Les problèmes que l'on rencontre en microservice : configuration, authentification et autres joyeusetés

Speaker : Quentin Adam (Clever Cloud)
Format : Conférence
Date : 7 avril 2017
Slides : https://fr.slideshare.net/quentinadam/problems-youll-face-in-the-microservice-word-configuration-authentication-devoxx-france-2017

Le commencement de Clever Cloud

Travail : exécuter du code sur plusieurs serveurs. Il y'a 6 ans le termes de microservice n'existait pas.
Leitmotiv : « Remote Code Execution as a Service »
Attention : code des autres pouvant être malicieux (attaque de Clever Cloud, d'autres clients ou externes).
Il fallait absolument sécuriser leur service. Pour se faire, il faut se dire qu'une brique peut potentiellement agresser les autres.
La contenerisation doit être fiable.

Mutli stacks : JEE, Rust, GO, rubby …

L'un des avantages des microservices est de pouvoir scaler par service.

Lorsque 2 micro-services discutent ensemble (server -> server), on ne peut pas considérer que le réseau soit fiable. Chaque échange doit être authentifié, chiffré, audité et loggé
Grosse mise en garde de Quentin.
Au démarrage de NPM, la base de données de NPM était ouverte en lecture/écriture depuis 8 mois. Pas de logs. Très dangereux.

Le routage est un problème : comment faire aller une communication du MS A au MS B.
Lorsque c'est hard-codé, il faut implémenter une fonction de retry. Quentin privilégie l'utilisation d'une Message Box.

Besoin de dupliquer les messages afin de pouvoir intégrer facilement de nouvelles briques.
A partir du moment où on peut dupliquer les messages, on peut les insérer en base de données pour faire de l'analytique et de l'audit.

Quentin n'est pas convaincu de mettre du HTTP pour tout.
Utilisation d'un Message brokers : RabbitMQ pour les petits besoin, Kafka (la star du moment, plus compliqué), Redis ? (mais pas d'acquittement), 0MQ ? (plus compliqué car il gère moins de chose). Pour agréger l'ensemble des messages, utiliser des bases de données évènementiels tels que Warp10.

Quelle est la bonne taille d'un microservice ? Quelle frontière doit-on avoir ?
Ne pas trop découper : attention au massive RPC car on passe par le réseau. Or, le réseau est fragile, lent (vraiment lent).

Un micro-service n'est pas un modèle de workload distribution. Faire du Akka ou du Erlang ETP. Le microservice ne distribue pas du compute mais de la logique. Il ne doit pas passer son temps à faire des appels aux autres.
Est-ce que plusieurs microservices partagent la même lib ?
Chaque microservice dispose-t-il de son propre datastore ?

Configuration

Configuration = ce qui permet de parler à l'extérieur
La configuration ne doit pas être dans le code
Solutions dont Quentin n'est pas fan : Zk, etcd, consul => deviennent des SPOF
La plupart des logiciels ne savent pas changer dynamiquement leur configuration. Exemple d'Hibernate ou des reverse-proxy. Idée simple d'immutable infrastructure : pas de changement de conf à chaud.
Solution agnostique et simple : utilisation des variables d'environnement. L'application redémarrera si sa configuration doit changer. Chez Clever, ils ont développé un Service Dependencies.

Distribution d'authentification

Comment sait-on quel est l'utilisateur et si il a le droit d'y accéder ?
Solutions existantes :

- Shared data repository (par exemple avec Redis) : très mauvaise idée
- Proxy : reverse-proxy qui authentifie l'utilisateur puis réinjecte l'utilisateur dans les headers de la requête. Pas super : le développeur doit avoir le reverse-proxy sur son poste + code métier dans le reverse-proxy + SPOF + dégradation des perfs du proxy
- Central API call to authenticate request : Chaque service authentifie la requête en attaquant une API d'Authentification. L'API Authentification reste un SPOF. Assez lent. Difficulté à mocker. C'est ce qui est majoritairement utilisé chez Clever. Nécessité d'outils comme JWT ou Macaroons.
- JWT (JSON Web Token) : les services ont justes à vérifier si le token reçu a été signé par le serveur d'authentification.
- Macaroons (vient de chez Google) : type de token. Partage d'une clé entre 2 serveurs. Plus besoin de service d'authentification. Dans un Macaroon, on peut ajouter des droits. A partir d'un Macaroon, on peut en créer un autre avec autant de droits ou moins puis le donner à quelqu'un d'autre.

Faire attention à la maintenance d'un microservice (et de ses dépendances).
Etre capable de jeter le code du microservice. Les microservices peuvent être interchangés. Le code a besoin d'être jeté. Ne pas accumuler de la dette de code.

Faire des microservices nécessite d'automatiser l'orchestration des déploiements.

No dogma, full developer hapiness oriented architecture
L'architecture doit être drivée par la developper hapiness => développeurs efficaces