.;
- `docker cp`: сохранить файл из контейнера в локальную систему;
- `docker history`: показать историю образа с указанным именем.

Все возможные состояния контейнера Docker:

- Created — контейнер создан, но не активен.
- Restarting — контейнер в процессе перезапуска.
- Running — контейнер работает.
- Paused — контейнер приостановлен.
- Exited — контейнер закончил свою работу.
- Dead — контейнер, который сервис попытался остановить, но не

смог.

Dockerfile содержит инструкции для сборки образов, которые передаются в Docker. Также его можно описать как текстовый документ, содержащий все возможные команды, с помощью которых пользователь, последовательно их запуская, может собрать образ.

Swarm Mode — это встроенная система оркестровки Docker для масштабирования контейнеров в кластере физических машин. Несколько независимых клиентов, на которых работает Docker Engine, объединяют свои ресурсы, образуя рой.

Эта функция поставляется в комплекте с Docker и включает все необходимое для развертывания приложений на узлах. Swarm Mode имеет декларативную модель масштабирования, в которой вы указываете количество необходимых реплик. Менеджер роя принимает меры, чтобы сопоставить фактическое количество реплик вашему запросу, создавая и уничтожая контейнеры по мере необходимости.

1.2 Полезные ссылки

1 50 вопросов по Docker, которые задают на собеседованиях, и ответы на них | Хабр. — Текст: электронный [сайт]. — URL: <https://habr.com/ru/company/southbridge/blog/528206/>

2 Docker Documentation | Docker Documentation — Текст: электронный [сайт]. — URL: <https://docs.docker.com/>

3 Что такое режим Docker Swarm и когда его использовать? — CloudSavvy ИТ | Сраб. — Текст: электронный [сайт]. — URL: <https://cpab.ru/chtotakoe-rezhim-docker-swarm-i-kogda-ego-ispolzovat-cloudsavvy-it/>

4 Dockerfile reference | Docker Documentation — Текст: электронный [сайт]. — URL: <https://docs.docker.com/engine/reference/builder/>

1.3 Практическая часть

В этой практике рассматриваются рекомендуемые передовые практики и методы создания эффективных образов.

Docker автоматически создает образы, читая инструкции из Dockerfile - текстового файла, который содержит все команды в порядке, необходимом для создания данного образа. Dockerfile придерживается определенного формата и набора инструкций, которые вы можете найти в справочнике по Dockerfile .

Образ Docker состоит из слоев, доступных только для чтения, каждый из которых представляет инструкцию Dockerfile. Слои сложены, и каждый из них представляет собой дельту изменений по сравнению с предыдущим слоем. Рассмотрим этот Dockerfile на Рисунке 1.

```
# syntax=docker/dockerfile:1
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

Рисунок 1 — Dockerfile с Python проектом

Каждая инструкция создает один слой:

- FROM создает слой из ubuntu:18.04 образа Docker.
- COPY добавляет файлы из текущего каталога вашего клиента Docker.
- RUN создает ваше приложение с make.
- CMD указывает, какую команду запускать в контейнере.

1.3.1 Правило эфемерных контейнеров

Образ, определенный вашим Dockerfile, должен генерировать как можно более эфемерныe контейнеры. Под «эфемерным» подразумевается, что контейнер можно остановить и уничтожить, а затем перестроить и заменить с абсолютно минимальной настройкой и конфигурацией.

1.3.2 Понимание контекста сборки

Когда вы вводите `docker build` команду, текущий рабочий каталог называется контекстом сборки. По умолчанию предполагается, что Dockerfile находится здесь, но вы можете указать другое местоположение с помощью флага файла (`-f`). Независимо от того, где на Dockerfile самом деле находится, все рекурсивное содержимое файлов и каталогов в текущем каталоге отправляется демону Docker в качестве контекста сборки.

1.3.3 Пример построения контекста

Создайте каталог для контекста сборки и `cd` в него. Напишите «hello» в текстовом файле с именем `hello` и создайте Dockerfile, который будет запускать `cat` на нем (Рисунок 2). Создайте образ из контекста сборки (`.`).

```
$ mkdir myproject && cd myproject
$ echo "hello" > hello
$ echo -e "FROM busybox\nCOPY /hello /\nRUN cat /hello" > Dockerfile
$ docker build -t helloapp:v1 .
```


Рисунок 2 — Создание образа с файлом `hello`

Переместите Dockerfile и `hello` в отдельные каталоги и создайте вторую версию образа (не полагаясь на кеш последней сборки). Используйте `-f`, чтобы указать на Dockerfile и указать каталог контекста сборки (Рисунок 3).

