Практическая работа №1.

Теоретическое введение.

Краткое напоминание про технологию Docker

Существует проблема разработки того или иного приложения и его развертывания на других машинах. Самыми частыми решениями данной проблемы является установочные скрипты, облачные сервисы и виртуальные машины. Описанные подходы не являются оптимальными что раздувает техническую поддержку до максимума, а также медленны и тяжеловесны. Одним из вариантов решения данной задачи является докер, который представляет технологию контейнеризации.

Подобно виртуальной машине докер запускает свои процессы в собственной, заранее настроенной операционной системе. Но при этом все процессы докера работают на физическом host-сервере, деля все процессоры и всю доступную память со всеми другими процессами, запущенными в host-системе. Подход, используемый Docker, находится посередине между запуском всего на физическом сервере и полной виртуализацией, предлагаемой виртуальными машинами. Этот подход называется контейнеризацией.

Основными компонентами докера является:

- docker daemon — сердце докера. Это демон, работающий на хост-машине, и умеющий сохранять с удалённого репозитория и загружать на него образы, запускать из них контейнеры, следить за запущенными контейнерами, собирать логи и настраивать сеть между контейнерами (а с версии 0.8 и между машинами). А еще именно демон создает образы контейнеров, хоть и может показаться, что это делает docker-client.

- docker — это консольная утилита для управления docker-демоном по HTTP. Она устроена очень просто и работает предельно быстро. Вопреки заблуждению, управлять демоном докера можно откуда угодно, а не только с той же машины. В сборке нового образа консольная утилита docker принимает пассивное участие: архивирует локальную папку в tar.gz и передает по сети docker-daemon, который и делает всю работу. Именно из-за передачи контекста демону по сети, лучше собирать тяжелые образы локально.

- docker Hub централизованно хранит образы контейнеров. Когда вы пишете “docker run ruby”, docker скачивает самый свежий образ с ruby именно из публичного репозитория. Изначально хаба не было, его добавили уже после очевидного успеха первых двух частей.

Для того, чтобы установить данное программное обеспечение нужно воспользоваться инструкцией на официальном сайте[10].

Рассмотрим самый простой сценарий работы с технологией докер. Для это возьмем готовый образ. Напомним, что образ (image) в контейнеризации представляет собой упорядоченную коллекцию изменений корневой файловой системы и соответствующих параметров выполнения для использования во время выполнения контейнера. Образ обычно содержит объединение многоуровневых файловых систем, расположенных друг на друге. У образа нет состояния, и оно никогда не меняется. Для хранения образов докер предоставляет специальный сервис, называемый docker registry. Docker registry предназначен для хранения и дистрибуции готовых образов.

Для того, чтобы получить готовый образ из данного хранилища нужно воспользоваться командой: *docker pull*.

Для примера будет рассмотрен тестовый образ busybox. Следует также упомянуть, что при неуказании версии будет скачана последняя.

Для получения последней версии busybox воспользуемся командой:

docker pull busybox.

Теперь, при вводе команды *docker images*, мы увидим список всех доступных образов.

Теперь, когда у нас есть образ запустим докер-контейнер. Опираясь на официальную документацию, контейнер — это экземпляр образа во время выполнения. Он состоит из образа, среды выполнения и стандартного набора инструкций. Для запуска докер контейнера воспользуемся инструкцией: *docker run busybox*.

Чтобы получить информацию о запущенном контейнере нужно выполнить команду: docker ps. Будет выведена информация о запущенных контейнерах: id, имя, образ, статус, сколько времени назад был создан контейнер, порты.

Для того, чтобы остановить контейнер воспользуемся командой: *docker stop <имя контейнера>*.

Часто возникает ситуация, когда конфигурации уже существующего не хватает. Чтобы создавать свои собственные образы нужен специальный скрипт. Образы наследуются и, обычно, для создания своего первого образа мы берём готовый образ и наследуемся от него. Рассмотрим типичный скрипт (листинг 1). Чтобы запустить данный скрипт он должен иметь имя *Dockerfile* и не должен иметь типа.

```
# первая строка — это образ, на основе которого создается новый образ
FROM ubuntu:20.04

#с помощью данной команды можно выполнить команду внутри контейне
# данная команда установит обновления линукс
RUN apt-get install
```

```
# сообщим контейнеру какие порты слушать во время выполнения
EXPOSE 80

# копируем в файловую систему контейнера свой файл
ADD test.txt relativeDir/

# Задаём текущую рабочую директорию
WORKDIR /usr/src/my_app_directory

# Создаём том для хранения данных
VOLUME /my_volume
```

Листинг 1. Пример Dockerfile.

