Ratpack - praktyczne wprowadzenie

- Kod źródłowy: https://github.com/wololock/ratpack-quickstart-demo
- Handout: https://github.com/wololock/ratpack-quickstart-demo/handouts_pl.adoc)_

Ratpack (https://ratpack.io) - lekki framework dla Java 8 oparty o Netty (4.1.22 w najnowszej wersji) do tworzenia lekkich, asynchronicznych i nieblokujących aplikacji HTTP.

> *Ciekawostka: nazwa Ratpack nawiązuje do Rat Pack - nieformalnej grupy muzycznej z lat 50 XX w. w której występował m.in Frank Sinatra. Inspiracją dla tego frameworka był Sinatra (http://sinatrarb.com/) - lekki framework dla języka Ruby.*

1. Jak zacząć?

- SDKMAN! (http://sdkman.io) przyda się do instalacji Lazybones:

```
sdk install lazybones
```

- Lazybones (https://github.com/pledbrook/lazybones) ułatwi stworzenie projektu z szablonu:

```
lazybones create ratpack [nazwa_projektu]
```

- Aplikację uruchomić możemy za pomocą Gradle (tryb continuous `-t` pozwoli nam hot swapować i restartować aplikację po zapisaniu zmian w plikach źródłowych):

```
./gradlew run -t
```

- Domyślnie Ratpack korzysta z portu `5050`. HTTPie (https://httpie.org/) ułatwi nam wykonywanie requestów do aplikacji:

```
http localhost:5050
```

2. Architektura aplikacji

- Sugerowanym narzędziem do budowania aplikacji w przypadku Ratpacka jest Gradle (https://gradle.org/)
- Można użyć Maven'a jako alternatywy, niemniej `pom.xml` trzeba będzie skonfigurować samemu - nie ma archetypu dla Ratpacka
- Ratpack dostarczany jest w formie modułowej - rozszerzenie aplikacji o nowe możliwości rozwiązuje się poprzez dodanie nowej zależności (np. `ratpack-rx` dla dodania wsparcia RxJava)

do `build.gradle` oraz właściwe skonfigurowanie serwera (Ratpack nie implementuje autokonfiguracji)

- Ratpack wykorzystuje `io.netty.channel.EventLoop` do wykonywania operacji asynchronicznych
- Domyślnie dostępne są dwie pule wątków: `ratpack-compute` przeznaczona do przetwarzania żądań HTTP oraz wykonywania obliczeń oraz `ratpack-blocking` do wykonywania operacji blokujących (m.in. I/O, JDBC itd.)
- Pula `ratpack-compute` domyślnie posiada ilość wątków obliczoną wg wzoru: `2 * ilość dostępnych procesorów/rdzeni`
- Pula `ratpack-blocking` to nic innego jak domyślny cached thread pool (60 sekund keep alive time, praktycznie nieograniczona ilość wątków w puli - `Integer.MAX_VALUE`)
- Podstawową klasą do wykonywania operacji asynchronicznych jest `ratpack.exec.Promise<T>` (klasa zbliżona funkcjonalnością do `CompletableFuture<T>` czy `Observable<T>` z RxJava)
- Operacje asynchronicznie możemy wywołać m.in. za pomocą `Promise.async(Upstream<T> upstream)`.
- Operacje blokujące delegujemy do puli wątków blokujących za pomocą `Blocking.get(Factory<T> factory)`
- Ratpack buduje się do "standalone JAR", który podobnie jak np. Spring Boot uruchomimy za pomocą `java -jar ratpack-application.jar`
- Dzięki temu możemy np. "zdockeryzować" naszą aplikację i uruchamiać ją w chmurze

3. Dlaczego Ratpack?

- Ratpack sprawdzi się idealnie w architekturze mikroserwisowej
- Nieblokujące przetwarzanie requestów HTTP pozwala na bardziej efektywną utylizację zasobów (w opozycji do serwletowego "one request = one thread")
- Ratpack zmusza nas do wyrażania explicite naszych oczekiwań względem aplikacji - nie ma tutaj miejsca na classpath scanning, annotation-driven development
- Ratpack nie próbuje być najszybszym frameworkiem dla JVM - zamiast tego koncentruje się na możliwie najbardziej optymalnym wykorzystaniu zasobów (CPU, mem)
- Rozwiązanie to sprawdzi się bardzo dobrze np. w przypadku produktu startupowego - zamiast stawiać dużej i zasobożernej aplikacji można wystartować z produktem, który uruchomi się skutecznie na maszynie z mniejszą ilością RAMu i CPU

4. Argumenty przeciwko Ratpackowi

- Ratpack nie jest frameworkiem typu Enterprise (brak supportu na poziomie enterprise, brak dużej firmy stojącej za rozwojem produktu)
- Ilość dostępnych integracji czy modułów jest wielokrotnie mniejsza w porównaniu do np. Spring Boota
- Na rynku istnieją inne rozwiązania oferujące architekturę asynchroniczną i nieblokującą

5. Alternatywy

- Reactor (https://projectreactor.io/)
- Vert.x (http://vertx.io/)
- Akka HTTP (https://doc.akka.io/docs/akka-http/current/index.html)

6. Przydatne narzędzia

- Siege (https://github.com/JoeDog/siege)

```
siege -c 100 -r 1 http://localhost:5050/recommendations
```

- Apache Benchmark (https://httpd.apache.org/docs/2.4/programs/ab.html)

```
ab -c 100 -n 2000 http://localhost:5050/recommendations
```

7. Więcej na temat Ratpack

- "Learning Ratpack" (2016), Dan Woods, wyd. O'Reilly - *książka jest świetnym startem w świat Ratpacka. Omawia ona większość możliwośći oraz wskazuje miejsca, w których można dowiedzieć się więcej*
- Prezentacja "Mastering Async Ratpack" (https://danhyun.github.io/mastering-async-ratpack/) - *omówienie asynchroniczności na przykładach. Na YouTube można znaleźć video - warto obejrzeć!*
- Prezentacja "Ratpack - Future(Server.HTTP(Java))" Jarka Ratajskiego (https://www.youtube.com/watch?v=EWmi10tTnpw) - *Ratpack w wersji Java 8 - sporo o asynchroniczności i non-blocking na przykładzie implementacji ciągu fibonacciego i popularnej gry Pong*
- Prezentacja "Ratpack: The Story So Far (Phill Barber)" (https://www.youtube.com/watch?v=q2vADSqi6XI) - *przykład wdrożenia Ratpacka w brytysjkim Sky + integracja z RxJava*
- Wpis na blogu Luke'a Daley'a (twórcy Ratpacka) o async execution model - http://ldaley.com/post/97376696242/ratpack-execution-model-part-1

8. Podziękowania

Dziękuję za wysłuchanie i udział w dzisiejszej prezentacji. Mam nadzieję, że przedstawiona treść jest dla Ciebie wartościowa i czegoś ciekawego się dzisiaj nauczyłeś(-aś). Jestem bardzo ciekaw Twojej opinii na temat tej prezentacji – podziel się nią ze mną proszę. Znajdziesz mnie na Twitterze (https://twitter.com/wololock) lub pod adresem e-mail szymon.stepniak@gmail.com Zachęcam również do kontaktu jeśli masz jakieś pytanie, na które dzisiaj nie uzyskałeś(-aś) odpowiedzi – z przyjemnością na nie odpowiem. Dzięki raz jeszcze i do zobaczenia następnym razem :-)

Szymon Stępniak