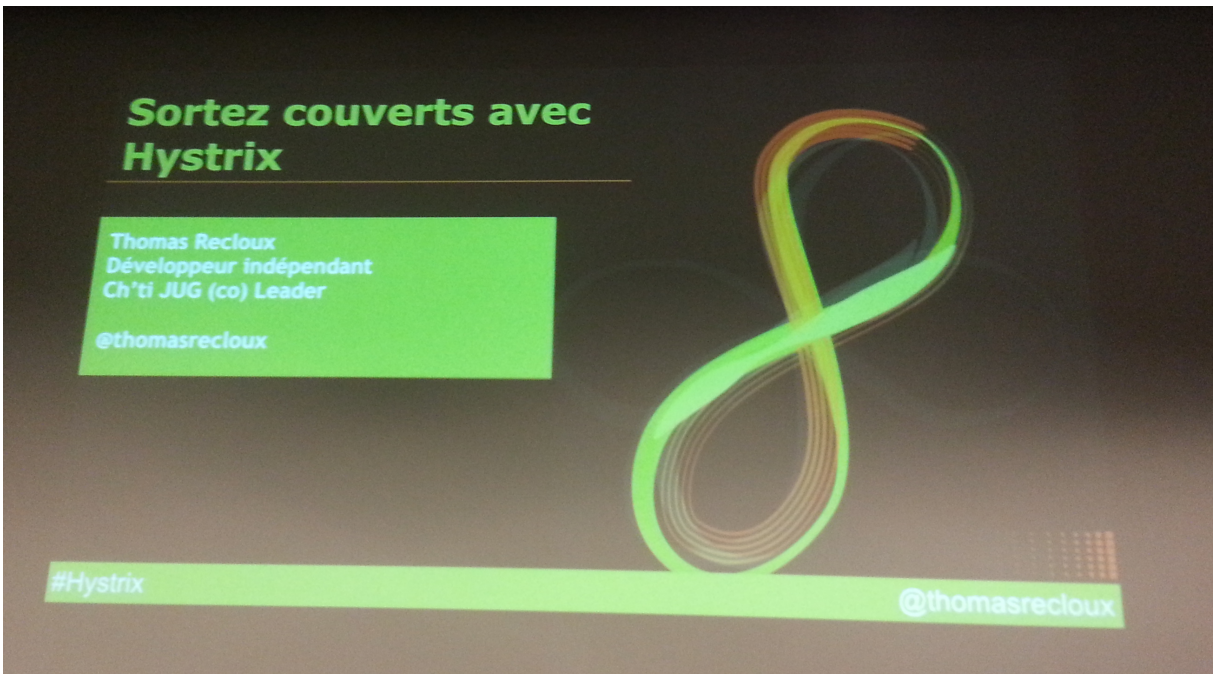
# Sortez couverts avec Hystrix

Date : 8 avril 2015
Format : Tools in action
Speakers : Thomas Recloux, développeur indépendant




Loi de Murphy : tout peut tomber en panne. C'est d'autant plus vrai dans le monde du distribué et dans le Cloud.
Référence littéraire : Release It ! de Michael T. Nygard. Analyse post-mortem de gros crash. Exemples en Java.

Patterns de stabilité :

1. **Use timeouts** : à chaque utilisation d'une ressource accédée sur le réseau, utiliser un timeout
2. **Bulkheads (partitions)** : essayer d'isoler les pannes sur un composant. S'applique à plusieurs niveaux : JVM, data center …
3. **Circuit Breaker** : lorsqu'une dépendance a pris feu, on l'isole
4. **Steady state**
5. **Fail Fast** : rien n'est pire qu'une non réponse. Retourner le plus vite une erreur
6. **Handshaking** : réaliser des check sur l'état de vie du système
7. **Test Harness** : tests des cas limites …
8. **Decoupling Middleware** : limiter les dépendances synchrones. Mieux vaut appeler un système en asynchrone.

Scénario nominal :

- Application de e-commerce qui appelle plusieurs services tiers : Paypal, CB, Twitter, service logistique
- Tout à coup, le service logistique part en timeout au bout de 60 secondes.

- Conséquences : le pool de threads « bloque » sur le service logistique. Ce dernier fait tomber tout le site (ex : service de normalisation d'adresse)

**Netflix**

- 50 000 000 de clients
- 35% de la bande passante aux US en période de pointe
- Beaucoup de MicroServices. Utilise AWS et Cassandra

**Hystrix**

- Rend une application tolérant aux pannes
- Bulkheads : les appels distants sont exécutés dans un pool de thread. **Un pool par dépendance**. Permet de limiter le nombre d'appel concurrent à ce service.
- Graceful degradation : en cas d'erreur, trouver une méthode alternative
- Use Timeouts : **timeout par défaut de 1 seconde** pour tout l'appel (pas de différenciation entre timeout de connexion ou timemout de lecture)
- Circuit Breaker : après plusieurs appels en échec, le circuit est coupé. Conditions de déclenchement : 50% des appels sur une fenêtre de 10s en erreur + 20 appels mini.
- **En interne, Hystrix utilise RxJava.**

Les valeurs par défaut ont été définies suite à l'expérience de Netflix.

## Démo avec JUnit

Dans Hystrix, l'objet encapsulant un appel distant s'appelle une commande : classe **HystrixCommand** paramétré par son type de retour. Implémentation nécessaire d'une méthode *run().* Une commande appartient à un groupe. Un thread pool par groupe.
La méthode *queue()* renvoie une Future : permet un appel en asynchrone.
Possibilité de renvoyer un Observable (programmation reactive)

Utilisation de Uninterruptibles.sleepUninterruptibly de Guava pour simuler un temps de réponse de 2 secondes et un TimeOutException levé par Hystrix.

Mise en place d'un fallback en cas de timeout : override de la méthode getFallback()

## Démo avec 2 applications Spring Boot

Charge du serveur avec Gatling
Simulation d'une panne du backend qui met 60 secondes à répondre

Module Hystrix de Spring Cloud : l'annotation @HystrixCommand permet de spécifier le nom de la méthode fallback.

## Démo du Hystrix Dashboard

Intégration dans Spring Boot avec 2 annotations Spring Boot
Monitor Hystrix