

TECH WEEK



KAIZEN

**PROGRAMMATION FONCTIONNELLE
SCALA ET SON ÉCOSYSTÈME**



LOÏC DESCOTTE

Architecte Scala/Big Data
chez



KAIZEN



THOMAS GRAMBERT

Développeur Scala/Java
chez

kelkoogroup



QUESTIONS AUX DÉVELOPPEURS

Qui a déjà eu des problèmes

- d'accès concurrents sur un état muable ?
- liés à des valeurs nulles ?
- à des levées d'exceptions imprévues ?
- de scalabilité ?

LA PROGRAMMATION FONCTIONNELLE

TECH
WEEK



KAIZEN

LA PROGRAMMATION FONCTIONNELLE

Définition pratique :

1. Programmer avec des valeurs immuables
2. Et avec des fonctions pures

Cette définition concerne la programmation fonctionnelle pure. Selon le type de code que l'on souhaite obtenir ou les langages/librairies/frameworks utilisés, ces règles seront appliquées de façon plus ou moins strictes.

SCALA : PRÉSENTATION RAPIDE



- Langage de programmation créé en 2003 par Martin Odersky à l'EPFL
- Sortie de Scala 2.0 en 2006
- Lancement de Typesafe (Lightbend) en 2011
- Fonctionne sur la JVM et dans le navigateur
- Intersection de la programmation fonctionnelle et de la programmation orientée objet

PROGRAMMER AVEC DES VALEURS IMMUABLES

val : valeur non réassignable

```
scala> val l = List("a", "b")
```

```
l: List[String] = List(a, b)
```

```
scala> l :+ "c"
```

```
res0: List[String] = List(a, b, c)
```

```
scala> l
```

```
res1: List[String] = List(a, b)
```

```
scala> l = l :+ "c"
```

```
<console>:8: error: reassignment to val
```

```
l = l :+ "c"
```


PROGRAMMER AVEC DES VALEURS IMMUABLES

Style impératif (Java) :

```
List <String> origin = Arrays.asList ( "john" , "jerry" , "bob" ) ;  
List <String> result = new ArrayList <String> ( ) ;  
for (String name : origin ) {  
    if ( name.startsWith ( "j" ) ) {  
        result.add ( name ) ;  
    }  
}
```

Style fonctionnel (Scala) :

```
List("john", "jerry", "bob").filter(_ startsWith "j")
```

PROGRAMMER AVEC DES VALEURS IMMUABLES

Style impératif (Java) :

```
int count = 0 ;  
for ( int i = 0 ; i < 10 ; i ++ ) {  
    count = count + i ;  
}
```

Style fonctionnel (Scala) :

```
Range(0,10).reduce((count, elem) => count + elem)
```

Une nouvelle version du compteur est renvoyée à chaque itération.

FONCTIONS PURES

Les fonctions pures sont :

1. Totales : elles renvoient toujours un résultat
2. Idempotentes : elles renvoient toujours le même résultat pour les mêmes paramètres d'entrée
3. Sans effet de bord : elles ne modifient rien qui leur est extérieur

FONCTIONS PURES

Avantages des fonctions pures

- Plus faciles à tester
- Plus faciles à maintenir
- Plus prédictibles
- Plus faciles à faire scaler : programmation parallèle
programmation distribuée

CONSÉQUENCES SUR LA MANIÈRE DE CODER

Quelques règles à respecter

Chaque fonction a un type de retour qui doit être respecté :

- On ne lève pas d'exceptions
- On ne renvoie pas null

Les fonctions ne dépendent pas d'un état externe

Les modifications d'état sont portées par le type de retour

LA THÉORIE DES CATÉGORIES

- Un ensemble d'outils pour écrire des fonctions pures, dont les propriétés sont prouvées par les mathématiques.
- Foncteurs : structures sur lesquelles on peut appliquer une fonction de transformation
- Monades : foncteurs combinables
- ...

Un foncteur en image



```
val box = Box(2)
```

```
val newBox = box.map(x => x+3)
```

```
//newBox: Box(5)
```

EXEMPLE DE MONADE : OPTION

Le type Option permet de remplacer la référence null :

```
def findUser(id: String): Option[User] = {  
    if(...) Some(user) else None  
}
```

Option[User] peut être vide (None) ou avoir une valeur définie (Some[User]).

EXEMPLE : COMBINER DES OPTIONS

```
val street: Option[String] = for {  
  user <- findUser(userId)  
  address <- user.address  
} yield address.street
```

Le chaînage s'arrête au premier None rencontré.

Le type en sortie de la combinaison est également une Option, ayant une valeur ou non.

