

Wprowadzenie do zaawansowanych systemów internetowych; konteneryzacja środowiska deweloperskiego

Projektowanie i programowanie systemów internetowych
II

mgr inż. Krzysztof Rewak

7 października 2020

Wydział Nauk Technicznych i Ekonomicznych
Państwowa Wyższa Szkoła Zawodowa im. Witelona w Legnicy

Plan prezentacji

1. Ramowy plan semestru
2. Warunki zaliczenia kursu
3. Konteneryzacja środowiska deweloperskiego
4. Podsumowanie

Ramowy plan semestru

Planowany rozkład jazdy

Wykład 1: Wprowadzenie; konteneryzacja środowiska deweloperskiego

Wykład 2: Testy jednostkowe i behawioralne

Wykład 3: Reaktywne aplikacje frontendowe

Wykład 4: Projektowanie i tworzenie API

Wykład 5: Architektura sterowana zdarzeniami

Wykład 6: Inne wzorce architektoniczne

Wykład 7: Analiza ruchu w aplikacji webowej

Warunki zaliczenia kursu

W + P

wykład - teoretyczna część kursu; przedstawienie wybranych zagadnień związanych z aplikacjami webowymi;

projekt - praktyczna część kursu; praca zespołowa nad projektowaniem, implementacją oraz wdrożeniem konkretnego systemu internetowego.

Warunki zaliczenia wykładu

Wykład kończy się egzaminem podsumowującym wiedzę przyswojoną w trakcie całego cyklu zajęć. Student, który otrzyma z projektu ocenę niedostateczną nie może podchodzić do egzaminu.

Ponadto na wykładach:

- będzie sprawdzana lista obecności na zasadzie białej listy;
- będzie mierzona (pozytywna i negatywna) aktywność studentów.

Z racji braku zajęć ćwiczeniowych lub laboratoryjnych, zalecane jest uczęszczanie na wykłady.

$$0.3W + 0.7P$$

Ocena niedostateczna z jednej formy rzutuje na ocenę niedostateczną za całość! Przykładowo:

- $3.0 E + 5.0 P = 4.4 \Rightarrow 4.5$
- $4.5 E + 4.0 P = 4.15 \Rightarrow 4.0$
- $2.0 E + 5.0 P = 4.1 \Rightarrow 2.0$

Osoby z oceną z projektu $p \geq 4.5$ zostaną zwolnione z egzaminu z przepisaną otrzymaną oceną.

Wysoka frekwencja oraz aktywność na wykładach mogą rzutować na obniżenie progu przepisywanej oceny do $p \geq 4.0$ dla indywidualnych studentów.

Konteneryzacja środowiska deweloperskiego

Jak wygląda typowe środowisko pracy?

Przykładowo dla aplikacji PHP możemy potrzebować następujących usług:

- PHP 7.1.12
- Apache httpd 2.4.27
- MySQL 5.7.19
- Redis 3.2.100

I oczywiście zainstalowane:

- git
- Composer
- npm

I co dalej?

Można oczywiście zainstalować wszystko osobno, ale:

- instalacja konkretnych wersji zajmie dużo czasu;
- przeniesienie na nową maszynę zajmie ponownie dużo czasu;
- na pewno pojawią się różnice między tymi samymi programami dla Windowsa i uniksów.

I co dalej?

Można oczywiście zainstalować wszystko osobno, ale co w przypadku, gdy kilka projektów będzie potrzebowało różnych wersji tego samego oprogramowania?

Rozwiązanie!

Rozwiązaniem może być **konteneryzacja**.

Kontener (ang. *container*) to w zasadzie osobna instancja środowiska uruchomieniowego z dowolnymi ustawieniami. Każdy kontener powinien charakteryzować się własnymi:

- pamięcią,
- interfejsami sieciowymi,
- wydzielonym obszarem na dysku.

Dla przykładu może to być Ubuntu w wersji 16.04.

Gdzie można taki kontener ustawić? Wszędzie, gdzie tylko się zmieści.
Nieistotne jest to, czy jest to inny linux, Windows czy Mac.

