**Практическая работа №1.**

**Введение**

Backend – это серверная часть веб-приложения, которая обычно отвечает за обработку запросов пользователя, взаимодействие с базой данных, доступ к файлам и другим ресурсам. Она обеспечивает работу веб-приложения с общими ресурсами, включая базы данных, сервера и другие системы. Backend веб-приложения могут быть написаны на различных языках программирования, включая PHP, Java, Golang, Python и т.д.

Backend является одной из ключевых компонентов веб-приложения. Он обеспечивает логику, необходимую для обработки запросов, а также управление данными и их хранение. Backend веб-приложения работает в связке с frontend (клиентской частью), которая отображает пользовательский интерфейс и обрабатывает пользовательский ввод.

Таким образом, backend является неотъемлемой частью веб-приложения и обеспечивает его надежную работу и безопасность. Разработка backend части веб-приложения - это высоко востребованный навык в сфере IT.

На данном курсе вам необходимо будет выполнить 16 практических работ, которые разделены на 4 блока по языкам программирования. У первого блока практик язык программирования PHP. В ходе работы по данной практике вы должны будете создать первый простейший веб-сервер.

**Теория**

1. **Docker**

В данной практики студенту будет необходимо вспомнить про такую технологию, как Docker, а также понять его назначение в разработке серверной части интернет-ресурса.

Одна из главных причин, почему Docker стал популярным инструментом в backend разработке, заключается в его способности к оптимизации процесса развертывания приложений на многих серверах. Далее ниже перечислено несколько пунктов, из-за которых Docker становится все более востребованным инструментом в backend разработке:

* Изолированность. Docker контейнеры создают изолированные среды, которые обеспечивает изоляцию приложения от других приложений на том же хосте. Контейнеры обычно содержать только те зависимости и пакеты, которые им необходимы для работы. Изолированные контейнеры помогают обезопасить приложение и защитить его от конфликтов, связанных с непредсказуемыми версиями библиотек и зависимостей. К тому же контейнеры гарантируют, что приложение не повлияет на другие приложения на хосте;
* Переносимость. Docker контейнеры обеспечивают переносимость вашего приложения между разными серверами и окружениями. Это означает, что вы можете разработать приложение на своем локальном компьютере и затем развернуть его в любой другой среде, где установлен Docker. Контейнеры позволяют выводить приложение быстрее на рынок и сокращать время релиза;
* Быстрое развертывание. С Docker можно быстро создавать и развертывать новые контейнеры с вашим приложением. Вы можете использовать готовые образы, которые содержат все необходимые зависимости и пакеты, что позволяет избежать дополнительной установки и настройки. Это позволяет разрабатывать и улучшать приложение более быстро и эффективно;
* Удобная локальная разработка. Docker позволяет запускать виртуальные контейнеры на вашем локальном компьютере, что облегчает локальную разработку и отладку вашего приложения. Вы можете использовать Docker для разработки на любой платформе, в том числе Windows, Mac OS и Linux. Docker контейнеры позволяют разрабатывать приложение локально и тестировать его в клиентском окружении.

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.

Чтобы установить docker, необходимо перейти на официальный сайт https://www.docker.com/ и следовать инструкции по установке.

Теперь можно проверить, успешно ли установился docker. Для этого нужно запустить docker и написать в командной строке следующую команду:

|  |
| --- |
| docker run hello-world |

Листинг 1. Команда запуска контейнера.

Если вы увидели следующее сообщение, то всё установлено верно.

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

Листинг 2. Сообщение об успешном запуске контейнера.

Также вот список основных команд, которые можно использовать в качестве шпаргалки:

|  |
| --- |
| 1. `docker run` - Запуск контейнера  2. `docker ps` - Отобразить список работающих контейнеров  3. `docker ps -a` - Отобразить список всех контейнеров (включая остановленные)  4. `docker stop <container-id>` - Остановить контейнер  5. `docker rm <container-id>` - Удалить контейнер  6. `docker images` - Отобразить список имеющихся образов  7. `docker rmi <image-id>` - Удалить образ  8. `docker pull <image-name>` - Скачать образ с Docker Hub  9. `docker push <image-name>` - Отправить образ на Docker Hub  10. `docker exec -it <container-id> bash` - Запустить bash внутри контейнера |

Листинг 3. Основные команды docker.

Теперь давайте рассмотрим пример создания docker-контейнера для веб-сервера.

|  |
| --- |
| # первая строка — это образ, на основе которого создается новый образ  FROM ubuntu:20.04  #с помощью данной команды можно выполнить команду внутри контейнера  # 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 4. Пример dockerfile.

1. **Docker-compose**

Docker-compose — это инструмент для управления множеством контейнеров Docker или же инструмент оркистрации контейнеров, который позволяет определять и запускать несколько контейнеров Docker в единой среде. Docker-compose позволяет автоматизировать процесс создания, запуска, остановки и удаления контейнеров, упрощая процесс развертывания много контейнерных приложений.

Docker Compose позволяет определить настройки контейнера, включающие сетевую конфигурацию, зависимости, переменные среды и другие параметры. Он позволяет создавать и запускать несколько контейнеров, связанных между собой, как единое приложение. Docker Compose позволяет легко масштабировать и управлять несколькими контейнерами в единой среде, что делает его инструментом для много контейнерного развертывания.

