
Ecole National d'Ingénieur de Sousse

---

# Projet module de réseau d'entreprise
# BUD BOOT

---

*Auteur :*
Oussama Faleh

*Encadrant :*
Mr. Anis Ben Arbia

4 juillet 2020

# Table des matières

# Chapitre 1

# Introduction

Le génie logiciel, l'ingénierie logicielle ou l'ingénierie du logiciel (en anglais : software engineering) est une science de génie industriel qui étudie les méthodes de travail et les bonnes pratiques des ingénieurs qui développent des logiciels. Le génie logiciel s'intéresse en particulier aux procédures systématiques qui permettent d'arriver à ce que des logiciels de grande taille correspondent aux attentes du client, soient fiables, aient un coût d'entretien réduit et de bonnes performances tout en respectant les délais et les coûts de construction . Pour cela le role d'un bon ingénieur pour moi est "d'optimiser le cout de développement en tenant compte de la qualité de produit " donc ,vu la difficulter de developpement des projet en utilisant l'archtecture microservices, j'est decidé de créer un socle de développement qui sert à simplifier le travail des développeurs informatiques (les codeurs si vous préférez), en leur offrant une architecture "prête à l'emploi" et qui leur permette de ne pas repartir de zéro à chaque nouveau projet .

# Chapitre 2

# presentation du Projet

Bud Boot est un fichier jar qui sert à auto-générer un code entièrement personnalisable pour la création , la gestion et le deploiment des sites Web extensibles basés sur des microservices encastrer dans les contenaires. avec plusieurs outils communs réutilisables comme "authentification" ,"mailing , "CI/CD" ,"adminstration dashboard" , "auto-generating custom microservices" .

## 2.1 les microservices ?

— Les microservices sont des fonctionnalités extrêmement petites qui sont responsables d'un domaine spécifique.

— Aucune norme industrielle n'existe pour les microservices. Contrairement aux autres premiers protocoles de service Web, les microservices adoptent une approche basée sur des principes et s'alignent sur les concepts de REST et JSON.

— L'écriture de microservices est facile, mais leur opérationnalisation complète pour la production nécessite une réflexion supplémentaire. Nous avons introduit plusieurs catégories de modèles de développement de microservices, notamment le développement de base, les modèles de routage, la résilience des clients, la sécurité, la journalisation et les modèles de génération / déploiement.

— Bien que les microservices soient indépendants du langage, nous avons introduit deux frameworks Spring qui aident de manière significative à la création de microservices : Spring Boot et Spring Cloud.

— Spring Boot est utilisé pour simplifier la construction de microservices basés sur REST / JSON. Son objectif est de vous permettre de construire rapidement des microservices avec rien de plus que quelques annotations.

— Spring Cloud est une collection de technologies open source d'entreprises telles que Netflix et HashiCorp qui ont été «enveloppées» d'annotations Spring pour simplifier considérablement l'installation et la configuration de ces services.

