

## Avant-propos

<b>1. Préambule</b>	<b>11</b>
<b>2. Introduction</b>	<b>15</b>
<b>3. Organisation du livre</b>	<b>16</b>
<b>4. Public visé</b>	<b>16</b>
<b>5. Pourquoi Spring ?</b>	<b>17</b>
<b>6. Prérequis pour aborder Spring et Java EE</b>	<b>17</b>

## Architectures alternatives

<b>1. Évolution des architectures</b>	<b>19</b>
<b>2. Domain-Driven Design</b>	<b>30</b>
<b>3. Event storming</b>	<b>33</b>
<b>4. Architecture hexagonale</b>	<b>34</b>
<b>5. Cœur de métier</b>	<b>35</b>

5.1 Cas d'utilisation	36
5.2 Couche de ports et d'adaptateurs	36
5.3 Repository	37
5.4 Couche d'anticorruption	37
5.5 Sens des dépendances	37
<b>6. Découpage de l'hexagone sur deux axes</b>	<b>39</b>
6.1 Axe vertical	39
6.2 Axe horizontal	39
<b>7. Respecter le modèle</b>	<b>40</b>
<b>8. Hexagones multiples</b>	<b>40</b>
<b>9. Modélisation événementielle et event sourcing</b>	<b>42</b>
<b>10. Event store</b>	<b>42</b>
<b>11. Reconstitution de l'état</b>	<b>42</b>
11.1 Dimension temporelle	43
11.2 Problèmes	43
11.3 Avantages	44
11.4 Rebranchements	44

11.5 Agrégats	44
---------------	----

<b>12. Points clés</b>	<b>45</b>
------------------------	-----------

## ZooKeeper et Kafka

<b>1. Apache ZooKeeper</b>	<b>47</b>
----------------------------	-----------

1.1 Installation de ZooKeeper	49
-------------------------------	----

1.2 Lancement de ZooKeeper	50
----------------------------	----

1.3 Connexion à ZooKeeper en ligne de commande	50
--	----

1.4 Mode répliqué	53
-------------------	----

1.5 Arrêt de ZooKeeper	53
------------------------	----

1.6 Utilisation de ZooKeeper dans un programme Spring Boot	54
--	----

1.7 Implémentations types	60
---------------------------	----

<b>2. Librairie Curator</b>	<b>61</b>
-----------------------------	-----------

<b>3. Apache Kafka</b>	<b>64</b>
------------------------	-----------

3.1 Système de messagerie	65
---------------------------	----

3.1.1 Système point à point	66
-----------------------------	----

3.1.2 Système publication-abonnement	66
--------------------------------------	----

3.2 Particularités de Kafka	67
-----------------------------	----

3.3 Installation de Kafka	69
3.4 Test en ligne de commande	70
3.5 Fonctionnalités de Kafka	71
3.5.1 Dépendances Maven	71
3.5.2 Exemple Spring Boot	72
3.6 Avro, un système de sérialisation de données	75

## Programmation fonctionnelle et streams

<b>1. Introduction</b>	<b>85</b>
<b>2. Streams</b>	<b>87</b>
<b>3. Exemple de programmation fonctionnelle en Java</b>	<b>90</b>
<b>4. Bibliothèque Vavr</b>	<b>90</b>
<b>5. Pour aller plus loin</b>	<b>92</b>
<b>6. Points clés</b>	<b>93</b>

## Programmation réseau asynchrone avec Netty

<b>1. Introduction</b>	
------------------------	--

	<b>95</b>
<b>2. Éléments de l'architecture de Netty</b>	<b>97</b>
2.1 Canal (channel)	97
2.2 Interfaces Future et ChannelFuture	97
2.3 Événements et handlers	98
2.4 Encodeurs et décodeurs	99
2.5 Serveur	99
<b>3. Exemples</b>	<b>100</b>
3.1 Serveurs et clients simples	100
3.2 Client Netty	106
<b>4. Transferts basés sur des streams</b>	<b>108</b>
<b>5. Arrêt de l'application</b>	<b>111</b>
<b>6. Pour aller plus loin</b>	<b>111</b>
<b>7. Points clés</b>	<b>111</b>
Programmation réactive	
<b>1. Flux réactifs (reactive streams)</b>	<b>113</b>