Чтобы собрать новый образ, нужно, в папке, где находится Dockerfile, выполнить команду: “*docker build -t my_image .”*. Тег -t задает название образа или тег, а точка обозначает работу в текущей директории.

Запуск собственноручно собранного образа идет такой же командой, что и скачанного из репозитория докера(Docker registry). Ведь при сборке образа на основе dockerfile взятие начального образа при его отсутствии идет именно из репозитория. Схематично данный процесс изображен на рисунке 1.

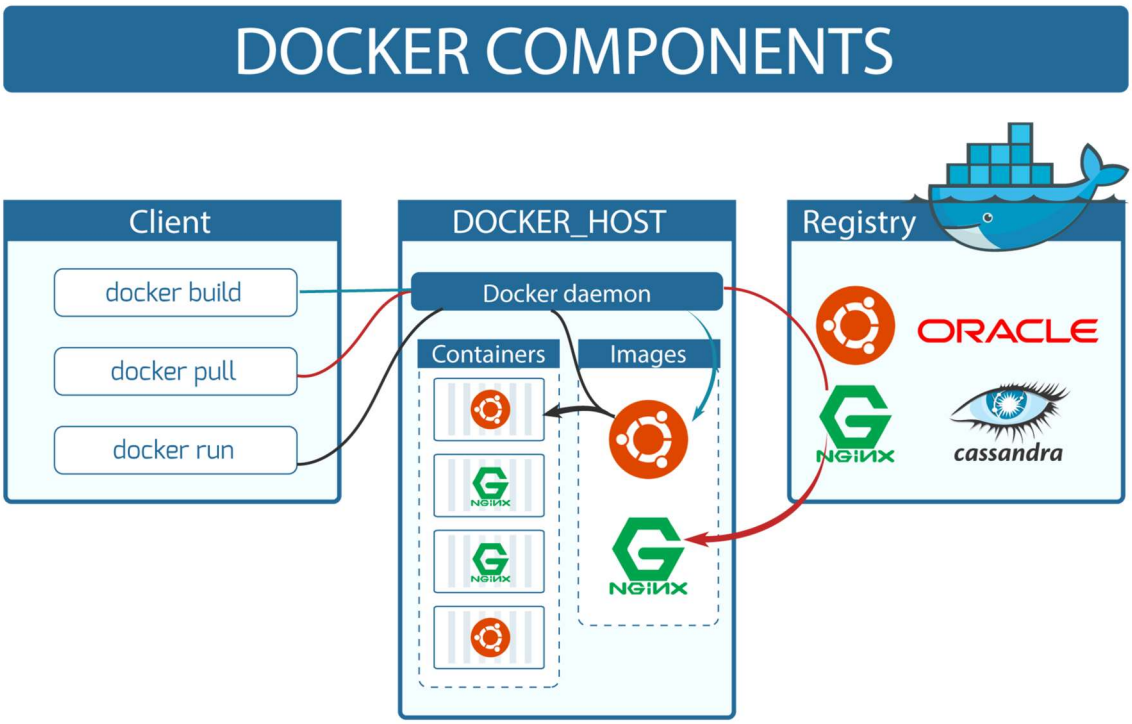

Рисунок 1. Компоненты докера.

Когда идет работа с несколькими контейнерами, то требуется механизм их объединения и оркестровки. Таким инструментом является Docker Compose [14]. Это средство для решения задач развертывания проектов. Docker Compose используется для одновременного управления несколькими контейнерами, входящими в состав приложения. Этот инструмент предлагает те же возможности, что и Docker, но позволяет работать с более сложными приложениями.

Также следует упомянуть такой инструмент, как публичный регистр контейнеров Docker Hub [15].

Для дальнейшего изучения данной технологии рекомендуем ознакомиться с полезными ссылками, которые представлены в списке полезных ссылок [1-11].

Установка PHP.

Для установки PHP введем следующие постулаты:

1. Встроенный в PHP веб-сервер является не полноценным веб-сервером, а application сервером, предназначенным как сказано в документации для помощи в разработке приложений. Он также может быть полезным в тестовых целях или для демонстрации приложения, запускаемого в полностью контролируемом окружении. Он не выполняет функции полноценного веб-сервера и не должен использоваться в общедоступных сетях и проектах.
2. PHP имеет множество настроек для создания требуемой конфигурации.
3. PHP имеет разные механизмы взаимодействия с разными веб-серверами.
4. Настройки конфигурации PHP производятся в специальном файле php.ini.