## 2.2 architecture du projet

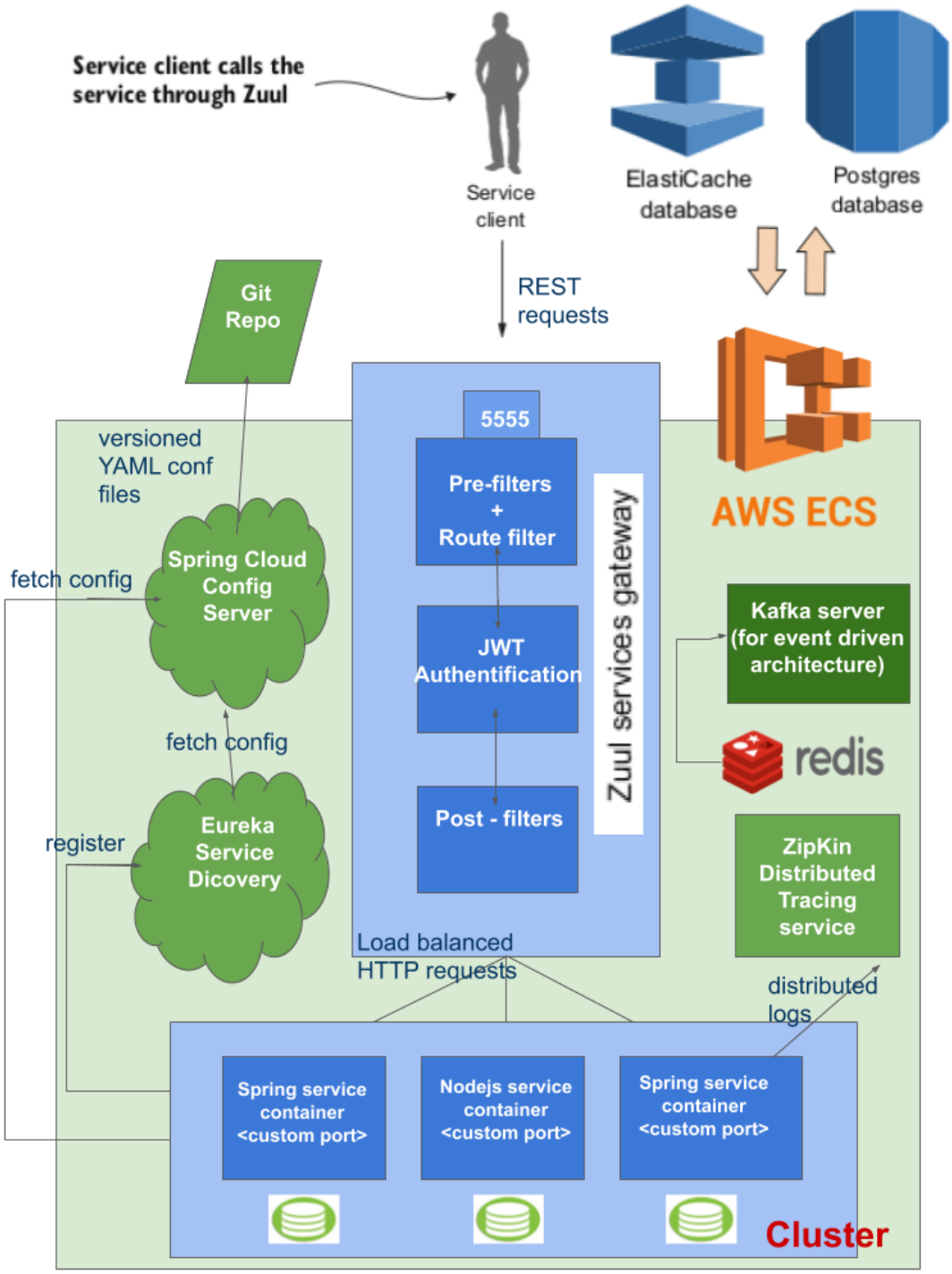


FIGURE 2.1 – Schéma descriptif

## 2.3 Pourqoui les microservices ?

La réalité, cependant, est que le développement de logiciels n'est pas un processus linéaire de définition et d'exécution, mais plutôt un processus évolutif où il faut plusieurs itérations pour communiquer avec le client, apprendre de lui et le livrer avant que l'équipe de développement ne comprenne vraiment le problème à portée de main.

Les défis liés à l'utilisation des méthodologies traditionnelles en cascade sont aggravés par le fait que la granularité des artefacts logiciels livrés dans ces projets est souvent :

— Tightly coupled - L'appel de la logique métier se produit au niveau du langage de programmation au lieu de passer par des protocoles neutres d'implémentation tels que SOAP et REST. Cela augmente considérablement les chances que même une petite modification d'un composant d'application puisse casser d'autres parties de l'application et introduire de nouveaux bogues.

— Leaky : la plupart des grandes applications logicielles gèrent différents types de données. Par exemple, une application de gestion de la relation client (CRM) peut gérer les informations sur les clients, les ventes et les produits. Dans un modèle traditionnel, ces données sont conservées dans le même modèle de données et dans le même magasin de données. Même s'il existe des limites évidentes entre les données, il est trop souvent tentant pour une équipe d'un domaine d'accéder directement aux données appartenant à une autre équipe.

— Monolithic : Étant donné que la plupart des composants d'application pour une application traditionnelle résident dans une base de code unique partagée entre plusieurs équipes, chaque fois qu'une modification du code est effectuée, l'application entière doit être recompilée, réexécutée tout au long d'un cycle de test et redéployé. Même de petits changements la base de code de l'application, qu'il s'agisse de nouvelles exigences des clients ou de corrections de bogues, devient coûteuse et prend du temps, et les modifications importantes deviennent presque impossibles à faire en temps opportun.

La réalité, cependant, est que le développement de logiciels n'est pas un processus linéaire de définition et d'exécution, mais plutôt un processus évolutif où il faut plusieurs itérations pour communiquer avec le client, apprendre de lui et le livrer avant que l'équipe de développement ne comprenne vraiment le problème à portée de main.

Une architecture basée sur des microservices adopte une approche différente pour fournir des fonctionnalités. Plus précisément, les architectures basées sur des microservices ont les caractéristiques suivantes :

— Contraint - Les microservices ont un seul ensemble de responsabilités et sont de portée étroite. Les microservices adoptent la philosophie UNIX selon laquelle une application n'est rien de plus qu'une collection de services où chaque service fait une chose et fait très bien cette chose.

— Couplage lâche : une application basée sur un microservice est un ensemble de petits services qui interagissent uniquement les uns avec les autres via une interface non spécifique à l'implémentation à l'aide d'un protocole d'appel non propriétaire (par exemple, HTTPL avec licence <null> 37 et REST). Tant que l'interface du service ne change pas, les propriétaires du microservice ont plus de liberté pour apporter des modifications au service que dans une architecture d'application traditionnelle.

— Abstracted : Les microservices possèdent entièrement leurs structures de données et leurs sources de données. Les données appartenant à un microservice ne peuvent être modifiées que par ce service. Le contrôle d'accès à la base de données contenant les données du microservice peut être verrouillé pour autoriser uniquement l'accès au service.

— Independent : chaque microservice d'une application de microservice peut être compilé et déployé indépendamment des autres services utilisés dans l'application. Cela signifie que les changements peuvent être isolés et testés beaucoup plus facilement qu'avec une application monolithique plus fortement interdépendante.