<b>2. Programmation asynchrone historique</b>	<b>116</b>
<b>3. API réactives</b>	<b>117</b>
3.1 Akka	117
3.2 Modèle d'acteur	118
3.3 Configurer un projet Akka	119
3.4 Créer un acteur	120
3.5 Configurer un acteur	121
3.6 Interactions et communications entre acteurs	121
3.6.1 Envoyer des messages	122
3.6.2 Recevoir des messages	123
3.7 Tuer un acteur	123
3.8 Bonnes pratiques avec Akka	124
3.9 Spring et Akka	124
3.9.1 Dépendances Maven	124
3.9.2 Récupérer des acteurs gérés par Spring	128
3.10 Utilisation de Akka Streams	129
3.10.1 Dépendances Maven	129
3.10.2 Créer un flux Akka	130
<b>4. RxJava 1.3.4</b>	<b>131</b>

4.1 Observateurs et observables	132
4.2 Opérations sur les observables	136
4.2.1 Opérateurs de création	136
4.2.2 Opérateurs de transformation	137
4.2.3 Opérateurs de filtrage	138
4.2.4 Combiner des observables	139
4.2.5 Opérateurs de gestion d'erreurs	140
4.2.6 Opérateurs de services utilitaires	140
4.2.7 Opérateurs conditionnels et booléens	141
4.2.8 Opérateurs mathématiques et agrégats	142
4.2.9 Conversion d'observables	143
4.2.10 Opérateurs de connexion	143
4.2.11 Opérateurs de contre-pression	143
4.2.12 Exemples d'utilisation d'opérateurs	144
4.2.13 Chaînage des opérateurs	145
4.2.14 Spécificités de l'observable Single	145
4.3 Sujets (subject)	148
4.3.1 AsyncSubject	148
4.3.2 BehaviorSubject	149
4.3.3 ReplaySubject	149
4.4 Schedulers	149

4.4.1 Schedulers avec RxJava 1.x	151
<b>5. Implémentation réactive avec Vert.x</b>	<b>153</b>
5.1 Vert.x avec RxJava	155
5.2 Support des résultats asynchrones	157
<b>6. Prise en charge du scheduler</b>	<b>160</b>
<b>7. JSON unmarshalling</b>	<b>161</b>
<b>8. Déploiement d'un verticle</b>	<b>162</b>
<b>9. HttpClient GET sur un abonnement</b>	<b>162</b>
<b>10. API Rx-ified</b>	<b>163</b>
10.1 Intégration de Rx-ified Vert.x	163
<b>11. Exemples d'API</b>	<b>163</b>
11.1 Timers	164
11.2 Requêtes de client HTTP	165
11.3 Requêtes du serveur HTTP	166
<b>12. Client WebSocket</b>	<b>167</b>
<b>13. Serveur WebSocket</b>	

	<b>168</b>
<b>14. Reactor</b>	<b>169</b>
<b>15. Mono et Flux</b>	<b>172</b>
15.1 Opérations sur les observables	172
15.1.1 Opérateurs de création de Flux	173
15.1.2 Opérateurs de transformation des éléments observables	174
15.1.3 Opérateurs de filtrage des observables	175
15.1.4 Opérateurs de combinaison des observables	176
15.1.5 Opérateurs de gestion d'erreurs	176
15.1.6 Opérateurs de services utilitaires	177
15.1.7 Opérateurs conditionnels	178
15.1.8 Opérateurs mathématiques	179
15.1.9 Opérateurs de conversion d'observables	180
15.1.10 Opérateurs de connexion	180
15.1.11 Spécificités de l'observable Mono	181
<b>16. Contre-pression</b>	<b>183</b>
<b>17. Chaud ou froid</b>	<b>183</b>
17.1 Reactor-test	184
17.2 Tester un scénario avec StepVerifier	185

17.3 Identifier les tests en erreur	186
17.4 Manipuler du temps	187
17.5 Exécuter des assertions post-exécution avec StepVerifier	189
17.6 Tester le Context	189
17.7 Émettre manuellement avec TestPublisher	190
17.8 Vérifier le chemin d'exécution	191
17.9 Réacteur de débogage	193
17.10 Conclusion sur Reactor	193
<b>18. Conclusion sur la programmation réactive</b>	<b>194</b>
Exemples d'applications avec une base SQL	
<b>1. Introduction</b>	<b>195</b>
<b>2. Isoler la partie JDBC</b>	<b>196</b>
<b>3. Adapter un pilote JDBC réactif à Reactor</b>	<b>197</b>
3.1 Solution avec rxjava2-jdbc	198
3.2 Solution avec vertx-jdbc-client	198
3.2.1 Créer un client	199
3.2.2 Obtenir une connexion	200

