Évaluation et comparaison d'un sous-ensemble d'outils de découverte de services

28 Février 2019

Contributors	**Organization**
Yuwei Wang	Télécom SudParis

Coordinators	**Organization**
Kavoos Bojnourdi	EDF Labs
Sophie Chabridon	Télécom SudParis
Denis Conan	Télécom SudParis

Document versions	
Version 1.0	28 Février, 2019

1 Introduction

La découverte de services est un composant important de la plupart des systèmes distribués et des architectures à base de microservices.

Dans un système dynamique, par exemple un système sur plate-formes Clouds, les localisations de services peuvent changer souvent en raison de la montée en charge, des nouveaux déploiements de services, ainsi que du fait que des machines échouent ou sont remplacées, etc. Dans ce cas, la découverte de services devient beaucoup plus importante afin d'éviter l'interruption du système.

Ce problème a été abordé de nombreuses manières différentes et continue à évoluer. Il existe déjà des solutions industrielles ou expérimentales en open-source. Dans ce document, nous avons choisi deux solutions populaires, Netflix Eureka[1] et HashiCorp[2] Consul, en tant que composant de la découverte de services et nous les comparons selon différents aspects.

Le document est organisé de la façon suivante. La section 2 explique la problématique. La section 3 détaille la comparaison des outils choisis. Les résultats et la conclusion sont discutés dans la section 4.

2 Problématique

Le problème concernant la localisation des services présente principalement deux aspects : l'enregistrement de service et la découverte de services [1].

- L'enregistrement de service : un processus enregistre la localisation d'un service dans un annuaire de services central. Il enregistre généralement son adresse IP et son port, et parfois les informations d'identification, de protocoles, de numéros de version et d'autres détails d'environnement.
- La découverte de services : le processus d'une application cliente interrogeant l'annuaire de services pour connaître la localisation des services.

Les solutions d'enregistrement et de découverte de services sont souvent liées avec d'autres aspects à considérer : la surveillance de l'état des services, l'équilibrage de la charge, l'intégration dans un système existant, les dépendances d'exécution, etc. En général, ils fournissent un mécanisme de coordination pour faciliter l'enregistrement et la recherche de services.

3 Anayse de deux solutions existantes et comparaison

Dans ce document, nous avons choisi deux solutions existantes pour la problématique présentée dans la section 2 : Netflix Eureka dans le groupe de Netflix Open Source Software et HashiCorp Consul.

Dans cette section, nous allons les comparer par rapport à leur architecture, leurs mécanismes de surveillance et d'équilibrage de la charge, leurs capacités d'intégration, ainsi que leur application du théorème CAP [2].

1 Netflix Eureka : https://github.com/Netflix/eureka
2 HashiCorp Consul : https://www.consul.io/

3.1 Architecture

3.1.1 Eureka

Eureka est conçu à l'aide du modèle Serveur/Client. L'architecture globale est présentée à la figure 1. Il existe deux rôles, à savoir *Eureka Server* et *Eureka Client*. *Eureka Client* est divisé en *Application Service* et *Application Client*, qui sont respectivement le fournisseur de services et le consommateur de services.

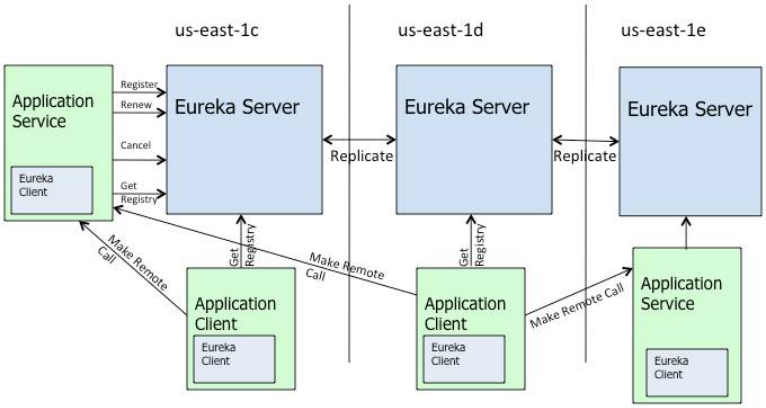

Figure 1 : l'architecture de Netflix Eureka [2]

Le serveur Eureka joue le rôle d'un annuaire de services qui sauvegarde les informations des clients enregistrés et fournit une interface au client pour obtenir des informations des autres services de l'annuaire. Les informations d'enregistrement sont répliquées sur toutes les instances d'*Eureka Server* pour synchroniser et maintenir la cohérence des données.