# Chapitre 3

# Contrôle de votre configuration avec Spring Cloud configuration server

La gestion de la configuration des applications peut sembler banale, mais elle est d'une importance cruciale dans un environnement cloud. Comme nous le verrons plus en détail dans les chapitres suivants, il est essentiel que vos applications et les serveurs sur lesquels elles s'exécutent soient immuables. et que l'intégralité du serveur promu n'est jamais configurée manuellement entre les environnements. Cela va à l'encontre des modèles de déploiement traditionnels où vous déployez un artefact d'application (par exemple, un fichier JAR ou WAR) avec ses fichiers de propriétés dans un environnement «fixe». Avec un modèle basé sur le cloud, les données de configuration de l'application doivent être complètement séparées de l'application, avec les besoins de données de configuration appropriés injecté au moment de l'exécution afin que les mêmes objets serveur / application soient systématiquement promus dans tous les environnements.

## 3.1 Cycle de vie de microservice

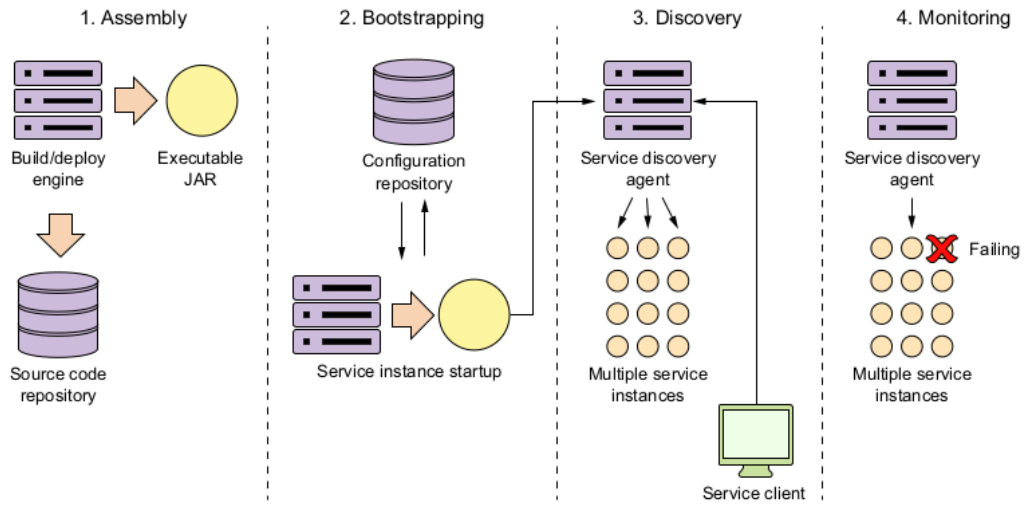


FIGURE 3.1 – Les données de configuration de l'application sont lues pendant la phase d'amorçage du service.

## 3.2 Multi-configuration centralisée

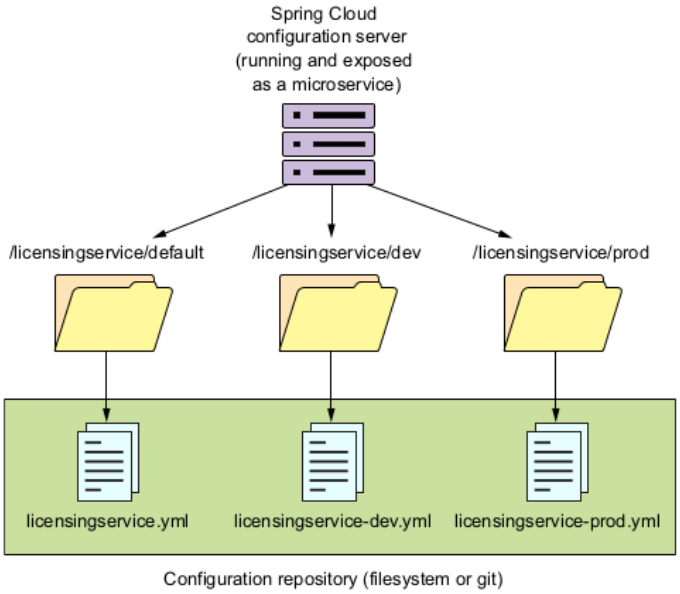


**Figure 3.3 Spring Cloud configuration exposes environment-specific properties as HTTP-based endpoints.**

FIGURE 3.2 – Configuration repository (filesystem or git)

Le serveur de configuration Spring Cloud vous permet de définir des propriétés d'application avec des valeurs spécifiques à l'environnement.

Spring utilise des profils Spring pour lancer un service afin de déterminer quelles propriétés d'environnement doivent être récupérées à partir du service Spring Cloud Config.

Le service de configuration de Spring Cloud peut utiliser un référentiel de configuration d'application basé sur fichier ou Git pour stocker les propriétés d'application.

Le service de configuration Spring Cloud vous permet de crypter des fichiers de propriétés sensibles à l'aide d'un cryptage symétrique et asymétrique.