Docker Compose используется в различных сценариях, таких как развертывание микросервисов, тестирование, разработка и многое другое. Он позволяет определять и управлять множеством контейнеров из одного файла YAML, что облегчает управление и обеспечивает более простую конфигурацию и управление контейнерами. Docker Compose также позволяет создавать собственные настраиваемые образы Docker, которые можно переносить между различными окружениями.

Таким образом, Docker Compose - это средство автоматизации в управлении множеством контейнеров Docker, которое облегчает процесс создания, запуска и управления многоконтейнерными приложениями.

Список основных команд для docker-compose:

|  |
| --- |
| 1. `docker-compose up` - Запустить контейнеры, определенные в docker-compose.yml  2. `docker-compose up -d` - Запустить контейнеры в фоновом режиме (без вывода логов)  3. `docker-compose down` - Остановить и удалить запущенные контейнеры, определенные в docker-compose.yml  4. `docker-compose build` - Построить образы, определенные в docker-compose.yml  5. `docker-compose start` - Запустить остановленные контейнеры  6. `docker-compose stop` - Остановить работающие контейнеры  7. `docker-compose restart` - Перезапустить контейнеры  8. `docker-compose logs` - Отобразить логи контейнеров, определенных в docker-compose.yml  9. `docker-compose ps` - Отобразить список контейнеров, определенных в docker-compose.yml  10. `docker-compose config` - Отобразить конфигурацию, определенную в docker-compose.yml |

Листинг 5. Основные команды docker-compose.

Давайте рассмотрим пример конфигурационного YAML файла docker-compose.

|  |
| --- |
| version: '3'  services:  db:  image: mysql:5.7  volumes:  - db\_data:/var/lib/mysql  restart: always  environment:  MYSQL\_ROOT\_PASSWORD: password  MYSQL\_DATABASE: text  MYSQL\_USER: User  MYSQL\_PASSWORD: 123456789  web\_server:  build:  context: .  dockerfile: Dockerfile  ports:  - 8000:80  restart: always  links:  - database  volumes:  db\_data: |

Листинг 6. Пример yaml файла docker-compose.

Описание:

* `version: '3'` - указывает на использование версии 3 docker-compose.
* `services:` - определяет список сервисов, которые будут развернуты.
* `db:` - определяет контейнер MySQL.
* `image: mysql:5.7` - указывает на использование образа MySQL версии 5.7 из Docker Hub.
* `volumes:` - указывает на использование именованного тома db\_data для хранения данных.
* - `restart: always` - контейнер будет автоматически перезапущен, если он остановится с ошибкой.
* `environment:` - определяет переменные окружения, используемые контейнером MySQL.
* `web\_server:` - определяет контейнер Веб-сервера.
* `build` - определяет как создать данный контейнер.
* `ports:` - привязывает порт 8000 узла к порту 80 контейнера.
* `context: .` - указывает откуда брать конфигурацию для контейнера.
* `dockerfile: Dockerfile` - указывает тип файла конфигурации.
* `restart: always` - контейнер будет автоматически перезапущен, если он остановится с ошибкой.
* `links: database` - контейнер будет в одной сети с конейнером базы данных и сможет взаимодействовать с ним.
* `volumes:` - определяет именованный том db\_data.

Этот пример демонстрирует, как использовать Docker Compose для запуска двух контейнеров - базы данных MySQL и веб-приложения, связанных друг с другом.

1. **Серверная конфигурация**

Для полноценной работоспособности конфигурации нужны: операционная система, Веб-сервер, язык программирования и База данных, но база данных будет устанавливаться в следующей практике. 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

Но в данной работе вам необходимо установить Apache.

Apache — это самый популярный веб-сервер, который запускается на миллионах веб-сайтов во всем мире. Apache является свободной и открытой программой, которая доступна для распространения и использования бесплатно.

Основной целью Apache является обслуживание веб-страниц пользователям, обрабатывая запросы от веб-браузеров и отображая содержимое веб-страниц в ответ на эти запросы.

В дополнение к основным функциям веб-сервера, Apache также предоставляет множество расширенных функций, приложений и библиотек, которые могут быть использованы веб-разработчиками для создания динамических веб-сайтов и приложений. Apache также поддерживает различные языки программирования, такие как PHP и Perl, которые могут быть использованы для создания динамических веб-страниц.

**Задание**

Вам необходимо создать конфигурацию docker-compose для вашего веб-сервера.

Docker контейнер должен включать:

* ОС Linux
* Веб-сервер Apache

Docker-compose:

* Docker-контейнер, созданный ранее
* Примонтированные тома
* Настройки портов

Конечной задачей студента будет корректный запуск приложенного php скрипта генерации страницы с характеристиками веб-сервера и его работа на созданном веб-сервере.

**Полезные ссылки**

1. Видео “Введение в Докер” на английском языке от создателя: [Introduction to Docker](https://www.youtube.com/watch?v=Q5POuMHxW-0) (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/>

**Вопросы**

* Что такое Apache и для чего он нужен?
* Что такое оркестрация контейнеров и для чего это нужно?
* Что такое сервер?