LES MONADES SONT UNE ABSTRACTION AU-DESSUS DES DONNÉES

- Option : gestion de l'absence de valeur
 - contient une valeur ou non
- Either : gestion des erreurs (remplace les exceptions)
 - contient une valeur ou une erreur
- Future : traitements asynchrones
 - contient une valeur, une erreur ou une attente de valeur

EXEMPLE : EITHER ET FUTURE

Gestion d'erreur :

```
def findUser(id: String): Either[NotFoundError, User] = {  
  ...  
}
```

Traitement asynchrone :

```
def findPerson(id: String) : Future[User] = {  
  ...  
}
```

En résumé

Nous avons vu comment, grâce à la programmation fonctionnelle répondre aux problématiques suivantes :

- Programmation concurrent et distribuée
- Scalabilité
- Gestion des erreurs (null, exceptions)

L'écosystème Scala

TECH
WEEK



KAIZEN

SPARK

Framework Big Data

- Calcul distribué et machine learning reposant sur Hadoop
- Jusqu'à 100x plus rapide qu'Hadoop

Exemple

```
val counts =  
textFile.flatMap(line => line.split(" "))  
  .map(word => (word, 1))  
  .reduceByKey(_ + _)
```



AKKA (LIGHTBEND)

- Framework pour écrire des applications distribuées et scalables
- Basé sur la programmation par acteurs et la programmation réactive
- Akka Streams : création, transformation et agrégation de flux typés



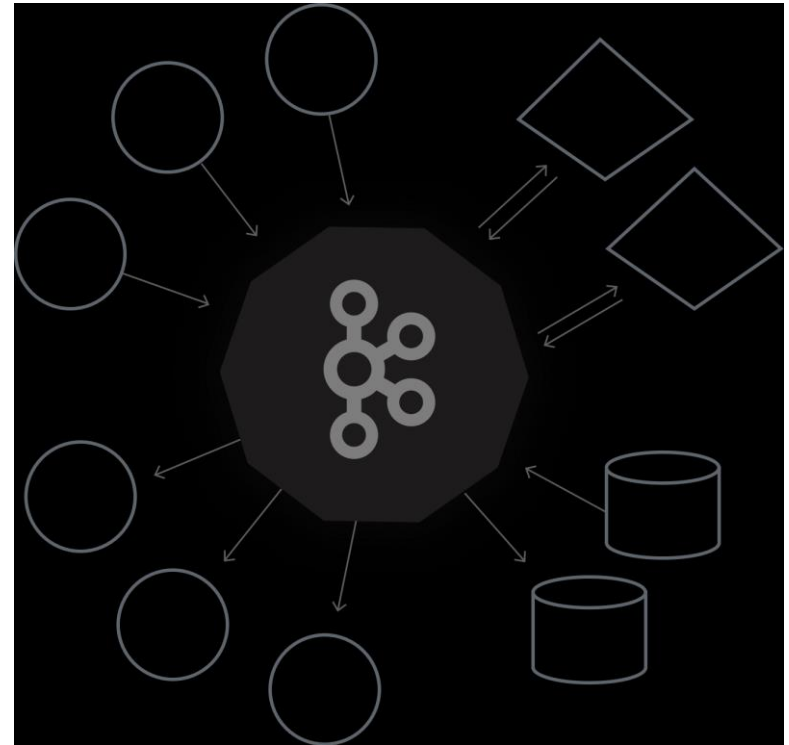
PLAY FRAMEWORK (LIGHTBEND)

- Framework Web orienté “programmation réactive”
- Architecture stateless
- Fournit une haute scalabilité