# Chapitre 4

# service discovery

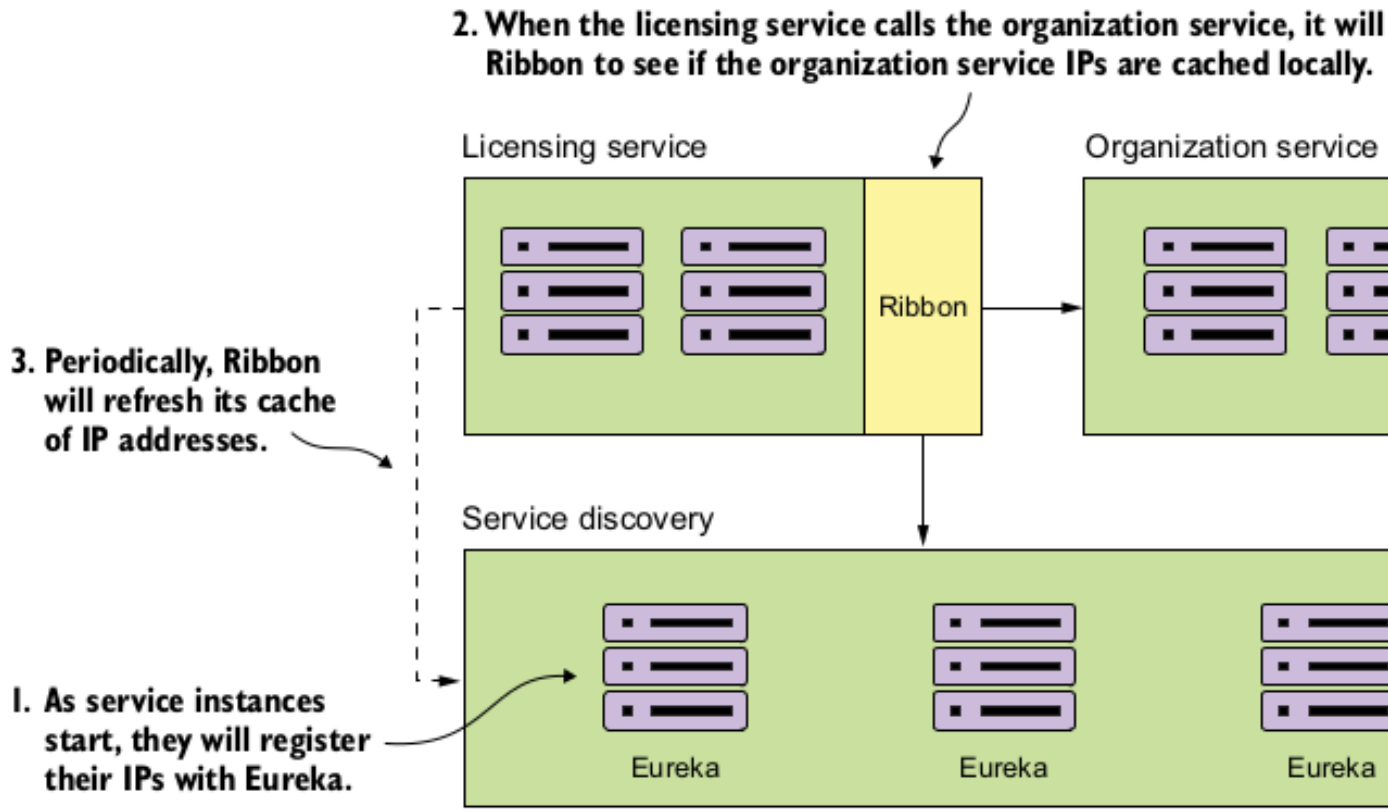


FIGURE 4.1 – service discovery [1]

Les pannes directes d'un service (par exemple, les pannes de serveur) sont faciles à détecter et à traiter.

Un seul service peu performant peut déclencher un effet en cascade d'épuisement des ressources car les threads du client appelant sont bloqués en attendant la fin d'un service.

Les trois modèles de résilience de base du client sont le modèle de disjoncteur, le modèle de secours et le modèle de cloison.

Le modèle de disjoncteur cherche à tuer les appels système lents et dégradés afin que les appels échouent rapidement et empêchent l'épuisement des ressources.

Le modèle de secours vous permet en tant que développeur de définir des chemins de code alternatifs dans le cas où un appel de service distant échoue ou le disjoncteur de l'appel échoue.

Le modèle de tête en bloc sépare les appels de ressources distants les uns des autres, isolant les appels vers un service distant dans leur propre pool de threads. Si un ensemble d'appels de service échoue, ses échecs ne doivent pas être autorisés à consommer toutes les ressources du conteneur d'application.

