Évaluation et comparaison d'un sous-ensemble d'outils de la 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.

Avec un systèmes dynamique, par exemple des systèmes 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 important 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. Donc, dans ce document, on choisit deux solutions populaires : Netflix Eureka[1] et HashiCorp[2] Consul en tant que composant de la découverte de service et les compare en différents aspects.

Dans ce document, La section 2 explique la problématique. La section 3 détaille la comparaison entre les 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 : le processus d'un service enregistrant son localisation 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 en tant que client interrogeant l'annuaire de services pour connaître la localisation des services.

Les solutions d'enregistrement et de découverte de services lient souvent avec d'autres aspects à considérer : la surveillance de l'état de 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 Comparaisons

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

Dans cette section, on va les comparer par rapport à ses architectures, ses mécanismes de la surveillance et l'équilibrage de la charge, ses capacités de l'intégration, ainsi que ses applications du théorie CAP [2].

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

3.1 Architecture

Un composant de la découverte de services réalise primordialement les fonctionnalités fondamentales : l'enregistrement et la découverte de services.

3.1.1 Eureka

Eureka est conçu à l'aide du modèle Serveur/Client. L'architecture globale est présente dans la figure 1. Il existe deux rôles à savoir Eureka Serveur 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.

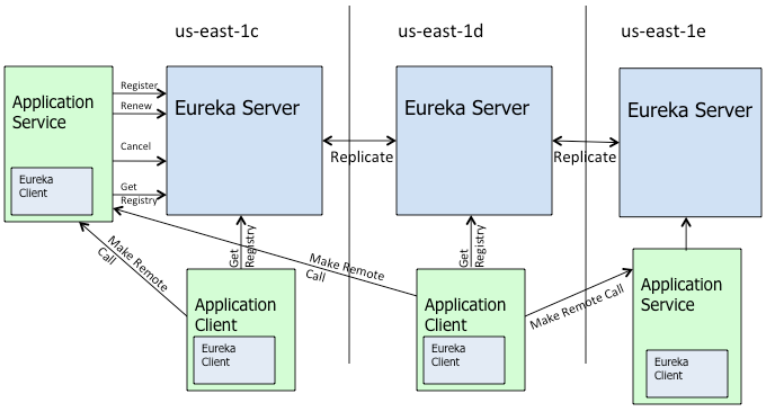

Figure 1 : l'architecture de Netflix Eureka [2]

Le Eureka Serveur joue un rôle de l'annuaire de services qui sauvegarde les informations de clients enregistrés et fournit une interface qu client pour obtenir des informations des autres services dans l'annuaire. Les informations d'enregistrement sont répliquées sur tous les instances d'Eureka Serveurs pour synchroniser et maintenir la cohérence de données.

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

Par défaut, les Eureka Clients utilise Jersey[3] (un *RESTful Web Service* en Java) et JSON pour communiquer avec Eureka Serveur.

