Разработка многостраничного сайта на PHP

ТЕМА 4.3 РАБОТА С КОНТЕЙНЕРАМИ
И СИСТЕМОЙ СБОРКИ DOCKER
ЗАНЯТИЕ № 8 - ЛЕКЦИЯ/ ПРАКТИКА

Тема занятия – Работа с контейнерами и системой сборки Docker

Цель занятия –

Изучить локальную разработку и тестирование в PHP с помощью Docker-compose, а также управление артефактами

Актуализация

На прошлом занятии мы познакомились с Docker, установили его на свой ПК, решили нашу первую задачу, написав ее на языке программирования PHP и загрузив в контейнер Docker.

Сегодня мы продолжим изучение этой системы, а также будем решать практические задачи на укрепление знаний, полученных по этой теме.

Содержание

- Введение
- Docker-compose
- Локальная разработка
- Тестирование с Docker-compose
- Управление артефактами
- Решение практических задач
- Заключение

Введение

Сегодня мы поговорим с вами о локальной разработке и тестировании в PHP с помощью Docker-compose и управлением артефактами.

Когда мы пишем программы, мы хотим, чтобы они работали правильно и быстро. Чтобы это проверить, нам нужно протестировать наши программы. Но как это сделать? Для этого мы можем использовать docker-compose!

Docker-compose

Docker-compose —

это инструмент, который позволяет запускать несколько контейнеров Docker одновременно, чтобы мы могли тестировать наши приложения на интерфейсе автомобиля.

Таким образом, мы предполагаем, что наши программы работают правильно перед тем, как загружать их на сервер.

Docker-compose

Также мы используем docker-compose для управления артефактами, то есть файлы, которые вызывают подозрения в разработке программ. Например, если мы создаем базу данных для нашего приложения, мы можем использовать docker-compose для создания и управления этой базой данных. Это поможет нам восстановить и восстановить состояние нашей базы данных при необходимости.

Docker-compose

В PHP мы можем использовать docker-compose для запуска и тестирования наших приложений, а также можем использовать его для управления базами данных, веб-серверами и другими сервисами, которые нужны нам для разработки наших приложений.

Docker-compose

Docker Compose не является абсолютно необходимым для управления несколькими контейнерами, поскольку это можно сделать с помощью одного только Docker, но на практике это очень неудобно.

Контейнеры описываются в конфигурационном файле YAML, а Docker Compose позаботится о сборке образов и запуске контейнеров, а также о некоторых других полезных вещах, таких как автоматическое подключение контейнеров к внутренней сети.

Docker-compose

После установки Docker и Docker Compose создайте папку для вашего проекта. Для примера назовем ее my-php-app. В этой папке создайте два файла: Dockerfile и docker-compose.yml.

Dockerfile создает образ, на основе которого запускаются контейнеры в будущем проекте. В настоящее время мы используем образ php: 7.4-apache.

В данном примере мы используем

базовый php:7.4-apache и

копируем содержимое

```
FROM php:7.4-apache
COPY . /var/www/html/
```

папки в каталог /var/www/html/внутри контейнера.

Docker-compose

Чтобы использовать контейнер, можно использовать команду:

```
docker-compose build
docker-compose up -d
```

Команда docker-compose build собирает образы на основе Dockerfile, а docker-compose up -d запускает контейнер в фоновом режиме. После этого вы можете открыть свой и перейти на http://localhost:8080 , чтобы увидеть результат.

Docker-compose

Теперь вы можете начать разработку и тестирование своего приложения, используя этот контейнер. Если вы внесли изменения в код, то просто перезагрузите контейнер, используя команду docker-compose up -d.

Если вы хотите использовать другую версию PHP или добавить другие компоненты, просто измените Dockerfile и docker-compose.yml в соответствии с требованиями.

Docker-compose

Также Docker Compose позволяет управлять дефектами, которые регулируются в процессе разработки, как база данных, кэши, очереди и т.д. Для этого в docker-compose.yml необходимо определить сервисы, которые вы хотите использовать.

Например, вы можете добавить сервис MySQL в docker-compose.yml:

```
version: '3.8'
services:
    app:
    build: .
    ports:
        - "8080:80"
    depends_on:
        - db
    db:
    image: mysql:5.7
    environment:
        MYSQL_ROOT_PASSWORD: root
        MYSQL_DATABASE: mydb
    volumes:
        - ./data:/var/lib/mysql
```

Docker-compose

Мы добавили сервис MySQL, который зависит от основного сервиса (приложения) и использует порт по умолчанию 3306. В конце мы добавили том для сохранения данных базы данных между запусками контейнера.

После этого вы можете использовать этот сервис в приложении, указав время подключения к базе данных. Все данные, сохраненные в базе данных, запоминаются в папке ./data

Docker-compose

Также можно использовать сервисы для кэширования, очередей и других инструментов, которые могут быть собраны для вашего приложения.

В конце концов, использование Docker и Docker Compose для оценки производительности и тестирования в PHP-проектах позволяет значительно ускорить процесс разработки, снизить вероятность возникновения ошибок в результате окружения, а также повысить портируемость вашего приложения.

