Cloudifie ton monolithe !
Les microservices pour une architecture comme les « grands du web »

Speakers : François Berthaut (TalanLabs), Cyril Deverson (Softeam), Christophe Thepaut (BNP)
Format : Conférence
Date : 6 avril 2017
Slides : https://speakerdeck.com/fanfansama/cloudifie-ton-monolithe

Les speakers vont nous expliquer comment ils sont partis d'un monolithe développé il y'a 10 ans et qu'ils comptent lancer dans le Cloud. Thématique de la conférence sur les jeux vidéo. Le chemin est difficile. Comme dans les jeux vidéo, il y'a des monstres.

1er monstre : la complexité

Plusieurs niveaux de complexité :

1. Application très spécifique pour certains clients (ex : extraction de compte vers des formats spécifiques aux clients)
2. Application 24/7 utilisée dans le monde entier
3. Cycle de développement en cascade (4 cycles de release en //)
4. L'application n'est pas bien testée : très peu de TU, tests end-to-end OK, pas mal de bugs en prod

2ième Monstre : l'obsolescence

- Obsolescence : Java 1.4, XDoclet, Spring, Hibernate, Websphere, Oracle, IBM AIX
- Compliqué de passer d'une version à une autre à cause des versions en //
- Application RIA dont le front est développé en Flex

3ième Monstre : L'ère de l'industrie

Nécessité de reprendre le contrôle de l'application.
1ière étape : industrialiser l'application avec une Software Factory (Maven, GitLab, Jenkins, Jora, Nexus) permettant de reconstruire le Monolite manière répétable

4ième monstre : comment migrer : la genèse de nos microservices

- En 2012 : utilisation de l'approche « Business Component Factory ». Introduction de code dupliqué. Le Monolithe a grossit.
  En 2013 : un jar exécutable en mode message (un simple Business Component pour notifier le status des ordres de paiement). Il s'agissait de leur premier microservice.

5ième monstre : peur du changement

La bonne manière d'écrire un composant (aka. Un Microservice) :

- Responsabilité la plus simple possible
- Il sera petit (quelques méga)
- Déployé en dehors du serveur d'application WAS. Ne plus être lié à l'infrastructure.
- Processus autonome. Testable
- Un microservice aura : son propre code, sa propre base de données, sa propre implémentation, son propre cycle de livraison
- Communique avec un protocole lâches (Pub / Sub)
- Scalable : ne pas utiliser le mot local.

Maintenant que les règles sont posées, il faut les appliquer.

1ière étape : implémenter 2 nouvelles fonctionnalités en dehors du monolithe (permet d'acquérir de l'expérience). Ils sont testés Spring Boot et Drop Wizard. Les développeurs ont retrouvé le plaisir de coder. Inconvénient : 3 applications différentes. Mécontentement des Ops.

Pour ne pas tomber dans le chaos technique, une 2nde étape d'industrialisation a été faite. Spring Boot a été retenu. Ils ont créé un archetype Maven à partir de la coquille / châssis sur Spring Boot. Ce châssis est devenu auto-exécutable, auto-installable et auto-configurable dans l'environnement cible. Solution maison d'une infrastructure as code. Contentement des Ops.

Comment faire fonctionner tous ensemble les microservices ?
Utilisation de l'annuaire Eureka avec Spring Cloud et la suite Netflix.
Les microservices s'enregistre auprès d'Eureka.
Utilisation de ZUUL pour la découverte des services, le circuit breaker et le load-balancing.

Prêt pour migrer le Monolithe

Approche utilisée pour lever l'ambiguïté et mettre à plat les blocs fonctionnels : Domain Driven Design. Définition des frontières des microservices. On sait désormais quoi migrer.

2nde étape : expérimentation en faux mircoservices. On sort une feature. Création d'une vue afin d'avoir un schéma dédié.

3ième étape : sortir les tables de la bases.

Ensuite, il a fallu éplucher le monolithe jusqu'à sortir toutes les features.

6ième monstre : silos organisationnels

Construction d'une feature team composée de business-analyste, de dévs, d'Ops et d'architecte. Pour s'affranchir de l'infra existante et sa paperasserie, ils sont partis sur du Cloud en approchant une autre équipe d'Ops ayant mis en place du Cloud privé. Mise à disposition de Docker et Ranger.

Comment cloudifier son code ? « Packaging as code »
Configuration maven pour construire un JAR auto-exécutable et une image Docker à l'aide d'un simpe mvn clean install.
Un mvn deploy va déposer l'image dans un Nexus 3.
Pour déployer l'application, déclaration d'un docker-compose.yml => déploiement sur 1 seule machine.
Rancher va mettre à disposition un rancher-compose afin de pouvoir déployer les services sur N serveurs du Cloud Privé.

Comment faire fonctionner son API Gateway dans Docker ? Voir la vidéo de Bootiful microservices de Josh Long.

7ième Monstre : cohérence des donnés

Comment faire de la recherche multi-critères sur des domaines / micro-services distincts ? Faut-il un agrégateur de réponses ?
Utilisation du Command Query Responsability Segregation (CQRS) et de l'Event Sourcing. Chaque événement d'écriture est publié dans le bus d'évènements Kafka qui sert d'Event Store. Un micro service d'historique d'ordre souscrit au bus d'événement et les stocke dans un cache (Elasticsearch).

Les conseils en bref

Eviter d'attendre un crash de prod pour démarrer la Transformation
Utiliser les couts due à l'obsolescence comme levier pour initier le changement
Commencer par la reproductibilité par le CI si vous voulez créer le besoin de livrer par incrément
N'ayez pas peur de vous tromper jusqu'à ce que vous trouviez votre Style d'architecture et retrouviez le plaisir de coder
Utilisez vos réussites pour définir vos règles d'architecture
Standardisez vos microservices pour en faciliter l'adoption (pas de polyglotes trop prématuré)
Utilisez un Discovery Server (API Gateway) pour simplifier les déploiements et augmenter la résilience