**Министерство образования и науки Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Владимирский государственный университет**

**имени Александра Григорьевича и Николая Григорьевича Столетовых»**

(ВлГУ)

Кафедра информационных систем и программной инженерии

Технологии разработки веб-приложений

Методические указания к лабораторным работам

**«Использование Docker для запуска приложений на Laravel»**

Владимир 2020

**Цель работы**

Познакомиться со средством контейнеризации Docker и использовать его для запуска приложения на Laravel.

**Теоретические сведения**

Docker — это открытая платформа для разработки, доставки и эксплуатации приложений. Docker разработан для более быстрого выкладывания ваших приложений.

С помощью Docker можно отделить приложение от инфраструктуры и обращаться с инфраструктурой как управляемым приложением. Docker помогает выкладывать код быстрее, быстрее тестировать, быстрее выкладывать приложения и уменьшить время между написанием кода и запуска кода.

Docker делает это с помощью легковесной платформы контейнерной виртуализации, используя процессы и утилиты, которые помогают управлять и выкладывать ваши приложения.

В своем ядре Docker позволяет запускать практически любое приложение, безопасно изолированное в контейнере. Безопасная изоляция позволяет вам запускать на одном хосте много контейнеров одновременно. Легковесная природа контейнера, который запускается без дополнительной нагрузки гипервизора, позволяет вам добиваться больше от вашего железа.

Саму актуальную документацию о Docker можно найти на официальном сайте: https://docs.docker.com/.

**Использование Docker**

* Docker прекрасно подходит для организации цикла разработки. Docker позволяет разработчикам использовать локальные контейнеры с приложениями и сервисами. Что в последствии позволяет интегрироваться с процессом постоянной интеграции и выкладывания (continuous integration and deployment workflow).
* Основанная на контейнерах Docker платформа позволят легко портировать полезную нагрузку. Docker контейнеры могут работать на локальной машине, как реальной, так и на виртуальной машине в дата центре, так и в облаке.
* Docker легковесен и быстр. Он предоставляет устойчивую, рентабельную альтернативу виртуальным машинам на основе гипервизора. Он особенно полезен в условиях высоких нагрузок, например, при создания собственного облака или платформа-как-сервис (platform-as-service). Но он так же полезен для маленьких и средних приложений, когда вам хочется получать больше из имеющихся ресурсов.

**Архитектура Docker**

Docker использует архитектуру клиент-сервер. Docker клиент общается с демоном Docker, который берет на себя тяжесть создания, запуска, распределения контейнеров. Оба, клиент и сервер могут работать на одной системе, вы можете подключить клиент к удаленному демону Docker. Клиент и сервер общаются через сокет или через RESTful API.

**Docker-демон**

Демон запускается на хост-машине. Пользователь не взаимодействует с сервером на прямую, а использует для этого клиент.

**Docker-клиент**

Docker-клиент, программа docker — главный интерфейс к Docker. Она получает команды от пользователя и взаимодействует с docker-демоном.

**Образы**

Docker-образ — это read-only шаблон. Например, образ может содержать ОС Ubuntu c Apache и приложением на ней. Образы используются для создания контейнеров. Docker позволяет легко создавать новые образы, обновлять существующие, или вы можете скачать образы созданные другими людьми. Образы — это компонента сборки docker-а.

**Реестр**

Docker-реестр хранит образы. Есть публичные и приватные реестры, из которых можно скачать либо загрузить образы. Публичный Docker-реестр — это Docker Hub. Там хранится огромная коллекция образов. Как вы знаете, образы могут быть созданы вами или вы можете использовать образы созданные другими. Реестры — это компонента распространения.

**Контейнеры**

Контейнеры похожи на директории. В контейнерах содержится все, что нужно для работы приложения. Каждый контейнер создается из образа. Контейнеры могут быть созданы, запущены, остановлены, перенесены или удалены. Каждый контейнер изолирован и является безопасной платформой для приложения. Контейнеры — это компонента работы.

**Как работает образ?**

Каждый образ состоит из набора уровней. Docker использует union file system для сочетания этих уровней в один образ. Union file system позволяет файлам и директориями из разных файловых систем (разным ветвям) прозрачно накладываться, создавая когерентную файловую систему. Одна из причин, по которой docker легковесен — это использование таких уровней. Когда вы изменяете образ, например, обновляете приложение, создается новый уровень. Так, без замены всего образа или его пересборки, как вам возможно придётся сделать с виртуальной машиной, только уровень добавляется или обновляется. И вам не нужно раздавать весь новый образ, раздается только обновление, что позволяет распространять образы проще и быстрее. В основе каждого образа находится базовый образ. Например, ubuntu, базовый образ Ubuntu, или fedora, базовый образ дистрибутива Fedora. Также вы можете использовать образы как базу для создания новых образов. Например, если у вас есть образ apache, вы можете использовать его как базовый образ для ваших веб-приложений. Docker образы могут создаться из этих базовых образов, шаги описания для создания этих образов мы называем инструкциями. Каждая инструкция создает новый образ или уровень. Эти инструкции хранятся в файле Dockerfile. Docker считывает это Dockerfile, когда вы собираете образ, выполняет эти инструкции, и возвращает конечный образ.