Для полноценной работоспособности конфигурации нужны: операционная система, Веб-сервер, язык программирования и База данных. Из всего этого следует идея технологии LAMP — акроним, обозначающий набор (комплекс) серверного программного обеспечения, широко используемый в интернете. LAMP назван по первым буквам входящих в его состав компонентов:

- Linux — операционная система Linux;
- Apache — веб-сервер;
- MariaDB / MySQL — СУБД;
- PHP — язык программирования, используемый для создания веб-приложений (помимо PHP могут подразумеваться другие языки, такие как Perl и Python).

Содержание данного набора может варьироваться в зависимости от задач и технического задания, например:

- LEMP — Nginx вместо Apache (Nginx читается Engine-X)
- LNMP — другой вариант названия 'Nginx вместо Apache'
- LLSMP - Linux, LiteSpeed, MySQL, PHP
- BAMP — BSD вместо Linux
- MAMP — Mac OS X вместо Linux.
- SAMP — Solaris вместо Linux
- WAMP — Microsoft Windows вместо Linux
- WASP — Windows, Apache, SQL Server и PHP
- WIMP — Windows, IIS, MySQL и PHP
- PAMP — Personal Apache MySQL PHP — набор серверов для платформы S60. Специфика платформы накладывает свой отпечаток на работу комплекса. Так, в частности, модули PHP получают и возвращают строки только в кодировке UTF-8.
- FNMP — FreeBSD и Nginx вместо Linux и Apache.
- XAMPP — кроссплатформенная сборка веб-сервера, X (любая из четырёх операционных систем), Apache, MySQL, PHP, Perl

Установка PHP для различных конфигураций описана в официальной документации[12].

Полезные ссылки.

1. Видео “Введение в Докер” на английском языке от создателя: Introduction to Docker (https://www.youtube.com/watch?v=Q5POuMHxW-0)
2. Статья о назначении докера простыми словами: https://habr.com/ru/post/309556/
3. Более сложная и подробная статья про докер: https://habr.com/ru/post/277699/
4. Хорошая статья с пингвинами для прочтения после туториала по докеру: https://habr.com/ru/post/250469/

5. Официальная документация докера: https://docs.docker.com/
6. Статья о конкретном опыте использования докер контейнеров: https://habr.com/ru/post/247969/
7. Туториал по докеру: https://badcode.ru/docker-tutorial-dlia-novichkov-rassmatrivaiem-docker-tak-iesli-by-on-byl-ighrovoi-pristavkoi/
8. Туториал по докеру с Хабра: https://habr.com/ru/post/310460/
9. Шпаргалка с командами Docker: https://habr.com/ru/company/flant/blog/336654/
10. Ссылка на скачивание докера с официального сайте: https://www.docker.com/products/docker-desktop
11. Отличная статья про dockerfile: https://habr.com/ru/company/ruvds/blog/439980/
12. Установка и настройка PHP: https://www.php.net/manual/ru/install.php
13. Настройка среды PhpStorm и полезные фичи: https://habr.com/ru/post/282003/
14. Про docker compose: https://habr.com/ru/company/ruvds/blog/450312/
15. Docker hub: https://hub.docker.com/

Задание.

Вам предлагается создать свою конфигурацию серверного программного обеспечения, в которой должны присутствовать веб-сервер, операционная система, язык программирования и база данных. Данная конфигурация будет использоваться для выполнения следующих практических работ по данной дисциплине и для выполнения курсового проектирования.

Дается рекомендация использовать ОС Linux, язык программирования PHP, веб-сервер Apache и СУБД MySQL.

Для проверки работоспособности вашей конфигурации требуется инициализировать базу данных: создать отдельного пользователя для работы с ней, создать базу данных, в которой создать таблицу пользователи с полями: идентификационный номер, имя, фамилия. Также для проверки вашей конфигурации требуется сгенерировать тестовую страничку, содержащую выборку из созданной таблицы и информационное сообщение о версии языка программирования, его настройках и конфигурации. Примерный скриншот (рисунок 2) приведен ниже.