Le service Application en tant que fournisseur de services s'enregistre dans l'annuaire de services lorsqu'il démarre et se renouvelle toutes les 30 seconds (par défaut) en envoyant des pulsations au serveur. Tous les clients Eureka peuvent demander les informations des autres services enregistrés via des interfaces d'Eureka Serveur pour les appels à distance.

Par défaut, les clients Eureka utilisent Jersey[3] (un *RESTful Web Service* en Java) et JSON pour communiquer avec le serveur Eureka.

3.1.2 Consul

L'architecture de HashiCorp Consul est présentée à la figure 2. Tout d'abord, Consul peut prendre en charge plusieurs centres de données. Dans la figure 2, il y a deux centres de données interconnectés via Internet.

Dans le même centre de données, tous les nœuds sont appelés « *Agents* ». Un agent est un démon qui s'exécute sur chaque membre du *cluster* Consul. Un agent peut fonctionner en tant qu'agent Serveur ou agent Client. Les agents Serveurs sont l'endroit où les données sont stockées et répliquées, et les agents Clients s'occupent de vérifier l'état des services et de transférer toutes les demandes de découverte de services aux agent Serveurs.

3 Jersey : https://jersey.github.io/

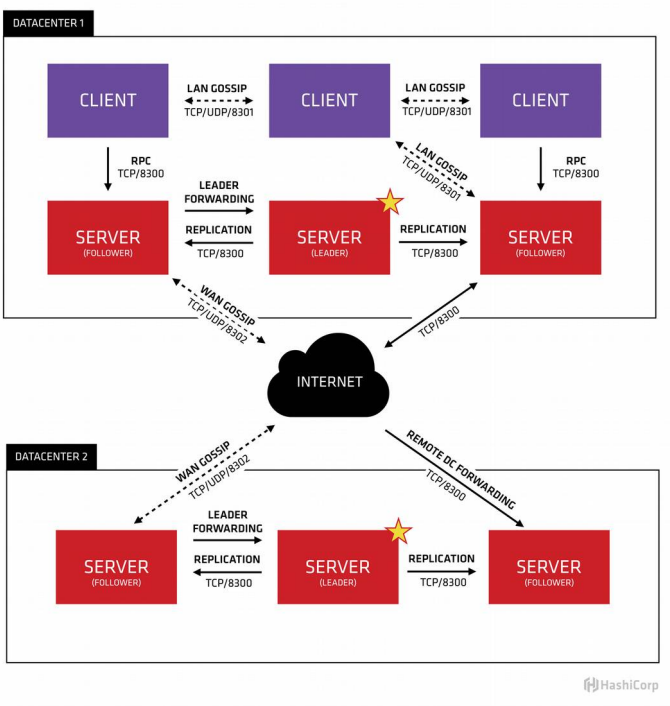

Figure 2 : l'architecture de HashiCorp Consul [3]

Un Serveur peut être soit un leader, soit un suiveur, soit un candidat. Chaque agent Serveur commence comme suiveur, et une fois que tous les serveurs sont démarrés, un leader de serveur est élu. C'est le leader qui traite toutes les requêtes des agents Clients et réplique leurs états à ses suiveurs. Si un suiveur reçoit une requête, il va la transférer au leader. Même si Consul peut fonctionner avec un seul serveur, il est recommandé d'en utiliser de 3 à 5 afin d'éviter les défaillances et la perte de données.

Consul utilise un protocole *Gossip* basé sur le protocole *SWIM* [4] pour gérer les membres et diffuser des messages au cluster. Consul se sert de *LAN (Local Area Network) Gossip Pool* au sein du même centre de données et *WAN (Wireless Area Network) Gossip Pool* entre différents centres. Ces deux pools communiquent via TCP ou UDP, mais la réplication des données utilise TCP.

En somme, un service peut être enregistré en utilisant une définition de service écrite dans les fichiers de configuration en JSON ou en utilisant l'API HTTP. Un microservice peut s'enregistrer en utilisant les API HTTP lors de démarrage. Une fois le service enregistré dans Consul, tout service qui se trouve dans le même cluster Consul peut le découvrir et le consommer via HTTP API ou le DNS.

3.2 Surveillance de l'état

La surveillance de l'état (« *Health Check* » en anglais) est un mécanisme pour détecter les instances de services qui sont en cours d'exécution mais incapables de traiter les demandes. De ce fait, l'annuaire de services ne doit pas acheminer les demandes vers les services défaillants. La surveillance de l'état est effectuée afin d'identifier et d'expulser les microservices inaccessibles dans l'annuaire de services.