3.1.2 Consul

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

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

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

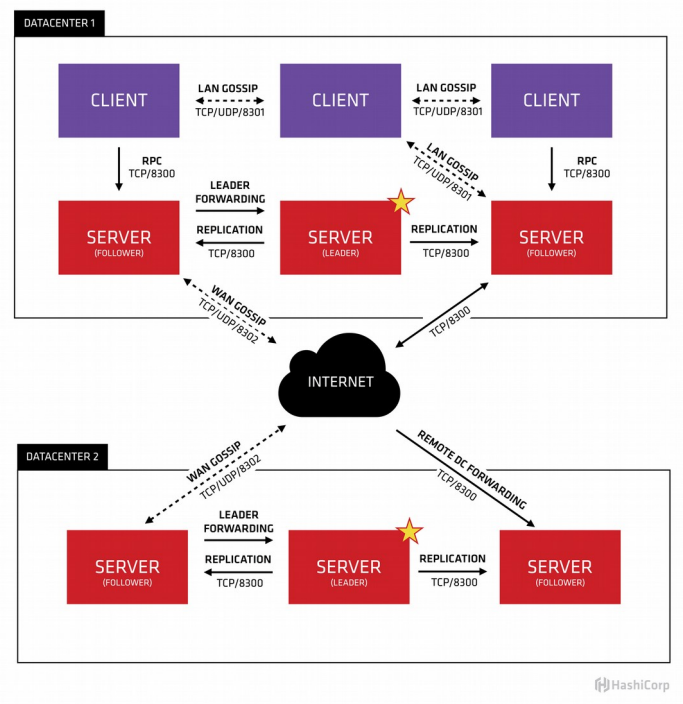

Figure 2 : l'architecture de HashiCorp Consul [3]

Un Serveur peut être soit un leader, soit un suiveur, soit un candidat. Chaque Agent Serveur commence comme un 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 Agent 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* qui est dans le même centre de données et *WAN (Wireless Area Network) Gossip Pool* entre différents centres. Ces deux se communiquent via TCP ou UDP, mais la réplication des données de font par TCP.

En somme, un service peut être enregistré en utilisant une définition de service écrit dans les fichiers de configuration de JSON ou en utilisant l'API HTTP. Le 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 en manière de HTTP API ou de 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 est en cours d'exécution mais incapable de traiter les demandes. De ce fait, l'annuaire de services ne doit pas acheminer les demandes vers les services défaillantes. La surveillance de l'état est effectués afin d'identifier et d'expulser les microservices inaccessibles dans l'annuaire de services.

3.2.1 Eureka

Par défaut, les Eureka Clients envoient des pulsations toutes les 30 seconds au Eureka Serveur afin de l'informer et renouveler de leur statut. Une illustration est présenté dans la figure 3.

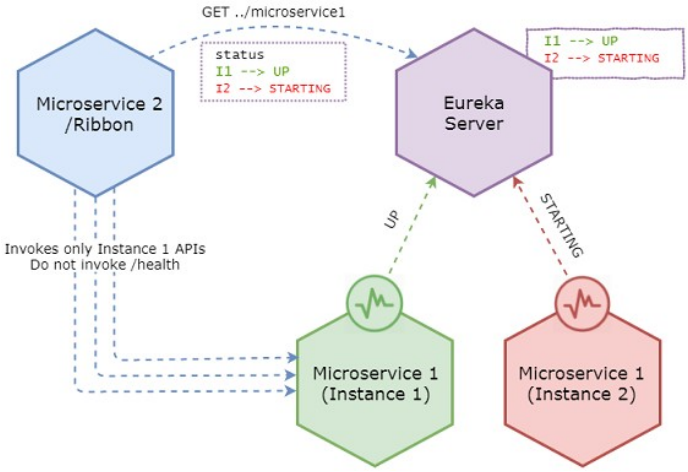

Figure 3 : La surveillance de l'état dans Eureka par envoyer des pulsations [5]

Eureka Serveur expose certaines opérations de REST pour que les clients peuvent 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 Serveur 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 de « *UP* » lorsque les services sont bien enregistrés et exécutés. En même temps, tous les autres services n'envoient pas de demandes vers lesquels dans des états autre que « *UP* ».

Le Eureka Serveur enregistre l'état de services mais ne l'utilise pas. 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, le Eureka Serveur répond également l'état de services.