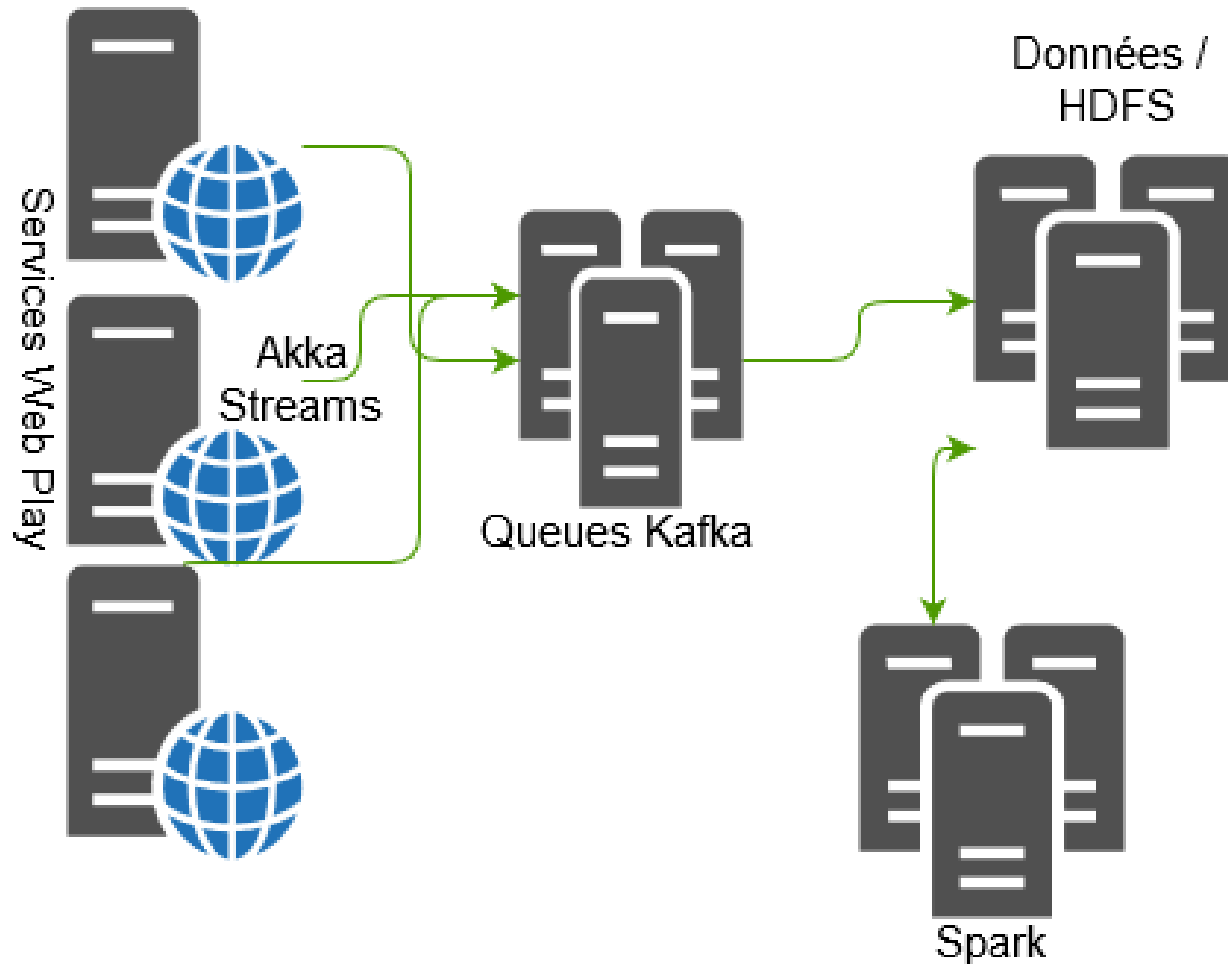


KAFKA

- Plateforme de “data streaming” distribuée
- Haute disponibilité
- En mode “publish/subscribe”



Exemple d'architecture

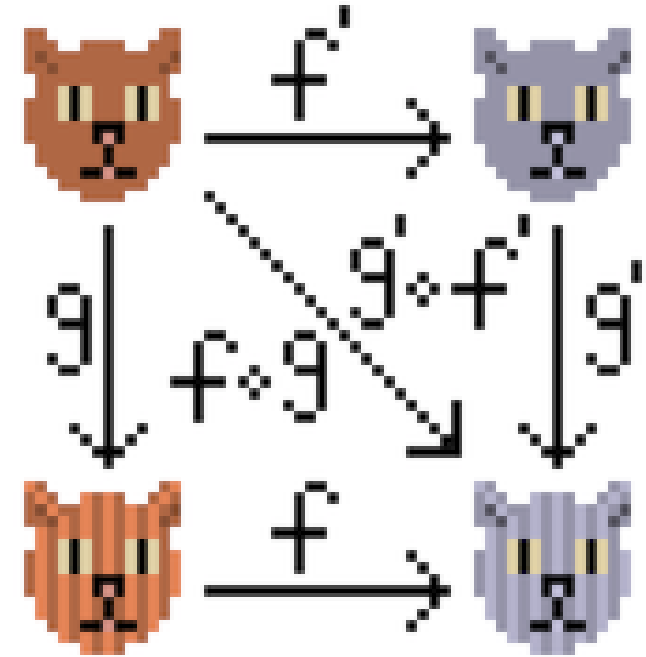


CATS

Librairie de programmation fonctionnelle complémentaire à la librairie standard Scala.

Sert de fondation à l'écosystème typelevel, un ensemble de librairies et frameworks orientés « programmation fonctionnelle typée pure » :

- shapeless (programmation générique)
- http4s (librairie http)
- doobie (accès aux bases de données)
- monix : programmation réactive



AUTRES FONCTIONNALITÉS INTÉRESSANTES DU LANGAGE

- Le pattern matching
- Les type classes (types extensibles)
- Les macros (méta-programmation)
- ...

Quelques pointeurs pour s'y mettre

- MOOC Coursera/EPFL : Functional Programming Principles in Scala
- Site web : The Neophyte's Guide to Scala
- Livre : Functional Programming, Simplified d'Alvin Alexander

MERCI DE VOTRE ATTENTION

Des questions ?

Loïc Descotte

Thomas Grambert

TECH
WEEK



KAIZEN