3.2.1 Eureka

Par défaut, les clients Eureka envoient des pulsations toutes les 30 secondes au serveur Eureka afin de l'informer et renouveler leur statut. Une illustration est présentée à la figure 3.

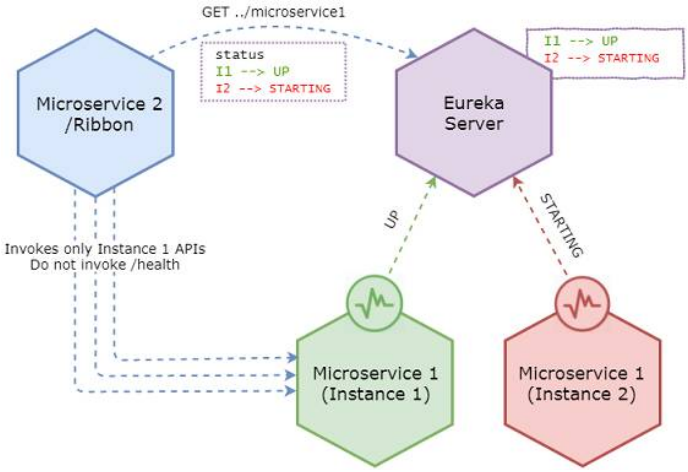

Figure 3 : Surveillance de l'état dans Eureka par envoi de pulsations [5]

Eureka Server expose certaines opérations de REST pour que les clients puissent publier des pulsations, par exemple :

PUT /eureka/apps/{app id}/{instance id}?status={status}

La commande {*instance id*} prend la forme « *hostname:app id:port* » où elle identifie une instance *Eureka Client* unique. *Eureka Server* reconnaît plusieurs états : « *UP* », « *DOWN* », « *STARTING* », « *OUT_OF_SERVICE* » et « *UNKNOWN* ». L'état d'un *Eureka Client* est déterminé par une implémentation de « *HealthCheckHandler* ». Il annonce par défaut que les services sont dans l'état « *UP* » lorsque les services sont bien enregistrés et exécutés. Par ailleurs, les autres services n'envoient pas de demandes vers des services dont l'état n'est pas « *UP* ».

Eureka Server enregistre l'état des services mais ne l'utilise pas directement. Quand les autres services, par exemple des répartiteurs de charge comme *Netflix Ribbon* pour prendre des décisions d'équilibrage de charge ou des outils de déploiement comme *Netflix Asgard* pour prendre des décisions de déploiement, interrogent les informations d'enregistrement, *Eureka Server* fournit en réponse l'état des services.

3.2.2 Consul

Consul peut attribuer une ou plusieurs procédures de surveillance de l'état aux services individuels ou à l'ensemble de l'hôte. L'état peut être vérifié par un appel HTTP au service, une requête *Ping* ou par un programme externe. Ensuite, en fonction du résultat de la vérification, Consul met le service ou tous les services dans un même hôte à l'un des trois états : « *Healthy* », « *Warning* » ou « *Critical* ».

Les services en « *Healthy* » *ou* « *Warning* » se comportent normalement, mais les services en « *Critical* » ne peuvent pas être découverts par le DNS. Si le service « *Critical* » n'est pas rétabli dans un état correct dans un délai fixé à l'avance, il peut être complètement désenregistré.

Il existe plusieurs types de surveillance de l'état [3] :

- **HTTP** : ce type de vérification effectue une requête HTTP GET vers une URL spécifiée périodiquement, par exemple toutes les 30 secondes. L'état du service dépend du code de réponse HTTP. Les code 2xx sont considérés comme dans le bon état « *Healthy* ». Le code 429 « *Too Many Requests* » représente une alarme « *Warning* ». Tous les autres codes représentent un échec « *Critical* ».

- **TCP** : ce type de vérification tente une connexion TCP périodiquement, par exemple toutes les 30 secondes, vers une adresse IP ou nom d'hôte et un port spécifiés. L'état du service dépend du succès de la tentative de connexion (c'est-à-dire l'acceptation des connexions ou si l'hôte est accessible ou pas). Si la connexion est acceptée, le statut est positif « *Healthy* », sinon le statut est « *Critical* ».
- **TTL (*Time To Live*)** : ce type de vérification conserve les derniers états connus du service pour un TTL configuré. Le statut doit être mis à jour périodiquement via l'interface HTTP. Si le renouvellement de l'état du service est échec dans un TTL donné, le statut « *Critical* » va être mis. Ce mécanisme est similaire à celui de Netflix Eureka présenté à la section 3.2.1.
- **Script** : ce type de vérification lance un service externe ou un script et interprète son code de sortie comme un nouvel état du service. 0 pour « *Healthy* », 1 pour « *Warning* », et tous les autres pour « *Critical* ». Ce type de vérification peut aussi impliquer la surveillance de l'utilisation d'espace disque, de mémoire, de CPU, etc.
- **Docker** : ce type de vérification est similaire à la vérification par script. La différence est qu'il utilise la commande « *Docker exec* » pour exécuter le script.