```
$ mkdir -p dockerfiles context
$ mv Dockerfile dockerfiles && mv hello context
$ docker build --no-cache -t helloapp:v2 -f dockerfiles/Dockerfile context
```

Рисунок 3 — Создание второй версии образа с файлом `hello`

Непреднамеренное включение файлов, которые не нужны для построения образа, приводит к большому контексту сборки и большому размеру образа. Это может увеличить время создания образа, время его извлечения и отправки, а также размер среды выполнения контейнера. Чтобы увидеть, насколько велик ваш контекст сборки, найдите сообщение, подобное этому, при создании вашего Dockerfile (Рисунок 4):



```
Sending build context to Docker daemon 187.8MB
```

Рисунок 4 — Окончание сборки Docker образа

1.3.4 Исключение при помощи .dockerignore

Чтобы исключить файлы, не относящиеся к сборке (без реструктуризации исходного репозитория), используйте .dockerignore файл. Этот файл поддерживает шаблоны исключения, аналогичные .gitignore файлам.

1.3.5 Многоэтапные сборки

Многоэтапные сборки позволяют значительно уменьшить размер конечного изображения, не пытаясь уменьшить количество промежуточных слоев и файлов.

Поскольку образ создается на заключительном этапе процесса сборки, вы можете свести к минимуму количество слоев изображения, используя кэш сборки.

Например, если ваша сборка содержит несколько слоев, вы можете упорядочить их от менее часто изменяемых (чтобы обеспечить повторное использование кэша сборки) к более часто изменяемым:

- Установите инструменты, необходимые для создания вашего приложения;
- Установите или обновите зависимости библиотеки;
- Создайте свое приложение.

Пример многоэтапного Dockerfile для приложения на Go (Рисунок 5):

```

# syntax=docker/dockerfile:1
FROM golang:1.16-alpine AS build

# Install tools required for project
# Run `docker build --no-cache .` to update dependencies
RUN apk add --no-cache git
RUN go get github.com/golang/dep/cmd/dep

# List project dependencies with Gopkg.toml and Gopkg.lock
# These layers are only re-built when Gopkg files are updated
COPY Gopkg.lock Gopkg.toml /go/src/project/
WORKDIR /go/src/project/
# Install library dependencies
RUN dep ensure -vendor-only

# Copy the entire project and build it
# This layer is rebuilt when a file changes in the project directory
COPY . /go/src/project/
RUN go build -o /bin/project

# This results in a single layer image
FROM scratch
COPY --from=build /bin/project /bin/project
ENTRYPOINT ["/bin/project"]
CMD ["--help"]

```

Рисунок 5 — Многоэтапная сборка Go-проекта

1.3.6 Минимизирование количества слоев

В старых версиях Docker было важно свести к минимуму количество слоев в образах, чтобы обеспечить их производительность. Следующие функции были добавлены, чтобы уменьшить это ограничение:

Только инструкции RUN, COPY, ADD создают слои. Другие инструкции создают временные промежуточные образы и не увеличивают размер сборки.

По возможности используйте многоэтапные сборки и копируйте в окончательный образ только те артефакты, которые вам нужны. Это позволяет включать инструменты и информацию об отладке на промежуточных этапах сборки без увеличения размера конечного образа.

1.3.7 Работа с многострочными аргументами

По возможности упрощайте последующие изменения, сортируя многострочные аргументы в алфавитно-цифровом порядке. Это помогает избежать дублирования пакетов и упрощает обновление списка. Также это улучшает возможность чтения файла. Пример с `buildpack-deps` на Рисунке 6.