Można też pójść dalej i stworzyć osobne kontenery dla:

- PHP 7.1.12
- Apache httpd 2.4.27
- MySQL 5.7.19
- Redis 3.2.100

Czy czegoś to nie przypomina?

Realizacją idei konteneryzacji zajmuje się **Docker**.

Na następnych slajdach przedstawię przykładową prostą inicjalizację podstawowego środowiska deweloperskiego.



docker

docker-compose.yml

```
version: '2'

volumes:
  database_data:
    driver: local

services:
  nginx:
    image: nginx:latest
    ports:
      - 8080:80
    volumes:
      - ./docker/nginx.conf:/etc/nginx/conf.d/default.conf
    volumes_from:
      - php
```

(gdzie ./docker/nginx.conf to oczywiście plik z konfiguracją serwera)

docker-compose.yml

```
services:
  nginx:
    // (...)

  php:
    build: ./docker/php/
    expose:
      - 9000
    volumes:
      - ./var/www/html
```

gdzie `./docker/php/Dockerfile` może powinien następująco:

```
FROM php:7.0-fpm
RUN docker-php-ext-install pdo_mysql docker-php-ext-install json
```

docker-compose.yml

```
services:
  nginx:
    // (...)

  php:
    // (...)

  mysql:
    image: mysql:latest
    expose:
      - 3306
    volumes:
      - database_data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: secret
      MYSQL_DATABASE: project
      MYSQL_USER: project
      MYSQL_PASSWORD: project
```


`docker-compose.yml`

Pozostaje zbudować kontenery:


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


I *voila!* Pod `localhost:8080` powinniśmy widzieć naszą aplikację.


Docker


 mchiang


Containers [+ NEW](#)


 **hello-world-nginx**
hello-world-nginxlatest


 **minecraft**
minecraftlatest


 **postgres**
postgreslatest

 **redis**
redislatest


 DOCKER CLI








 Search image on Docker Hub


Recommended


**kitematic**
hello-world-nginx
A light-weight nginx container that demonstrates the features of Kitematic
8
[CREATE](#)


**official**
ghost
Ghost is a free and open source blogging platform written in JavaScript
52
[CREATE](#)


**official**
jenkins
Official Jenkins Docker image
421
[CREATE](#)


**official**
redis
Redis is an open source key-value store that functions as a data structure server.
825
[CREATE](#)


**official**
rethinkdb
RethinkDB is an open-source, document database that makes it easy to build and scale realtime...
47
[CREATE](#)


**kitematic**
minecraft
The Minecraft multiplayer server allows two or more players to play Minecraft together
12
[CREATE](#)


**official**
elasticsearch
Elasticsearch is a powerful open source search and analytics engine that makes data easy to...
188
[CREATE](#)


**official**
postgres
The PostgreSQL object-relational database system provides reliability and data integrity.
773
[CREATE](#)


**official**
ubuntu-upstart
Upstart is an event-based replacement for the /sbin/init daemon which starts processes a...
25
[CREATE](#)


**official**
memcached
Free & open source, high-performance, distributed memory object caching system.
55
[CREATE](#)

**official**
rabbitmq
RabbitMQ is a highly reliable enterprise messaging system based on the emerging AMQP...
115
[CREATE](#)

**official**
celery
Celery is an open source asynchronous task queue/job queue based on distributed...
17
[CREATE](#)




**official**
mysql
MySQL is a widely used, open-
[CREATE](#)

**official**
mongo
MongoDB document databases
[CREATE](#)

**official**
mariadb
MariaDB is a community-
[CREATE](#)

Podsumowanie

Bibliografia i ciekawe źródła

-  Krzysztof Rewak, *Projektowanie i programowanie obiektowe*, materiały do zajęć laboratoryjnych
-  Krzysztof Rewak, *Projektowanie i programowanie systemów internetowych I*, slajdy z wykładów
-  <https://www.masterzendframework.com/docker-development-environment/>

Pytania?

Kod prezentacji dostępny jest w repozytorium git pod adresem
<https://github.com/krzysztofrewak/pwsz-inf-resources/tree/ppsi2/wykłady>



Wszystkie informacje dot. kursu dostępne są pod adresem
<http://pwsz.rewak.pl/kursy/6>