3.2.3 Configuration	201
3.2.4 Types de données	203
<b>4. Conclusion sur les pilotes JDBC asynchrones</b>	<b>205</b>
<b>5. Pour aller plus loin</b>	<b>205</b>
<b>6. Points clés</b>	<b>206</b>
Librairie Reactor Kafka	
<b>1. Introduction</b>	<b>207</b>
<b>2. Producteur</b>	<b>208</b>
<b>3. Consommateur</b>	<b>210</b>
<b>4. Scénarios</b>	<b>211</b>
4.1 Flux d'entrée	211
4.2 Flux de sortie	212
4.3 Transformations internes	212
<b>5. Points clés</b>	<b>213</b>

## Applications web avec Spring WebFlux

<b>1. Introduction</b>	<b>215</b>
1.1 Types de endpoints	218
<b>2. Quand privilégier l'utilisation de Spring WebFlux ?</b>	<b>220</b>
<b>3. Serveurs Spring WebFlux</b>	<b>221</b>
<b>4. Organisation du framework réactif</b>	<b>221</b>
4.1 HttpHandler et adaptateurs de serveur	221
4.2 Initialisation d'un HttpHandler	223
<b>5. HttpHandler des différentes implémentations</b>	<b>224</b>
5.1 Reactor Netty	224
5.2 Undertow	224
5.3 Jetty	225
5.4 Tomcat	225
<b>6. Déploiement d'un war</b>	<b>226</b>
6.1 API WebHandler	228
6.2 Beans Spring spécifiques	228

<b>7. DispatcherHandler</b>	<b>232</b>
7.1 Beans spéciaux pour le DispatcherHandler	233
7.2 Configuration WebFlux	234
7.3 Traitement d'un appel	234
7.4 Gestion des valeurs de retour	234
<b>8. Exemple de service REST réactif avec WebFlux et la version réactive de MongoDB</b>	<b>236</b>
<b>9. Endpoints fonctionnels</b>	<b>242</b>
9.1 HandlerFunction	242
9.2 RouterFunction	245
9.2.1 Exécuter une fonction sur un serveur	247
9.2.2 HandlerFilterFunction	248
<b>10. Tests spécifiques à WebFlux</b>	<b>249</b>
10.1 Spring Security pour les méthodes réactives	249
10.2 WebTestClient	250
10.3 Tests d'authentification	251
<b>11. Tests sans Spring</b>	<b>252</b>
<b>12. WebSocket</b>	<b>254</b>
12.1 HTTP ou WebSocket	

12.2 Quand utiliser WebSocket avec WebFlux ?	255
12.3 API WebSocket	256
12.3.1 WebSocketHandler	257
12.3.2 WebSocket Handshake	257
12.3.3 Configuration du serveur	258
12.4 WebSocketClient	258
	259

## Microservices Docker et DevOps

<b>1. Introduction</b>	<b>261</b>
<b>2. Docker et machines virtuelles</b>	<b>263</b>
<b>3. Spring Boot avec Docker</b>	<b>264</b>
3.1 Utilisation de profils de Spring	268
3.2 Débogage de l'application dans un conteneur Docker	268
<b>4. Conteneurs et écosystème Spring Boot</b>	<b>269</b>
<b>5. Limites des conteneurs</b>	<b>269</b>
<b>6. Pour aller plus loin</b>	<b>270</b>
<b>7. Points clés</b>	

**270**

## Exemple utilisant le générateur JHipster

### 1. Introduction

**271**

### 2. Génération d'une application avec Kafka

**271**

### 3. Points clés

**277**

## Microservices et applications dans le cloud

### 1. Introduction

**279**

### 2. Spring Cloud

**282**

### 3. Spring Serverless

**285**

#### 3.1 Fonctionnement du FaaS

**285**

#### 3.2 Limitations du FaaS

**286**

### 4. Spring Function

**287**

### 5. Pour aller plus loin

**290**

### 6. Points clés

**291**

**Index**

**293**