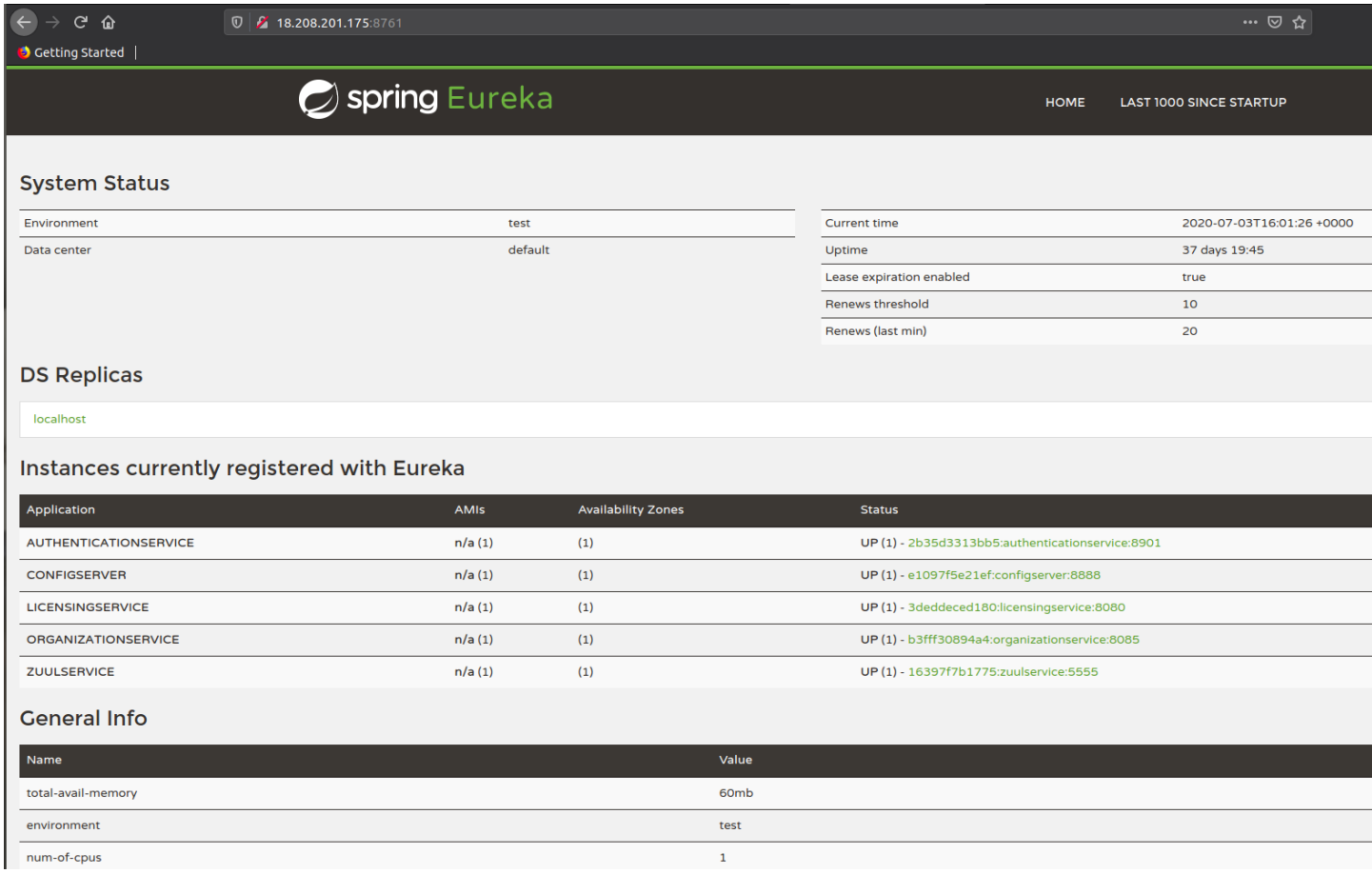


FIGURE 4.2 – service discovery

Spring Cloud et les bibliothèques Netflix Hystrix fournissent des implémentations pour les modèles de disjoncteur, de secours et de cloison.

Les bibliothèques Hystrix sont hautement configurables et peuvent être définies aux niveaux global, classe et pool de threads. Hystrix prend en charge deux modèles d'isolement : THREAD et SEMAPHORE.

Le modèle d'isolation par défaut d'Hystrix, THREAD, isole complètement un appel protégé par Hystrix, mais ne propage pas le contexte du thread parent au thread géré Hystrix.

L'autre modèle d'isolement d'Hystrix, SEMAPHORE, n'utilise pas de thread distinct pour effectuer un appel Hystrix. Bien que cela soit plus efficace, il expose également le service à un comportement imprévisible si Hystrix interrompt l'appel. Hystrix vous permet d'injecter le contexte du thread parent dans un thread géré par Hystrix via une implémentation HystrixConcurrencyStrategy personnalisée.