Il existe également le méthode **gRPC** pour les applications compatibles avec le standard *gRPC Health Check Protocol* et le méthode **Alias** de manière asynchrone. De plus, le protocole *Gossip* fournit la détection d'échec pour la surveillance de l'état du service.

3.3 Équilibrage de charge

L'équilibreur de charge agit en tant que « responsable de la circulation » installé dans les serveurs et achemine les requêtes des clients sur tous les serveurs capables de répondre à ces requêtes d'une manière qui maximise la vitesse et l'utilisation de la capacité. Il assure qu'aucun serveur n'est surchargé, ce qui pourrait dégrader la performance. Si un serveur tombe en panne, l'équilibreur de charge redirige le trafic vers les autres serveurs actifs plus libres.

Eureka et Consul possèdent le plus simple équilibreur de charge intégré qui effectue un équilibrage de charge basique de manière « *Round-Robin* ». Ce type d'équilibreur transfère la requête du client à chaque serveur en parcourant la liste de serveurs.

Mais pour réaliser un réel équilibrage de charge, Eureka et Consul ont chacun besoin d'un autre service spécialisé pour les aider. Par exemple, Eureka est toujours utilisé avec Netflix Ribbon[4], mais Consul intègre souvent Fabio[5] ou Nginx/HAProxy[6] .

3.4 Théorème CAP

Le théorème CAP [6] dit qu'il est impossible sur un système distribué de garantir en même temps la cohérence (C : *Consistency* en anglais), la disponibilité (A : *Availability* en anglais) et la tolérance au partitionnement (P : *Partition tolerance* en anglais).

La cohérence implique que tous les nœuds du système voient exactement les mêmes données au même moment. La disponibilité est la capacité de garantir que toutes les requêtes reçoivent une réponse. Avec la tolérance au partitionnement, en cas de morcellement en sous-réseaux, chaque partition doit pouvoir fonctionner de manière autonome et correcte.

4 Netflix Ribbon : https://github.com/Netflix/ribbon
5 Fabio : https://github.com/fabiolb/fabio
6 HAProxy : http://www.haproxy.org/

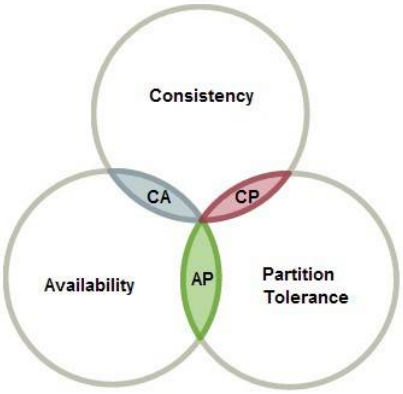

Figure 4 : Théorème CAP

La figure 4 illustre le concept de théorème CAP.

3.4.1 Eureka

D'après la théorème CAP, Eureka est un système **AP**, qui satisfait la disponibilité et la tolérance au partitionnement. Les services peuvent se retrouver dans des situations non prévues telles que des partitions de réseau ou des pannes de serveur. La haute disponibilité est atteinte à deux niveaux :

- Le *cluster* de serveur : la configuration comprend un cluster de *Eureka Servers*. Tant qu'un serveur est disponible, l'enregistrement et la découverte de services sont disponibles.
- Le cache côté client : les *Eureka Clients* récupèrent et mettent en cache les informations d'enregistrement venant de *Eureka Server*. Dans le cas où tous les serveurs tombent en panne, un *Eureka Client* conserve au moins le dernier snapshot de l'annuaire de services.

Cependant, il est possible que les informations trouvées ne soient pas les plus récentes, c'est-à-dire avec des incohérences entre les serveurs pendant une partition de réseau. Eureka n'utilise pas d'algorithme de consensus. De plus, Eureka a une mécanisme d'auto-protection pour minimiser cette incohérence et pour garantir la disponibilité lors d'une partition de réseau.

