

## Program szkolenia:

# Zaawansowana architektura systemów PHP - projektowanie i implementacja skalowalnych aplikacji webowych

## Informacje:

<b>Nazwa:</b>	<b>Zaawansowana architektura systemów PHP - projektowanie i implementacja skalowalnych aplikacji webowych</b>
<b>Kod:</b>	<b>PHP-arch</b>
<b>Kategoria:</b>	PHP
<b>Odbiorcy:</b>	developerzy
<b>Czas trwania:</b>	3 dni
<b>Forma:</b>	70% wykłady / 30% warsztaty

Szkolenie zostało przygotowane z myślą o programistach i architektach pracujących nad dużymi aplikacjami webowymi typu mission-critical, funkcjonujących w oparciu o klaster redundantnych serwerów.

Szkolenie zawiera szereg praktycznych rozwiązań opracowanych na podstawie doświadczeń w dużych projektach webowych, hostowanych na klastrach wielkości 10-600 fizycznych serwerów.

## Zalety szkolenia:

- Zaingrowane podejście do: skalowania, bezpieczeństwa, failover, CQRS
- Całościowe i szersze spojrzenie na problemy
- Dostęp do wiedzy eksperckiej

## Szczegółowy program:

### 1. Wprowadzenie

1.1. Dlaczego warto zawnazu zadbać o skalowanie aplikacji?

1.2. Historie popularnych witryn internetowych, ich dobre i złe wybory

1.2.1. Zestawienie najczęściej występujących problemów

1.3. Korzyści wynikające z zastosowania architektury umożliwiającej skalowanie

1.4. Skalowalność a wydajność systemu

### 2. Architektura kodu aplikacji

2.1. Backend i frontend aplikacji

2.1.1. Separowalność i specjalizacja poszczególnych części aplikacji

2.1.2. Komunikacja

2.2. Nowoczesne frameworki PHP

2.2.1. Symfony

2.2.2. Laravel

2.2.3. Zend

2.3. Nowoczesne frameworki JavaScript

2.3.1. Backbone

2.3.2. EmberJS

2.3.3. AngularJS

2.4. Organizacja kodu

2.4.1. Architektura MVC

2.4.2. Moduły oprogramowania

2.4.2.1. Możliwości i zastosowanie Comosera

2.4.3. Wzorce projektowe

2.4.4. Serwisy usługowe i Dependency Injection

2.4.5. Mapowanie relacyjno-objektowe

2.4.5.1. Przegląd dostępnych rozwiązań

2.4.5.2. Ograniczenia narzędzi ORM

2.4.6. Command-query Responsibility Segregation

2.4.6.1. Stos Command

2.4.6.2. API dla modelu biznesowego

2.4.6.3. Stos Read

2.4.6.4. Optymalizacja pod kątem dużej ilości odczytów

2.4.6.5. Synchronizacja stosów

2.4.6.6. SQL, widoki, zdarzenia biznesowe

2.4.7. Struktura monolityczna vs architektura mikroserwisów

2.4.7.1. Case studies projektów migrujących pomiędzy stylami, przyczyny i osiągnięte rezultaty

### 3. Architektura systemu

3.1. Środowisko uruchomieniowe aplikacji, kod źródłowy to nie wszystko

3.2. Typowa ścieżka skalowania aplikacji

3.2.1. Środowisko single-server jako punkt startowy

3.2.2. Rozproszenie usług w pojedynczych instancjach

3.2.3. Rozproszenie usług w wielu instancjach

3.2.4. Przegląd typowych problemów

3.3. Redundancja usług dla wysokiej dostępności

3.3.1. Kierowanie ruchem pomiędzy redundantnymi serwerami

3.3.2. Obsługa sytuacji krytycznych, failover

3.3.3. Wpływ redundancji na kod aplikacji

3.4. Eliminacja obciążeń

### 3.4.1. Przyspieszanie aplikacji PHP

#### 3.4.1.1. Opcode cache

#### 3.4.1.2. Buildy aplikacji

### 3.4.2. Cache'owanie danych

#### 3.4.2.1. Wprowadzenie do cache'owania

#### 3.4.2.2. Tworzenie i inwalidacja cache

#### 3.4.2.3. Memcached

#### 3.4.2.4. Techniki wykorzystania dedykowanych narzędzi w celu odciążenia warstwy bazy danych

### 3.4.3. Cache HTTP

#### 3.4.3.1. Wprowadzenie do cache'owania

#### 3.4.3.2. Varnish jako web-application accelerator

### 3.4.4. Serwery Full-Text-Search

#### 3.4.4.1. Problem wyszukiwania tekstu w relacyjnych bazach danych

#### 3.4.4.2. Sphinx jako serwer FTS

#### 3.4.4.3. Podstawowa konfiguracji serwera Sphinx i indeksacja źródeł danych

#### 3.4.4.4. Flow komunikacji pomiędzy aplikacją a serwerem Sphinx

#### 3.4.4.5. Metody indeksacji dużych źródeł danych

#### 3.4.4.6. Shardowanie indeksów

## 3.5. Skalowanie warstwy bazy danych

### 3.5.1. Modele replikacji

#### 3.5.1.1. Master-slave(s)

#### 3.5.1.2. Master-master

#### 3.5.1.3. Problemy aplikacyjne wynikające z replikacji

#### 3.5.1.4. Zakleszczenia

#### 3.5.1.5. Replication-lag

### 3.5.2. Shardowanie tabel

#### 3.5.2.1. Wprowadzenie do koncepcji shardowania

#### 3.5.2.2. Korzyści i ograniczenia

#### 3.5.2.3. Techniki podziału baz danych

#### 3.5.2.4. Konsekwencje dla warstwy dostępu do danych w aplikacji

### 3.5.3. Rozwiązania NoSQL

#### 3.5.3.1. CAP theorem

#### 3.5.3.2. Przegląd dostępnych rozwiązań i ich możliwości

## 3.6. Systemy kolejkowe

### 3.6.1. Wprowadzenie do systemów kolejkowych

### 3.6.2. Wykorzystanie systemów kolejkowych do rozdzielania systemów

### 3.6.3. Narzędzia

#### 3.6.3.1. AMQP

#### 3.6.3.2. RabbitMQ

## 3.7. Deployment aplikacji na klaster serwerowy

### 3.7.1. Różnice pomiędzy deploymentem w ramach pojedynczego serwera a klastrem

### 3.7.2. Konsekwencje deploymentu systemów wielowarstwowych

## 3.8. Monitoring

### 3.8.1. Monitoring pracy infrastruktury serwerowej

### 3.8.2. Monitoring pracy aplikacji

### 3.8.3. Narzędzia

## 3.9. Infrastruktura serwerowa

### 3.9.1. Cloud computing vs infrastruktura dedykowana

### 3.9.2. Narzędzia developerskie / devops

3.9.2.1. Vagrant

3.9.2.2. Docker

#### 4. Case-studies realnych projektów

4.1. Shardowane bazy danych

4.2. Skalowalne storage plików

4.3. Skalowalne systemy kolejkowe

4.4. Skalowalne wyszukiwanie full-text-search w dużym datasecie

4.5. Wykorzystanie systemów NoSQL w celu zwiększenia skalowalności i wydajności aplikacji