Micronaut, le framework JVM ultra-light du futur

Speaker : Olivier Revial (Stack Labs)
Format : Tools in action
Date : 17 avril 2019

Dans ce Tools in Action, Olivier introduit Micronaut comme un framework récent, très moderne et ultra-light.
Bref historique : Graem Rocher a créé Micronaut avec l'aide de contributeurs du framework Spring. Pour rappel, Graeem est le fondateur de Grails (créé en 2006).
La version 1.0.0 de Micronaut date d'août 2018.
La version 1.1 est sortie en avril 2019.

Les créateurs de Micronaut ont pris le meilleur de Spring, Grails, Spring Boot et autre :

- IoC
- Auto-configuration
- Service discovery (comme le font Eureka, Kubernetes et ZooKeeper)
- HTTP Routing + Clients : quelques lignes de code pour exposer et consommer des API REST
- Supporte les langages Kotlin, Groovy et Java.

Micronaut essaie de ne pas reproduire les mêmes erreurs que les frameworks pré-cités :

- Réflexion et usage de proxies => temps de démarrage et empreinte mémoire conséquents. Micronaut essaie d'avoir un temps de démarrage et une empreinte mémoire réduits.
- Lorsque les tests fonctionnels ou d'intégration nécessitent de démarrer le serveur, ils ne sont pas inclus dans le build CI classique du mvn clean install. Avec une application qui se lance en 1 seconde, on peut laisser les tests d'intégration à côté des tests unitaires

Comment fonctionne Micronaut ?

Micronaut essaie de faire un maximum de choses à la compilation (pas de réflexion).

- Java et Kotlin : Micronaut utilise les processeurs d'annotations
- Groovy : passe par des AST transformations

Cela permet d'améliorer le temps de démarrage. Cela permet d'aller encore plus loin et de générer du code natif via GraalVM.

Démo

Olivier poursuit sa présentation par une démo.

Commande : *mn profile-info service*
Permet d'afficher les différents modules supportés par Micronaut : swagger, spock, rabbitmq, zipkin …

Création d'une application simple et d'un contrôleur REST :

> Hello Devoxx Controller

```
@Controller("/hello")
class HelloDevoxxController {

    @Get("/simple")
    fun helloSimple(): String {
        return "Hello Devoxx 2019 !"
    }

    @Get("/reactive")
    fun helloReactive(): Flowable<String> {
        return Flowable.intervalRange(0, 20, 1, 1, TimeUnit.SECONDS)
                .map { i -> """Hello Devoxx n°$i""" }
    }
}
```

Le test unitaire associé est écrit avec le framework de test Spoke.
Le build est un peu long (notamment le premier avec Gradle) : 20 secondes.
Le temps de démarrage varie entre 2 et 6 secondes (secteur vs batterie ?).

On bascule sur une autre application démo attaquant l'API Twitter.

Le bean MongoService est de portée singleton.
Passé dans le constructeur, MongoClient est un bean injecté via l'auto-configuration.

> MongoService

```
@Singleton
class MongoService(private val mongo: MongoClient) {
    fun list(total: Int): Flowable<Tweet> =
            Flowable.fromPublisher<Tweet>(getCollection().find().limit(total))

    fun save(tweets: List<Tweet>): Single<String> {
        LOG.info("""Saving ${tweets.size} tweets in Mongo...""")
        return Single.fromPublisher<Success>(getCollection().insertMany(tweets))
                .map { """Successfuly saved ${tweets.size} tweets to Mongo..."""" }
    }

    fun deleteAll(): Single<String> {
        LOG.info("""Deleting entire  $TWEETS_COLLECTION collection from Mongo...""")
        return Single.fromPublisher<Success>(getCollection().drop())
                .map { "Successfuly deleted entire $TWEETS_COLLECTION from Mongo..." }
    }

    private fun getCollection(): MongoCollection<Tweet> =
        mongo
            .getDatabase("micronaut")
            .getCollection<Tweet>(TWEETS_COLLECTION, Tweet::class.java)
}
```

Dans le bean TweetController, sont utilisés les annotations Bean Validation. Micronaut supporte Hibernate Validator.

> TweetController

```
@Validated
@Controller("/tweets")
class TweetController(private val mongoService: MongoService) {

    @Get("/")
    fun list(@NotNull @Min(1) @Max(100) total: Int?): Flowable<Tweet> {
        return mongoService.list(total)
    }
}
```

L’annotation @Scheduled permet de planifier l’appel à des fonctions (comme dans Spring).

Micronaut fournit des métriques sur le endoint /metrics

Par défaut, Micronaut supporte le mécanisme de sécurité Basic Auth.

Le build pour GraalVM dure plusieurs minutes. Le binaire fait 56 Mo.
Le temps de démarrage oscille entre 15 et 200 ms.

Conclusion

- Framework très intéressant
- Encore jeune, il grandit très vite
- Pensé nativement pour les conteneurs
- On peut déployer Java sur du Serverless
- Manque encore quelques composants
- Certaines dépendances ne supportent pas (encore) GraalVM

Mot de la fin : « **Micronaut a incité Redhat a créé Quarkus et pousse Pivotal a optimisé Spring Boot. Toute la communauté Java en profitera.** »