Bootiful Spring Boot 3

Date : 14 avril 2023
Speakers : Josh Long (Pivotal)
Format : Conférence (45mn)
Repo de la demo : https://github.com/joshlong/bootiful-spring-boot-3

Josh est heureux d'être parmi nous car c'est sa première venue à Devoxx France depuis le Covid.
Très enthousiaste dans la sortie de Spring Boot 3 sorti lors de la fête de Thanksgiving 2022 qui est férié aux USA. Une grosse release avec pleins de changements.
Notre Java Champion rappelle l'importance d'écrire du code qui part en prod.
Josh commence par générer une application depuis start.spring.io qui, au passage, configure par défaut l'outil de build Gradle et non plus Maven.
Version de Java possible : Java 17 minimum ou + pour Spring Boot 3. Java 8 et 11 sont trop vieux pour la production. Nécessité d'avancer sans nostalgie.

Cette conférence sera sans slide et Josh enchainera des démos live dans son IDE préféré : IntelliJ.
Première démo avec l'utilisation d'un **record** Customer sans Lombok, d'un DAO avec *JdbcTemplate* et d'un *RowMapper* à l'ancienne.

```
@Service
class CustomerService {

    private final JdbcTemplate template;

    private final RowMapper<Customer> customerRowMapper = new RowMapper<Customer>() {

        @Override
        public Customer mapRow(ResultSet rs, int rowNum) throws SQLException {
            return new Customer(rs.getInt("id"), rs.getString("name"));
        }
    };

    CustomerService(JdbcTemplate template) {
        this.template = template;
    }

    Collection<Customer> byName(String name) {
        return this.template
                .query("select * from customer where name  = ? ", this.customerRowMapper, name);
    }

    Collection<Customer> all() {
        return this.template.query("select * from customer", this.customerRowMapper);
    }
}

record Customer(Integer id, String name) {
}
```

Déclaration d'un bean de type ApplicationRunner affichant les 4 customers.
Création d'un *RestController*.
Ajout de validation avec un *Assert.state(Character.isUpperCase(name.charAt(0), …)*

Création d'un *@ControllerAdvice* et d'un *@ExceptionHandler*
Montre l'usage natif de la **RFC Problem Details** introduit dans Spring Framework 6 pour remonter les erreurs (plus besoin de la lib Zalando Problem).

```
@ControllerAdvice
class ErrorHandlingControllerAdvice {

    @ExceptionHandler
    ProblemDetail handle(IllegalArgumentException iae ) {
        return ProblemDetail
                .forStatusAndDetail(HttpStatusCode.valueOf(503) ,"the name is invalid");
    }
}
```

Josh montre au passage que Spring Boot 3 utilise désormais le package **jakarta.servlet.http** et non plus javax.servlet.http.
L'équipe de Spring a travaillé longtemps avec la communauté pour que tous les projets supportés par Spring Boot basculent sur le package jakarta. Spring Boot 3 est basé sur Jakarta EE 9 et supporte le 10.

Support de l'observabilité : métriques et traces.
En SB2 on utilisait Sleuth pour le tracing et micrometer pour les métriques.
SB3 utilise désormais Micrometer pour les traces et les métriques
Ajout d'un ObservationRegistry dans le contrôleur REST

```
    @GetMapping("/customers/{name}")
    Collection<Customer> byName(@PathVariable String name) {
        if (name == null && Character.isUpperCase(name.charAt(0)))
            throw new IllegalArgumentException("the name must be valid");

        return Observation
                .createNotStarted("byName", this.registry)
                .observe(() -> this.service.byName(name));
    }
```