# Chapitre 5

# Service routing avec Spring Cloud and Zuul

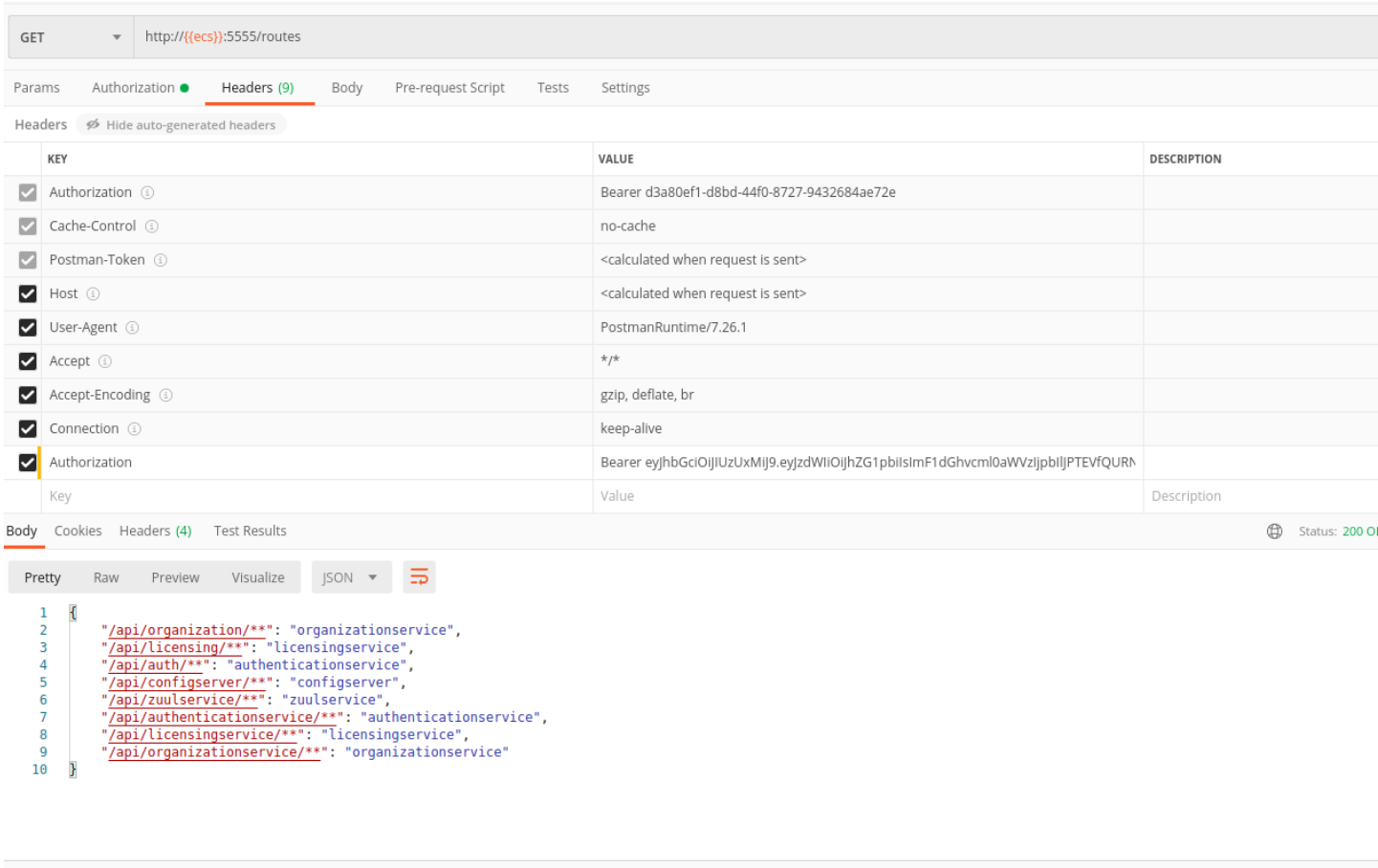


FIGURE 5.1 – route config

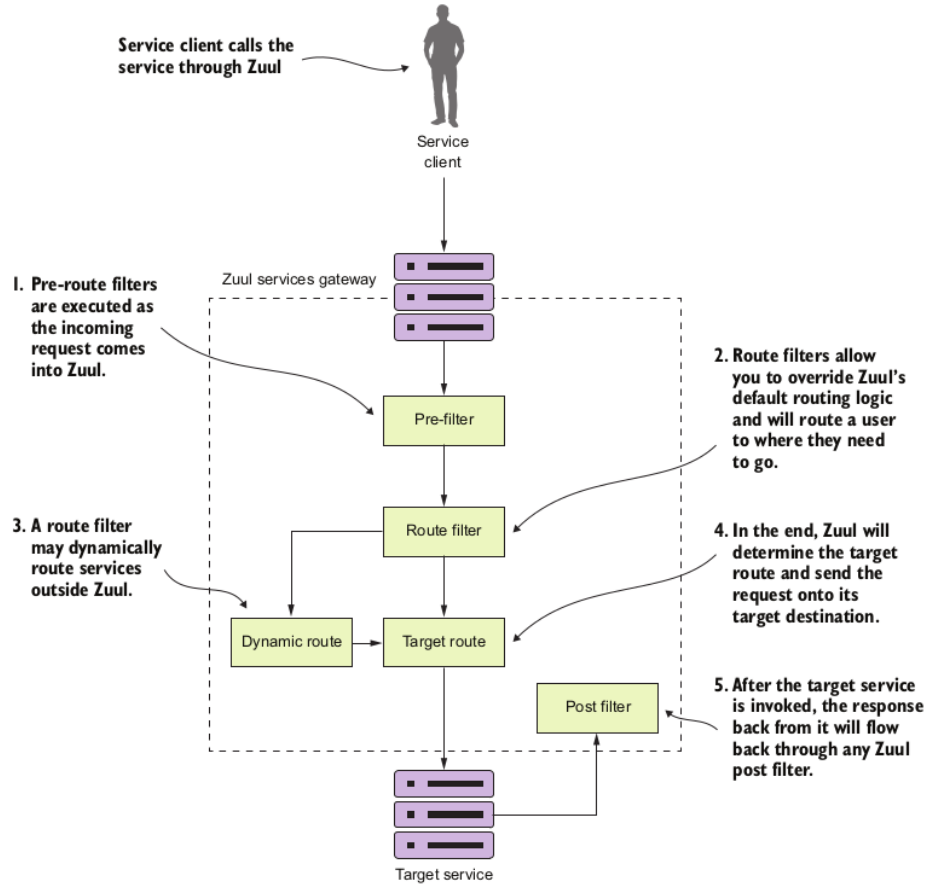


FIGURE 5.2 – architecture du gateway

Spring Cloud simplifie la création d'une passerelle de services.