**Как работает контейнер?**

Контейнер состоит из операционной системы, пользовательских файлов и метаданных. Как мы знаем, каждый контейнер создается из образа. Этот образ говорит docker-у, что находится в контейнере, какой процесс запустить, когда запускается контейнер и другие конфигурационные данные. Docker образ доступен только для чтения. Когда docker запускает контейнер, он создает уровень для чтения/записи сверху образа (используя union le system, как было указано раньше), в котором может быть запущено приложение.

**Docker Compose**

Docker Compose - это высокоуровневый инструмент для описания и запуска приложений, которые состоят из нескольких Docker контейнеров. С помощью Compose-файла можно описать все необходимые сервисы для приложения и запустить одной командой, Compose создаст и запустит все необходимые сервисы для вашего приложения. Работа с Docker Compose состоит из трёх шагов:

* создание Dockerfile, который содержит описание окружения, которое необходимо для вашего приложения;
* создание файла docker-compose.yml, в котором указываются сервисы вашего приложения и связи между ними;
* запуск приложения командой docker-compose.yml, после чего будут запущены все нужные сервисы вашего приложения.

**Выполнение работы**

В рамках этой работы будет выполнена ”контейнеризация” приложения, которое было разработано в рамках предыдущей работы.

**Установка Docker**

Самую свежую версию инструкции по установке Docker на можно найти по адресу: https://docs.docker.com/install/. Необходимо использовать **Community Edition**. После выполнения всех шагов из инструкции в вашем окружении будет установлена свежая версия Docker.

**Установка Docker Compose**

Самую свежую версию инструкции по установке Docker Compose можно найти по адресу: https://docs.docker.com/compose/install/. После выполнения всех шагов из инструкции в вашем окружении будет установлена свежия версия Docker Compose.

**”Контейнеризация” приложения**

**Создание Dockerfile**

В новой директории создаём файл Dockerfile со следующим содержимым:

FROM php:7.4.3-fpm-alpine

RUN docker-php-ext-install pdo

WORKDIR /var/www

COPY . /var/www

EXPOSE 9000

CMD ["php-fpm"]

FROM - указывает на базовый̆ образ, который̆ будет взят за основу для нашего образа. В данном случае используется образ с уже предустановленным PHP 7.4.

EXPOSE – открывает порт из контейнеров, собранных на данном образе.

RUN - описывает команды, которые будут выполнены при построении образа. В нашем случае мы устанавливаем расширение pdo.

COPY - позволяет копировать указанные файлы рабочего окружения в контейнер.

**Создание docker-compose.yml**

Создадим файл docker-compose.yml в директории со следующим содержимым:

version: "3.2"

services:

nginx:

image: nginx:alpine

volumes:

- ./var/www:/var/www

- ./nginx/conf.d/:/etc/nginx/conf.d/

ports:

- 80:80

- 443:443

depends\_on:

- php

php:

build:

context: .

volumes:

- ./var/www:/var/www

ports:

- 9000:9000

Здесь описывается два сервиса: веб-сервер Nginx и приложение. Также в файле описывается конфигурация портов, которые будут пробрасываться в контейнер с приложением и конфигурация монтирования томов.

Для корректной работы nginx необходимо описать конфигурацию, для этого создадим файл по пути ***nginx/conf.d/default.conf*** в проекте. Этот файл будет монтироваться в запущенные контейнеры nginx.

server {

listen 80*;*

index index.php index.html*;*

error\_log /var/log/nginx/error.log*;*

access\_log /var/log/nginx/access.log*;*

root /var/www/public*;*

location ~ \.php$ {

try\_files $uri =404*;*

fastcgi\_split\_path\_info ^(.+\.php)(/.+)$*;*

fastcgi\_pass php:9000*;*

fastcgi\_index index.php*;*

include fastcgi\_params*;*

fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name*;*

fastcgi\_param PATH\_INFO $fastcgi\_path\_info*;*

}

location / {

try\_files $uri $uri/ /index.php?$query\_string*;*

gzip\_static on*;*

}

}

Для проверки работоспособности можно добавить файл index.php в папку ./var/www/public с содержимым

<?php echo '<p>Hello World</p>';

Чтобы запустить описанные сервисы надо выполнить команду

docker-compose up --build

**Задания**

Для выполнения заданий необходимо использовать проект из лабораторной работы по Laravel.

1. Воспроизвести шаги из методички и показать работоспособность.
2. Добавить сервис БД и показать работоспособность взаимодействия приложения с БД запущенной в Docker.
3. Подключить сервис для обработки кэширования. Включить кэширование в Laravel. Показать работоспособность.