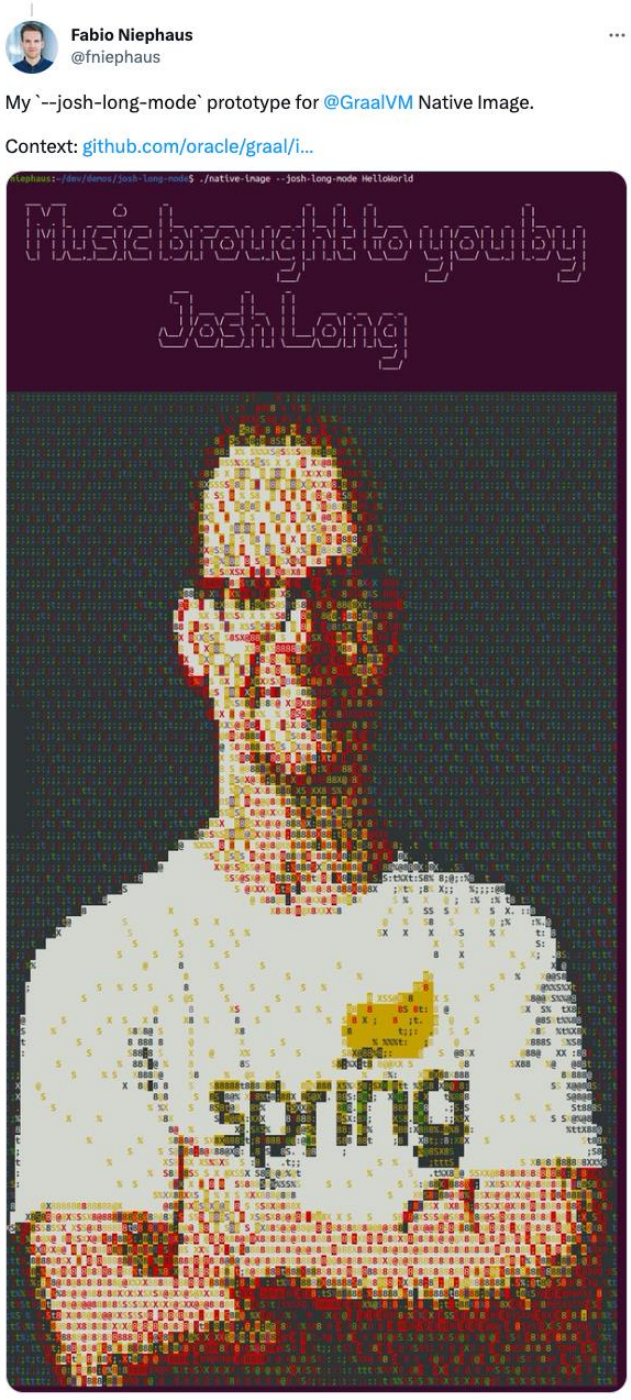
Une fois activité l'actuator /health/metrics affiche des stats d'usage sur l'API REST

SB3 utilise Buildpacks (CNCF) pour construire l'image Docker.

Simple commande pour créer l'image :
gradlew buildBootImage

Josh est fier d'annoncer que Java arrive 4ieme dans une étude classant les langages de

programmation par leur consommation énergétique.
Java a un Amazing runtime, et notamment le JIT qui nécessite un warmup.
Le compilateur GraalVM demande beaucoup de configuration.
L'équipe Spring Boot 3 l'a réduit au maximum.
La commande *gradle buildNative* est très longue à s'exécuter. Josh nous explique entendre dans sa tête la musique d'un ascenseur lorsqu'il fait ces compilations natives. Il va jusqu'à nous montrer l'issue #5327 ouverte sur le repo de Graal et qu'un chercheur de l'équipe de Graal a implémenté.



L'application native occupe en mémoire 1 Mo de RAM, et ceci alors qu'elle embarque un Tomcat.

Josh poursuit sa démo en démarrant une nouvelle appli visant à consommer l'API REST créée précédemment.

Il créer une interface *CustomerClient* et utilise la nouvelle annotation **@GetExchange** permettant de générer un client REST moyennant un peu de plomberie.

```
interface CustomerClient {

    @GetExchange("/customers")
    Flux<Customer> all();

    @GetExchange("/customers/{name}")
    Flux<Customer> byName(@PathVariable String name);
}

record Customer(Integer id, String name) {
}

 @Bean
 CustomerClient client(WebClient.Builder builder) {
     return HttpServiceProxyFactory
             .builder(WebClientAdapter.forClient(builder.baseUrl("http://localhost:8080/").build()))
             .build()
             .createClient(CustomerClient.class);
 }
```

Il configure ensuite Spring Gateway pour router et filtre des requêtes.

```
@Bean
RouteLocator gateway(RouteLocatorBuilder rlb) {
    return rlb
            .routes()
            .route(rs -> rs.path("/proxy")
                    .filters(f -> f.setPath("/customers"))
                    .uri("http://localhost:8080/"))
            .build();
}
```

Josh poursuit par une démo du projet Spring for GraphQL. Josh écrit un schéma GraphQL. L'annotation **@QueryMapping** est ajoutée au contrôleur.

```java
@Controller
@ResponseBody
@ImportRuntimeHints(CustomerHints.class)
class CustomerGraphqlController {

    final CustomerClient client;

    CustomerGraphqlController(CustomerClient client) {
        this.client = client;
    }

    @QueryMapping
    Flux<Customer> customers() {
        return this.client.all();
    }

    @QueryMapping
    Flux<Customer> customersByName(@Argument String name) {
        return this.client.byName(name);
    }
}
```

Josh ajoute la notion de Profile et utilise conjointement **@BatchMapping** (à la place de @SchemaMapping) pour éviter l'effet du N+1 requêtes.