La passerelle de services Zuul s'intègre au serveur Eureka de Netflix et peut automatiquement mapper de façon systématique les services enregistrés auprès d'Eureka à un itinéraire Zuul.

Zuul peut préfixer tous les itinéraires gérés, vous pouvez donc facilement préfixer vos itinéraires avec quelque chose comme / api.

À l'aide de Zuul, vous pouvez définir manuellement des mappages d'itinéraire. Ces mappages de routes sont définis manuellement dans les fichiers de configuration des applications. En utilisant le serveur Spring Cloud Config, vous pouvez recharger dynamiquement les mappages de routes sans avoir à redémarrer le serveur Zuul.

Vous pouvez personnaliser les délais d'expiration Hystrix et Ruban de Zuul aux niveaux de service global et individuel.

Zuul vous permet d'implémenter une logique métier personnalisée via des filtres Zuul. Zuul propose trois types de filtres : les filtres Zuul pré, post et routage.

Les préfiltres Zuul peuvent être utilisés pour générer un ID de corrélation qui peut être injecté dans chaque service traversant Zuul. Un post-filtre Zuul peut injecter un ID de corrélation dans chaque réponse de service HTTP à un client de service.

Un filtre d'itinéraire Zuul personnalisé peut effectuer un routage dynamique basé sur un ID de service Eureka pour effectuer des tests A / B entre différentes versions du même service.

# Chapitre 6

# sécuriser vos microservices jwt

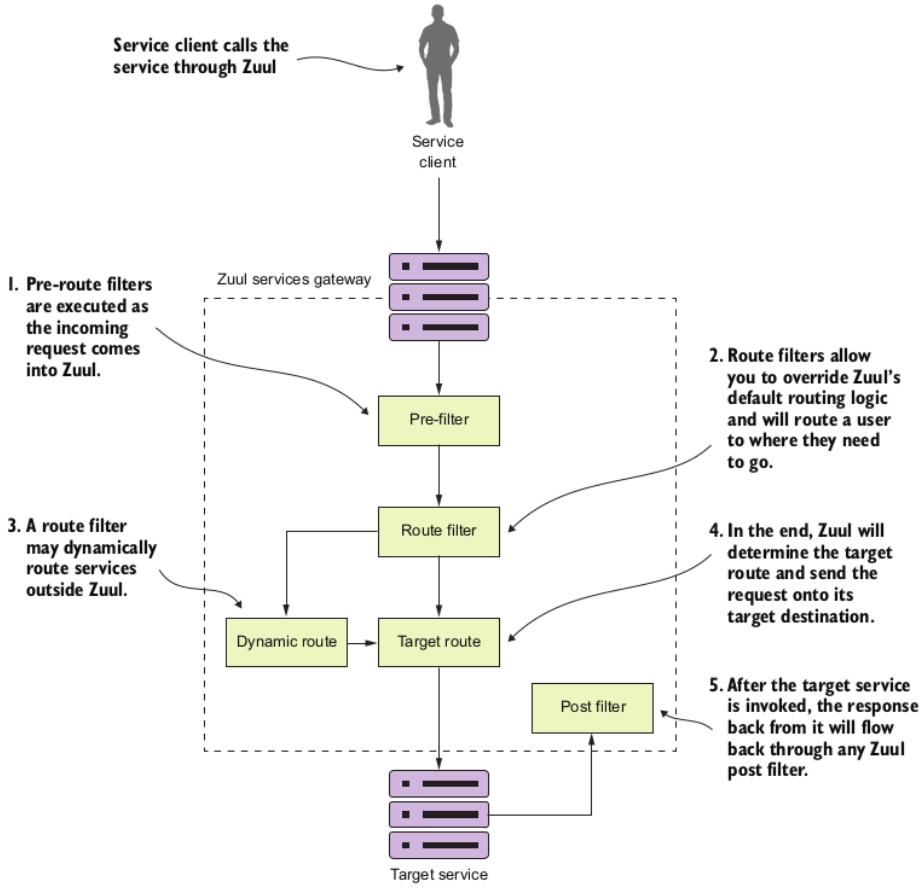


FIGURE 6.1 – architecture du gateway

OAuth2 est un cadre d'authentification basé sur des jetons pour authentifier les utilisateurs.

OAuth2 garantit que chaque microservice exécutant une demande utilisateur n'a pas besoin d'être présenté avec des informations d'identification utilisateur à chaque appel.

OAuth2 propose différents mécanismes de protection des appels de services Web. Ces mécanismes sont appelés subventions.

Pour utiliser OA uth 2 au printemps, vous devez configurer un service d'authentification basé sur OA uth 2.

Chaque application qui souhaite appeler vos services doit être enregistrée auprès de votre service d'authentification OA uth 2.

Chaque application aura son propre nom d'application et sa propre clé secrète. Les informations d'identification et les rôles des utilisateurs sont en mémoire ou dans un magasin de données et accessibles via la sécurité Spring. Chaque service doit définir les actions qu'un rôle peut entreprendre.

