

# Spring Boot : Modernisation des Applications Java

## Pourquoi Spring Boot ?

¥ Problématique historique de Spring :

- ! Configuration manuelle complexe (XML, JavaConfig).
- ! Dépendances conflictuelles et gestion fastidieuse.
- ! Temps de démarrage élevé pour les microservices.

## Pourquoi Spring Boot ?

¥ Solution Spring Boot :

- ! Convention over Configuration : Configuration automatique basée sur les dépendances.
- ! Embedded Server (Tomcat/Jetty) pour des déploiements légers.
- ! Starters pour bundle de dépendances prêts à l'emploi (ex: `spring-boot-starter-web` inclut Spring MVC + Tomcat):cite[7].

## Philosophie de Spring Boot

¥ Productivité accrue :

- ! Réduction de 70% du code boilerplate:cite[5].
- ! Exemple : Une API REST en 1 classe Java:cite[5].

## Philosophie de Spring Boot

¥ Prêt pour la production :

- ! Actuator pour métriques, health checks, logging:cite[9].

¥ Modularité :

- ! Choix des starters (Web, Data JPA, Security) sans conflits:cite[7].

## Principales Caractéristiques

\*. Auto-Configuration : - Détection automatique des dépendances (ex: Hibernate " config DataSource):cite[5]. - Override possible via `@Configuration` personnalisé.

# Principales Caractéristiques

- \*. CLI : - Création d'applications en Groovy avec `spring run app.groovy`.
- \*. Packaging exécutable : - Fichier JAR autonome avec `java -jar`

## Comparaison Spring vs Spring Boot

Spring Traditionnel	Spring Boot
Configuration manuelle (XML/Java)	Auto-configuration via starters
Déploiement WAR sur serveur externe	Déploiement JAR sur serveur embarqué (Tomcat/Jetty)
Starters pré-définis (ex: <code>spring-boot-starter-data-jpa</code> )	Dépendances gérées manuellement

## Spring Initializr - HelloWorld en 5 étapes

1. Accéder à [start.spring.io](https://start.spring.io).
2. Sélectionner :
  - ! Project : Maven/Gradle
  - ! Dependencies : `Spring Web`

## Spring Initializr - HelloWorld en 5 étapes

1. Générer et importer dans l'IDE.
2. Ajouter la classe:

```
@RestController
public class HelloController {
    @GetMapping("/hello")
    public String hello() {
        return "Hello World!";
    }
}
```

## Spring Initializr - HelloWorld en 5 étapes

1. Exécuter avec `mvn spring-boot:run` " <http://localhost:8080/hello>

## Modules Clés de Spring Boot

Spring Boot Starters (Packages pré-définis) : - Web : `spring-boot-starter-web` (REST + Tomcat embarqué) - Data : `spring-boot-starter-data-jpa` (Hibernate + JPA) - Security : `spring-boot-starter-security` (OAuth2/JWT) - Actuator : Monitoring (health, metrics, env)

Exemple de dépendance :

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

## Structure de `application.yml`

Format YAML recommandé pour une configuration lisible et hiérarchique :

```
server:
  port: 8080 # Port d'écoute
  servlet:
    context-path: /api # Préfixe d'API

spring:
  datasource:
    url: jdbc:postgresql://localhost/mydb
    username: ${DB_USER} # Variable d'environnement
    password: ${DB_PASSWORD}

logging:
  level:
    org.springframework: WARN # Niveau de log personnalisé
```

## Structure de `application.yml`

Avantages vs `.properties` : - Hiérarchie claire par indentation - Support des structures complexes (listes, maps) - Meilleure maintenance pour les configurations multi-environnements

## Gestion des Profils

Déclaration de profils :

```
# application.yml
---
spring:
  config:
    activate:
      on-profile: "prod"

server:
  port: 80
  error:
```

```
É whitelabel:
É enabled: false # Désactive la page d'erreur par défaut
```

## Gestion des Profils

Activation des profils : 1. En ligne de commande : `java -jar app.jar --spring.profiles.active=prod,cloud` 2. Variable d'environnement : `export SPRING_PROFILES_ACTIVE=dev` 3. Dans `application.yml` :

```
spring:
profiles:
active: dev
```

## Fonctionnalités Avancées de `application.yml`

```
app:
É # Utilisation de variables d'environnement avec valeur par défaut
É endpoint: https://${API_HOST:localhost}:${API_PORT:8080}/v1
É max-retries: ${RETRIES:3}
```

Syntaxe : `${nom_variable:valeur_par_défaut}`

## Fichiers Multi-Documents

Séparez les configurations avec `---` pour gérer plusieurs profils dans un même fichier :

```
# Config commune
spring:
É application:
É name: my-app

---
# Config DEV
spring:
É config:
É activate:
É on-profile: "dev"
server:
É port: 8081

---
# Config PROD
```

```

spring:
  config:
    activate:
      on-profile: "prod"
server:
  port: 80

```

## Import de Configurations Externes

```

spring:
  config:
    import:
      - classpath: database-config.yml # Fichier dans les ressources
      - file: /etc/app/secrets.yml     # Fichier système
      - configserver:http://config-server:8888 # Spring Cloud Config

```

Priorité : Les fichiers importés écrasent les configurations existantes.

## Ordre de Priorité

1. Arguments CLI (`--server.port=9000`)
2. Variables d'environnement
3. `application-{profile}.yml`
4. `application.yml`

## Transition Cloud-Native

Définition : - Applications conçues pour les plateformes cloud (Kubernetes, AWS) - Principes : 12-factor app, scalabilité horizontale

## Transition Cloud-Native

Fonctionnalités : - Config Server : Externalisation de la configuration - Service Discovery : Eureka pour les microservices - Circuit Breaker : Hystrix pour la résilience

## Spring Cloud vs Spring Boot

Spring Boot : - Framework autonome - Configuration simplifiée - Serveur embarqué

Spring Cloud : - Extension pour le cloud - Composants : - Gateway (routage) - Config Server - Sleuth (tracing distribué)

# Évolutions Récentes (2023+)

Native Image : - Compilation GraalVM - Temps d'initialisation <100ms - Mémoire réduite

# Évolutions Récentes (2023+)

Intégrations : - Kubernetes Operators - Serverless (AWS Lambda) - Observabilité (Micrometer)

## Cas d'Usage

Migration d'application legacy : - Réduction 80% de la config XML - JAR autonome avec Tomcat embarqué - Actuator pour le monitoring

## Bonnes Pratiques

Configuration : - Profils (`application-{prod|dev}.yml`) - Secrets managés (Vault/Kubernetes)

## Bonnes Pratiques

Tests : - `@SpringBootTest` (intégration) - `@DataJpaTest` (couche DB) - Cucumber/tzatziki

## Limitations

Points d'attention : - Taille des images Docker - Courbe d'apprentissage auto-config - Compatibilité native (GraalVM)

## Conclusion

Pourquoi choisir Spring Boot ? - Standard industriel - *ecosystème* mature - Aligné cloud-native

## Conclusion

Next Steps : - Spring Boot 3.4 (Java 21+) - Intégration AI/ML