```
RUN apt-get update && apt-get install -y \  
    bzip2 \  
    cvs \  
    git \  
    mercurial \  
    subversion \  
    && rm -rf /var/lib/apt/lists/*
```

Рисунок 6 — Многострочные аргументы

1.3.8 Использование кеша сборки

При создании образа Docker выполняет инструкции в вашем Dockerfile, выполняя каждую в указанном порядке. По мере проверки каждой инструкции Docker ищет существующий образ в своем кеше, который он может повторно использовать, а не создает новый (дубликат) образ.

Если вы вообще не хотите использовать кэш, вы можете использовать `--no-cache=true` опцию в `docker build` команде. Однако, если вы разрешаете Docker использовать свой кеш, важно понимать, когда он может и не может найти подходящее изображение. Основные правила, которым следует Docker, изложены ниже:

— Начиная с родительского образа, который уже находится в кэше, следующая инструкция сравнивается со всеми дочерними образами, полученными из этого базового образа, чтобы определить, был ли один из них создан с использованием той же самой инструкции. В противном случае кэш становится недействительным.

— В большинстве случаев достаточно просто сравнить инструкцию в Dockerfile с подобной в одном из дочерних образов. Однако некоторые инструкции требуют дополнительного изучения и пояснений.

— Для инструкций ADD и COPY содержимое файла (файлов) в образе проверяется, и для каждого файла вычисляется контрольная сумма. В этих контрольных суммах время последнего изменения и последнего доступа к файлу (файлам) не учитывается. Во время поиска в кэше контрольная сумма сравнивается с контрольной суммой в существующих образах. Если что-то изменилось в файле (файлах), например, содержимое и метаданные, кэш становится недействительным.

— Помимо команд ADD и COPY, проверка кеша не просматривает файлы в контейнере, чтобы определить совпадение с кешем. Например, при обработке RUN apt-get -y update команды файлы, обновленные в контейнере, не проверяются на наличие попадания в кэш. В этом случае для поиска соответствия используется только сама командная строка.

Как только кэш становится недействительным, все последующие Dockerfile Команды генерируют новые изображения, а кэш не используется.

Основные команды описаны в теоретическом материале, поэтому не приводятся здесь дополнительно.

1.3.9 Задание 1

Создать один или несколько Dockerfile, в которых каждая из нижеприведенных команд будет использована хотя бы 1 раз:

- FROM;
- RUN;
- LABEL;
- CMD;
- EXPOSE;
- ENV;
- ADD;
- COPY;
- ENTRYPOINT;

- VOLUME;
- USER;
- WORKDIR;
- ONBUILD.

1.3.10 Задание 2

- Составлен Dockerfile с веб-приложением.
- Параметры веб приложения:
 - Написано с помощью фреймворка Spring Boot;
 - Взаимодействует с СУБД PostgreSQL;
 - Реализованы следующие эндпоинты:
 - добавление элемента,
 - вывод списка всех элементов,
 - получение фото герба РТУ МИРЭА.
- При сборке проекта с помощью консольной утилиты wget скачан файл https://www.mirea.ru/upload/medialibrary/80f/MIREA_Gerb_Colour.png по адресу https://www.mirea.ru/upload/medialibrary/80f/MIREA_Gerb_Colour.png. Файл должен быть доступен по эндпоинту веб-сервиса.
- Порт СУБД должен быть получен из переменной окружения, указанной в Dockerfile.
- Dockerfile должен содержать 2 стадии: сборка и запуск jar файла, стадии должны быть изолированы.
- В LABEL должны быть указаны ФИО и группа студента.
- По завершению запуска сервиса произведен вывод строки с вашим ФИО “Сборка и запуск произведены. Автор: {ФИО студента}” с помощью команды ONBUILD.
- Docker образ загружен в DockerHub.

1.4 Вопросы к практической работе

- 1 Опишите процесс запуска приложения внутри контейнера Linux, используя Docker.
- 2 Что такое образ Docker и для чего он нужен?
- 3 Как соотносятся между собой файлы Dockerfile и Docker-Compose?
- 4 Что такое Dockerfile?
- 5 Опишите политики перезапуска контейнера.
- 6 Назовите все возможные состояния контейнеров.

1.5 Критерии оценки

- Показан(ы) Dockerfile удовлетворяющие требованиям в практическом задании
- Сделан отчет с описанием и скриншотами выполненных заданий
- За выполнение данной практической работы можно максимально получить 2 балла.

1.5.1 Критерии на выставление 2 баллов:

- Соблюдены общие требования выполнения практических работ, представленные в документе “Требования к выполнению практических работ”.
- Реализован Dockerfile со всеми требуемыми командами в задании 1.
- Реализован Dockerfile по критериям в задании 2.
- Показана возможность запуска контейнеров
- Сделан отчет с описанием и скриншотами выполненных заданий
- Дан полный и развернутый ответ на все вопросы преподавателя, как по вопросам к практике, так и по дополнительным вопросам к выполненному заданию.

1.5.2 Критерии на выставление 1 балла:

- Соблюдены общие требования выполнения практических работ, представленные в документе “Требования к выполнению практических работ”.
- Реализован Dockerfile со всеми требуемыми командами в задании 1.
- Не реализован Dockerfile по критериям в задании 2.
- Показана возможность запуска контейнеров.
- Сделан отчет с описанием и скриншотами выполненных заданий
- Дан полный и развернутый ответ на все вопросы преподавателя на вопросы к практической работе, но дополнительные вопросы остались не отвечены: студент не смог полностью описать и аргументированно устно объяснить ход проделанной работы, все шаги, студент не может объяснить и описать используемые технологии.

1.5.3 Критерии на выставление 0 баллов:

- Не соблюдены общие требования выполнения практических работ, представленные в документе “Требования к выполнению практических работ”.
- Не реализован(ы) Dockerfile со всеми требуемыми командами в задании 1.
- Не реализован Dockerfile по критериям в задании 2.
- Не показана возможность запуска контейнеров
- Сделан отчет с описанием и скриншотами выполненных заданий
- Студент не смог ответить ни на вопросы к практической работе, ни на вопросы к ходу выполнения работы.