The Bootiful Microservice

Speaker : Josh Long (Pivotal)
Format : Conférence
Date : 22 avril 2016
Présentation : http://github.com/joshlong/bootiful-microservices

Josh est auteur du livre Cloud Native Java et Java Champion.
Il a écrit 5 livres et plein de bugs ☺
Il travaille chez Pivotal et contribue à Redis, Spring.IO, Cloud Foundry, Tomcat, RabbitMQ

Dans ce monde, il faut aller vite si on ne veut pas se faire disrupter. Ils ont tué la compétition. Uber, Air bnb, Tesla, Booking, Netflix …
Le développeur doit se concentrer sur une petite partie du travail => micro service.
A notre niveau, Josh conseille d'utiliser les micro-services pour de développer rapidement et non pas pour absorber la charge de Google. 500 lignes d'instructions sur une page wiki ne permettent pas d'aller vite !

Spring Boot et Spring Cloud permettent d'aller vite. Josh nous le prouve au cours de cette conférence en enchaînant le développement de me micro-services en live coding.

Plateforme start.spring.io :
Permet de créer rapidement une application de réservation avec les technos : web, H2, JPA, Actuator, Config Client, Eureka, Stream Rabbit, Rest Repositories.
Il y'a très longtemps, on devait utiliser le packaging WAR. En 2016, on choisit le JAR.
« make jar not war »
Télécharge un zip et l'importe dans IntelliJ.

Classe Reservation : une entité JPA.
Interface Spring Data JPA : ReservationRepository annotée avec @RepositoryRestResource
@Component DummyCLR qui implémente CommmandLineRunner (joke : déclarer les attributs de dépendances final pour ne pas tuer un TU). Dans la méthode run, on alimente la base de données en mémoire.
Les données sont déjà disponibles en REST et HATEOS.

Génération depuis start.spring.io d'un config-service.

Permet d'avoir un serveur qui gère la config. Les micro services vont pouvoir se connecter à ce service de configuration à la place de récupérer les données dans application.properties. Dans l'application de réservation, créer un fichier bootstrap.properties contenant les paramères de connexion au config service.

Annotation @ResfreshScope ?
Changement en live d'un message dans le config service
Appel du refresh par un POST sur l'appli de réservation.

Permet de s'affranchir du DNS. La conf est en cache et peut être rafraichie.

Création d'un **eureka-service** pour mettre en relation les services avec leurs clients.

Création d'un client de réservation. Ce client peut être localisé partout : Heroku, Tesla, Réfrigérateur …
Utilisation du reverse proxy : @EnableZuulProxy
Sur le client, on utilise @LoadBalanced RestTemplate en ne mettant pas le DNS mais le nom du service : http://reservation-service/reservations. Le Load balancing est assuré.
Pour être résistant aux pannes : @EnableCircuitBreaker (Netflix Hystrix). Permet de créer une méthode fallback si le service de réservation n'est pas là.
Création d'une file RabbitMQ pour créer une nouvelle réservation
@EnableBinding(Source.class)
Côté microservice, on utilise le @EnableBinding(Sink.class) (sink pour input) et un @MessageEndpoint avec un @SerciceActivator

Création d'un Zikpin-service pour suivre les messages dans une architecture distribuée et asynchrone (est-ce que Dynatrace le fait ?).

Spring Cloud est développé par Twiter, Netflix, Pivotal.