Du monolithe aux microservices chez leboncoin

Speakers : Eric Lefevre-Ardant (Leboncoin)
Format : Conférence
Date : 18 avril 2019

Lors de sa création en 2006, Leboncoin appartenait au groupe norvégien Schibsted et avait un site en Suède qui marchait très bien.
Il démarre en France avec un moteur d'orchestration écrit en C, une base PostrgreSQL et un frontend PHP (génération des pages côté serveur avec code métier dans le PHP). Tout est fait maison : même jusqu'au langage de templating HTML. L'infra est composée de distribution CentOS.

En 2016, certains services sont écrits en PHP et Python (et plus en C). La base de données est découpée.

En 2019, la stack technique repose sur C, PHP, GO, PostreSQL, Elasticsearch et Kafka. Les services sont déployés dans des pods Kubernetes après avoir été packagé dans des images Docker. Les logs sont gérés avec Datalog. L'API Gateway est assurée par Kong même s'ils s'en détachent un peu (il fait la rupture de protocole HTTPS vers HTTP).

Quelques chiffres :

- 125 microservices
- 1,5 millions de LOC dans le monolithe ramené à 1 million
- 1 millions de LOC en GO.

Pourquoi ce changement d'architecture ?

Le code de la plateforme avait été développée au milieu des années 90. Le code en C devenait in-maintenable. Il y'a une machine à états finis. Les nouveaux développeurs ne se sentent pas propriétaire du code produit. Les mises en production sont laborieuses.
Aujourd'hui, certaines microservices sont redéployés plusieurs fois par semaine.
Les microservices ont permis d'améliorer les performances.
Les développeurs front ont souhaité passer sur Mobile first et React. Ils aimeraient de belles API REST.

En 2015, il y'a un nouveau service de modération automatique d'annonces. Tous les acteurs du groupe souhaitent utiliser ce service. Extraire des composants C est laborieuse.

Fin 2016, après quelques expérimentations heureuses en Python, GO et PHP, est lancé le chantier de refonte de l'architecture en microservices.

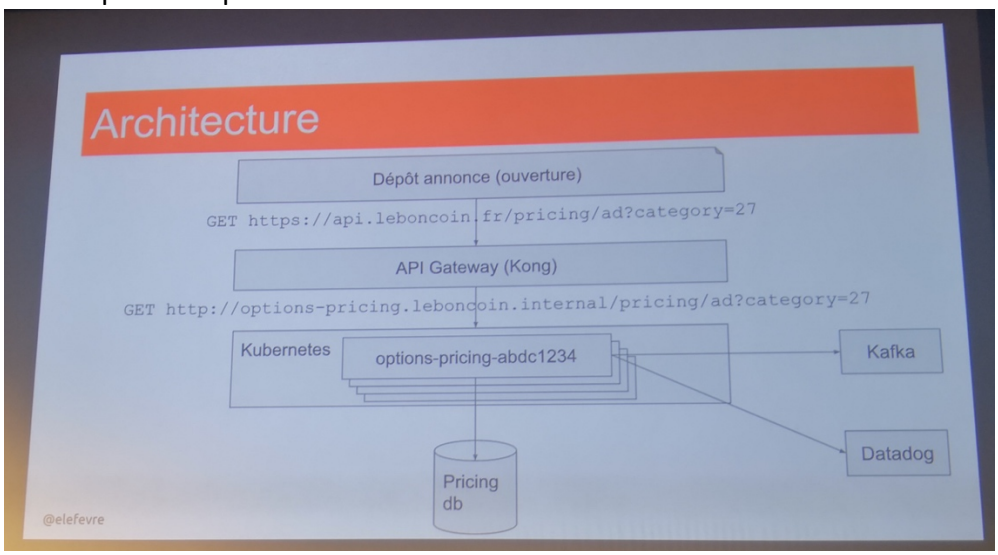
En 2017, réorganisation en Feature Teams venant des managers et développeurs.
L'équipe backend en 2016 : 15 à 18 personnes.
Avec les microservices, il est beaucoup plus facile d'assigner des responsabilités.
En 2019, on a 2 à 3 développeurs back-end par équipe.

Il y'a également eu des gains au niveau recrutement. La plupart des développeurs C venaient du monde de l'embarqué. Le passage en GO séduit davantage de développeurs. Et pour les développeurs C existants, l'apprentissage du GO se fait naturellement. Cela a également permis d'embaucher des développeurs un peu plus âgés.

A son arrivée, Eric avait été missionné pour déployer le TDD. En C, c'est compliqué. Au final, il se limite à coacher les dévs de sa Feature Team.
Les revues de code sont facilitées car on trouve plus facilement un pair même si lorsqu'il est en vacances, on doit parfois les squizzer.

Architecture

Cas simple du dépôt d'annonces



Bilan

Après 2 ans passés sur la refonte de l'architecture, il reste beaucoup de code back à migrer en microservices.
2 fois dans l'année, ils ont des séances de cleaning week (pour supprimer du code).
Aujourd'hui, le code en C n'est plus modifié. Les 25 développeurs codent en GO.
Au final, ils sont satisfaits de leur choix. Le passage en microservices demande à se faire accompagner : bons SRE, automatisation des déploiements.

Questions

Taille d'un microservice ?
Pour Eric, un microservice par route et par verbe HTTP. Le POST sera codé dans un autre microservice que le GET. D'autres collègues pensent qu'un seul microservice doit servir le GET et le POST. Pas de consensus.
Redécouper un microservice car il devient un peu gros reste laborieux.

Echéc dans les dépendances ?
Que faire lorsqu'une adhérence n'est pas disponible ? Ne pas démarrer le microservice ? Nécessité de connaître l'ordre des services. Pour Eric, le démarrage devrait échouer lorsque l'infra (base de données, kafka) n'est pas disponible. Dans la plupart des cas, ce sont les fichiers de configuration qui sont mauvais.

Compatibilité avec le legacy ?
Base de code datant de 1995. Connaissances perdues.

Divergences ?
Il n'y a plus d'architecte backend (il est parti). Depuis, tous les dévs se croient architecte. Difficulté d'avoir un consensus, même sur la documentation. Le code est de plus en plus hétérogène.

Communication ?
« Une base de données par microservice ! Alimentation par évènements », Eric en 2017
Cela n'a pas tenu. L'utilisation de Message Broker (Kafka) est utile à la Data mais est hyper couteux. Les copies locales sont aussi couteuses. A ne pas généraliser.
Au final, ils se retrouvent à faire plein d'appels directs à la base (ex : avoir le détail d'un utilisateur). Cela rend le code plus fragile.

Usage de librairies
Il en reste une : la lib de recherche est utilisée à plusieurs endroits. L'équipe responsable de la recherche a développé une lib masquant les appels Elastic.