Spring Cloud Security prend en charge la spécification JWT (JavaScript Web Token).

JWT définit une norme JavaScript signée pour générer des jetons OA uth 2.

Avec JWT, vous pouvez injecter des champs personnalisés dans la spécification.

Sécuriser vos microservices implique plus que simplement utiliser OA uth 2. Vous devez utiliser HTTPS pour crypter tous les appels entre les services.

Utilisez une passerelle de services pour réduire le nombre de points d'accès via lesquels un service peut être atteint.

Limitez la surface d'attaque d'un service en limitant le nombre de ports entrants et sortants sur le système d'exploitation sur lequel le service s'exécute.

# Chapitre 7

# deployer vos microservices

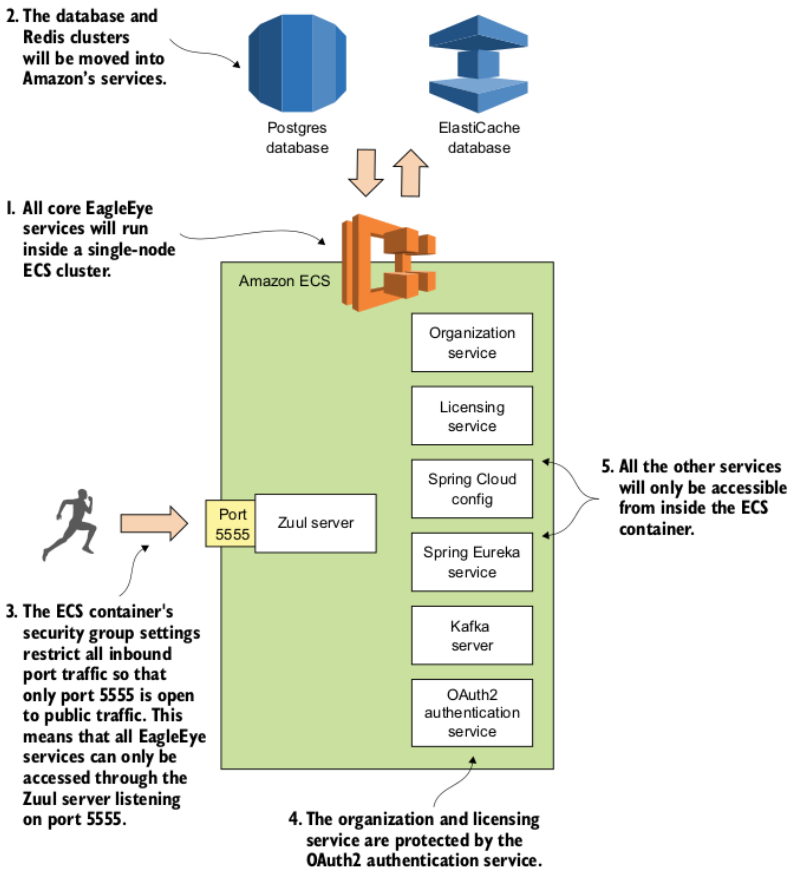


FIGURE 7.1 – architecture "elastic container service

subcaption

Le pipeline de génération et de déploiement est un élément essentiel de la fourniture de microservices.

Un pipeline de construction et de déploiement qui fonctionne bien devrait permettre de déployer de nouvelles fonctionnalités et des corrections de bogues en quelques minutes.

Le pipeline de génération et de déploiement doit être automatisé sans interaction humaine directe pour fournir un service. Toute partie manuelle du processus représente une opportunité de variabilité et d'échec.

L'automatisation du pipeline de génération et de déploiement nécessite beaucoup de scripts et de configuration pour bien fonctionner. La quantité de travail nécessaire pour le construire ne doit pas être sous-estimée.

Le pipeline de construction et de déploiement doit fournir un environnement virtuel immuable image de la machine ou du conteneur. Une fois qu'une image de serveur a été créée, elle ne doit jamais être modifiée.

La configuration du serveur spécifique à l'environnement doit être transmise en tant que paramètres au moment de la configuration du serveur.

.5

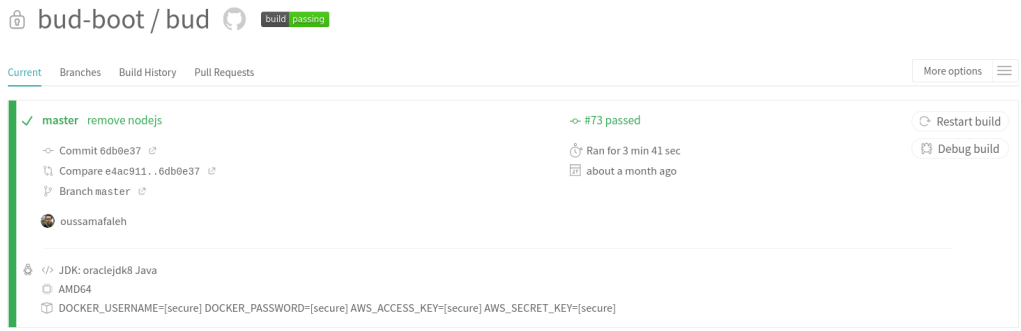


FIGURE 7.2 – travis-ci

.5


FIGURE 7.3 – docker container's tags

FIGURE 7.4 – deployment environment