Практическая часть

1) Создайте папку с названием ‘myapp’
2) Внутри папки ‘myapp’ создайте файл ‘index.php’

```php
<?php
echo "Hello, Docker!";
?>
```

3) Настройте файл ’Dockerfile’

```
FROM php:7.4-apache
COPY ./myapp /var/www/html/
EXPOSE 80
```

Практическая часть

4) Соберите образ Docker, используя команду docker build. Команда должна быть следующей:

```
docker build -t myapp .
```

5) Загрузить контейнер, используя команду docker run. Команда должна быть следующей:

```
docker run -p 80:80 myapp
```

Практическая часть

6) Открыть и оценить по адресу http://localhost. Вы должны увидеть сообщение "Привет, Докер!".

Эта задача сборки, как создать простое приложение на PHP, запаковать его в Docker-контейнер и реализовать его контексту на порту 80.

Практическая часть

Задача № 1

Создание Docker-образа для развертывания приложений на PHP с использованием Apache.

Решение

Для создания Docker-образа для развертывания приложений на PHP с использованием Apache можно использовать следующий Dockerfile:

```
FROM php:7.4-apache

COPY src/ /var/www/html/
```

Этот Dockerfile начинается с базового образа php:7.4-apache, который включает в себя PHP и веб-сервер Apache. Копируется содержимое каталога src/на веб-сервере /var/www/html/.

Решение

Затем необходимо собрать Docker-образ с помощью команды docker build. Допустим, Dockerfile находится в рабочем каталоге и представляет собой образ my-php-app:

```
docker build -t my-php-app .
```

Когда Docker-образ будет успешно собран, контейнер можно запустить с помощью команды docker run. Например, чтобы запустить контейнер с именем my-php-container:

```
docker run -d --name my-php-container -p 80:80 my-php-app
```

Решение

Эта команда запускает контейнер в фоновом режиме с именем

my-php-container, пробрасывает порт 80 контейнера только на порт 80 хоста и использует созданный Docker-образ my-php-app. Теперь приложение на PHP с использованием Apache будет запущено и доступно по адресу http://localhost.

Обратите внимание, что если приложение на PHP требует дополнительных расширений или модулей, их можно установить в Dockerfile с инструкциями RUN и инструкциями docker-php-ext-install.

Практическая часть

Задача № 2

Создание Docker-образа для развертывания приложений на PHP с использованием MySQL.

Решение

Для создания Docker-образа для развертывания приложений на PHP с использованием MySQL можно использовать следующий Dockerfile:

```
FROM php:7.4-apache

RUN docker-php-ext-install mysqli

RUN apt-get update && \
    apt-get install -y \
        mysql-client

COPY src/ /var/www/html/
```

Решение

Этот Dockerfile начинается с базового образа php:7.4-apache, который включает в себя PHP и веб-сервер Apache. Затем было разработано расширение mysqli для подключения к базе данных MySQL, а также разработан клиент MySQL. Копируется содержимое каталога src/на веб-сервере /var/www/html/.

Затем необходимо собрать Docker-образ с помощью команды docker build. Допустим, Dockerfile находится в рабочем каталоге и представляет собой образ my-php-mysql-app:

```
docker build -t my-php-mysql-app .
```

Решение

Когда Docker-образ будет успешно собран, контейнер можно запустить с помощью команды docker run. Например, чтобы использовать контейнер с именем my-php-mysql-container и подключить его к базе данных MySQL, используя имя хоста my-mysql-container:

```
docker run -d --name my-php-mysql-container -p 80:80 \
    --link my-mysql-container:mysql \
    -e MYSQL_HOST=mysql \
    -e MYSQL_USER=root \
    -e MYSQL_PASSWORD=secret \
    -e MYSQL_DATABASE=my_database \
    my-php-mysql-app
```

Решение

Эта команда запускает контейнер в фоновом режиме с именем my-php-mysql-container, пробрасывает порт 80 контейнера только на порт 80 хоста и использует созданный Docker-образ my-php-mysql-app. Контейнер также подключается к контейнеру базы данных MySQL с именем my-mysql-container и использует переменные оболочки для параметров подключения.

Обратите внимание, что если приложение на PHP требует дополнительных расширений или модулей, их можно установить в Dockerfile с инструкциями RUN и инструкциями docker-php-ext-install.

Заключение

Для создания Docker-образов для приложений на PHP необходимо использовать специальный файл Dockerfile, который содержит инструкции по установке и установке компонентов, а также инструкции по использованию приложения в контейнере.

Для большой разработки и тестирования в Docker, необходимо организовать свой проект в соответствии с лучшими практиками, разработать как использование docker-compose для управления контейнерами и образами, использовать утилиту для упрощения и управления Docker-контейнерами, а также использовать мониторинг и ведение журнала для плотности. и безопасность ваших приложений.

Таким образом, Docker предоставляет разработчикам PHP мощный инструмент для упрощения процесса разработки и развертывания приложений на PHP, что может значительно увеличить процесс разработки и доставки продукта на рынок.

Рефлексия

Таким образом, мы сегодня рассмотрели не только теоретические моменты нашей темы, но и пробовали решить практические задачи.

Подводя итог по занятию, вспомним, что мы проходили на уроке:

1) Что такое docker-compose?
2) Что такое артефакты?
3) Что Вам больше всего понравилось на уроке?
4) Что Вам меньше всего понравилось на уроке?


Спасибо
за
внимание