Les clients Eureka envoient des pulsations aux serveurs pour les renouveler, s'ils ne sont pas renouvelés plusieurs fois (90 secondes par défaut), ils expirent et sont retirés de l'annuaire de services. Cependant, l'auto-protection dans Eureka est une fonction par laquelle les *Eureka Servers* cessent d'expirer des instances de *Eureka Clients* lorsque les serveurs ne reçoivent pas de pulsations au-delà d'un certain seuil (15 % par défaut) parce qu'il est possible d'avoir une partition du réseau mais pas de défaillances de services.

3.4.2 Consul

D'après la théorème CAP, Consul est un système **CP**, qui satisfait la cohérence et la tolérance de partitionnement. Consul utilise un protocole de consensus pour assurer la cohérence. Le protocole de consensus est basé sur l'algorithme « *Raft* » [7] qui permet l'élection de leader, la réplication des transactions, le transfert des requêtes, etc.

En raison de la réplication de *Raft*, la performance est sensible à la latence du réseau. Pour cette raison, chaque centre de données élit un leader indépendant et maintient un ensemble de suiveurs disjoints. Les données sont partitionnées par le centre de données, donc un leader est responsable de ses propres données dans son centre de données. Lorsqu'une demande est reçue d'un centre de

données distant, cette demande est transmise au bon leader. Cette conception permet de réduire le temps de latence des transactions et d'augmenter la disponibilité sans sacrifier la cohérence.

4 Conclusion

Pour conclure, nous avons choisi deux solutions populaires de la découverte de services, Netflix Eureka et HashiCorp Consul, et nous les avons comparées selon plusieurs aspects. Les résultats sont résumés dans le Tableau 1. Il n'y a pas de solution « *Silver Bullet* », chacune a des avantages et des inconvénients. Le choix doit se faire en se basant sur les besoins du projet.

	Eureka	**Consul**
Langage	Développé en Java	Développé en Go
CAP	AP L'état de service est répliqué avec « best effort »	CP L'état de service est répliqué à l'aide de Raft
Surveillance de l'état	L'annuaire de service a un TTL et les clients envoient des pulsations	L'annuaire de service comporte une surveillance complexe de l'état, incluant par exemple la détection des fautes de *Gossip*
Lecture de données	Les lectures de données sont acheminées vers n'importe quel serveur, des données peuvent être obsolètes ou manquantes	Les lectures de données sont acheminées vers n'importe quel serveur, cohérentes par défaut, mais possible de demander données obsolètes
Échec	Échec si tous les serveurs sont hors service	Échec si la majorité des serveurs sont hors service
Interface d'exposition externe	HTTP	HTTP ou DNS
Intégration	Facile car intégré au projet en Java. Pour les projets dans un autre langage, Eureka peut être exécuté en tant que *Sidecar*	Langage indépendant
Divers	Travaille ensemble avec les autres services spéciaux de Netflix pour offrir des fonctionnalités supplémentaires telles que Netflix Ribbon, Netflix Zuul, etc.	Plus de fonctionnalités supplémentaires, par exemple gérer plusieurs centres de données, stockage clé/valeur, communication sécurisée, etc.

Tableau 1 : Résumé de comparaison Eureka vs. Consul

5 Références

[1] S. Haselböck, R. Weinreich, G. Buchgeher, "Decision Models for Microservices: Design Areas, Stakeholders, Use Cases, and Requirements.", 11th European Conference on Software Architecture (ECSA), Canterbury, UK, , pp. 155-170, September 11-15, 2017.

[2] Netflix Eureka Siteweb, https://github.com/Netflix/eureka

[3] HashiCorp Consul Siteweb, https://www.consul.io/

[4] A. Das, I. Gupta, A. Motivala, "SWIM: Scalable weakly-consistent infection-style process group membership protocol", Proceedings International Conference on Dependable Systems and Networks (DSN), pp. 303-312, 2002.

[5] F. Farook, "The Mystery of Eureka Health Monitoring", https://medium.com/@fahimfarookme/the-mystery-of-eureka-health-monitoring-5305e3beb6e9, Dec. 2017.

[6] S. Gilbert, Nancye A. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services", ACM SIGACT News, Volume 33, Issue 2, pp. 51-59, June 2002.

[7] D. Ongaro, J. Ousterhout, "In search of an understandable consensus algorithm", Proceedings of the USENIX Annual Technical Conference (ATC), pp. 305 – 320, June 2014.

9 rue Charles Fourier
91011 Évry Cedex
France
+33 (0)1 60 76 44 06

www.telecom-sudparis.eu