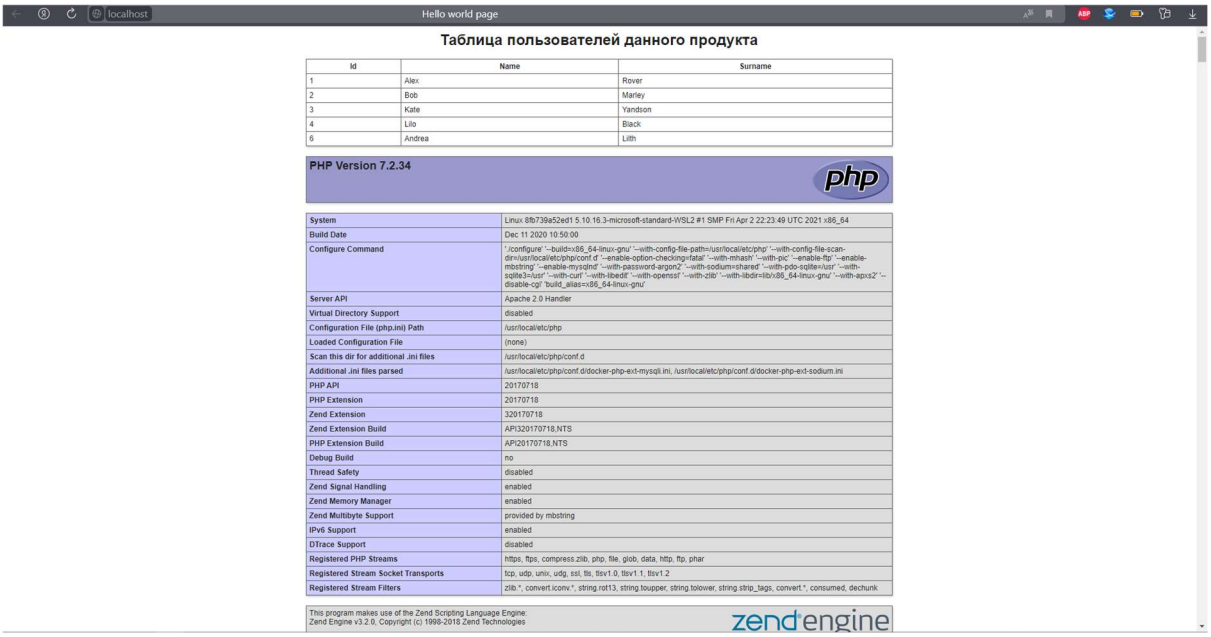

Рисунок 2. Тестовая страница.

Для облегчения работы с рекомендуемыми инструментами студенту предоставляется скрипт инициализации БД для СУБД MYSQL и скрипт генерации тестовой страницы вместе с оформлением на языке PHP.

Также для выполнения задания рекомендуется использовать технологию контейнеризации и оркестровки контейнеров.

Вопросы к практической работе.

1. Сервер и клиент.
2. База данных.
3. API.
4. Сервис, отличия от сервера.
5. Архитектура клиент-сервер.
6. Виды сервисов.
7. Масштабируемость.
8. Протоколы передачи данных.
9. Тонкий и толстый клиенты.
10. Паттерн MVC: общие тезисы.
11. Паттерн MVC: Model-View-Presenter.
12. Паттерн MVC: Model-View-View Model.
13. Паттерн MVC: Model-View-Controller.
14. Docker: общие тезисы и определения.
15. Dockerfile.
16. Docker Compose.
17. LAMP.

Критерии оценки.

За выполнение данной практической работы можно максимально получить 2 балла.

Критерии на выставление 2 баллов:

- Соблюдены общие требования выполнения практических работ, представленные в документе “Требования к выполнению практических работ”.

- Показана полная работоспособность серверной конфигурации в режиме реального времени.
- Дан полный и развернутый ответ на все вопросы преподавателя, как по вопросам к практике, так и по дополнительным вопросам к выполненному заданию.

Критерии на выставление 1 балла:

- Соблюдены общие требования выполнения практических работ, представленные в документе “Требования к выполнению практических работ”.
- Показана полная работоспособность серверной конфигурации в режиме реального времени.
- 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.

Критерии на выставление 0 баллов:

- Соблюдены общие требования выполнения практических работ, представленные в документе “Требования к выполнению практических работ”.
- Не показана полная работоспособность серверной конфигурации в режиме реального времени.
- Студент не смог ответить ни на вопросы к практической работе, ни на вопросы к ходу выполнения работы.