3.2.2 Consul

Consul peut attribuer un 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 3 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 DNS. Si le service « *Critical* » n'est pas rétabli en bon état dans un délai, 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 est considéré comme dans le bon état « *Healthy* ». Le code 429 « *Too Many Requests* » représente une alarme « *Warning* ». Tous les autres code représente un échec « *Critical* ».
- **TCP** : ce type de vérification tente une connexion TCP périodiquement, par exemple toutes les 30 seconds, vers un IP/nom d'hôte et un port spécifiés. L'état du service dépend du succès de la tentative de connexion (c'est-à-dire l'accepte des connexions ou si l'hôte est accéssible 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 services pour un TTL configuré. Le statut doit être mis à jour périodiquement via l'interface HTTP. Si la renouvellement de l'état du service est échec dans un TTL donné, un « *Critical* » va être mis. Ce mécanisme est similaire à celui dans le Netflix Eureka présenté dans 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 peut aussi supporter la surveillance de l'utilisation d'espace disque, de mémoire, de CPU, etc.
- **Docker** : ce type de vérification est similaire à la vérification du script. La différence est qu'il utilise la commande « *Docker exec* » pour exécuter le script.

Il existe également le méthode **gRPC** pour les applications qui supporte le standard gRPC Health Check Protocol et le méthode **Alias** de manière asynchrone. En plus, la 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 quelque serveur tombe en panne, l'équilibreur de charge redirige le trafic ver les autres serveurs actifs plus libres.

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

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

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

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 angais).

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

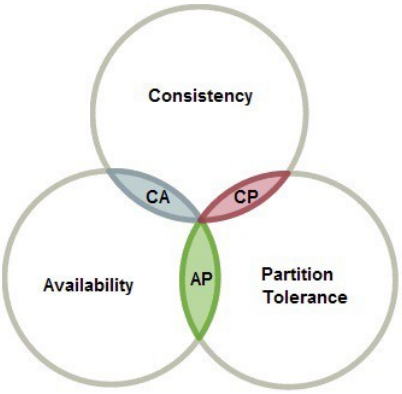

Figure 4 : Théorème CAP

La figure 4 illustre la concept de théorème CAP.

3.4.1 Eureka

D'après la théorème CAP, le Eureka est un système AP, qui satisfait la disponibilité et la tolérance de partitionnement. Les services peuvent se retrouver dans les situations non prévues tels 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 Serveurs. 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 Serveur. Dans le cas où tous les serveurs tombent en panne, le Eureka Client conserve au moins le dernier snapshot de l'annuaire de services.

Cependant, il est possible que les informations trouvées peuvent ne pas être les plus récentes, c'est-à-dire incohérent entre les serveurs pendant une partition de réseau. Eureka n'utilise pas des algorithme consensuel mais la plus de réplication. En plus, Eureka a une mécanisme d'auto-protection pour minimiser cette incohérence et pour garantir la disponibilité lors d'une partition de réseau difficile.

Les Eureka Clients envoient des pulsations aux serveurs pour les renouveler, s'ils ne sont pas renouvelés plusieurs fois (90 seconds par défaut), ils expirent et sont retiré de l'annuaire de services. Cependant, l'auto-protection dans Eureka est une fonction par laquelle les Eureka Serveurs cessent d'expirer des instances de Eureka Client lorsque les serveurs ne reçoivent pas de pulsations

au-delà d'un certain seuil (15 % par défaut) parce que il est possible d'une partition de réseau mais pas de défaillances de services.

3.4.2 Consul

D'après la théorème CAP, le Consul est un système CP, qui satisfait la cohérence et la tolérance de partitionnement. Consul utilise un protocole consensuel pour assurer la cohérence. Le protocole de consensus est basé sur une algorithme « *Raft* » [7] qui supporte 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 son propre 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, on a choisi deux solutions populaire de la découverte de services et les comparé en plusieurs aspects : Netflix Eureka et HashiCorp Consul. Les résultats sont résumées dans le Tableau 1. Il n'y a pas de solution « *Silver Bullet* », tous les deux ont des avantages et des inconvénients. Il faut choisir basé sur les besoins du projet.

	Eureka	**Consul**
Langage	Développé en Java	Développé en Go
CAP	AP système, l'état de service est répliqué avec « best effort »	CP système, 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 des surveillance de l'état complexe, inclut 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 intégré au projet en Java. Pour les projets en autres langages, Eureka exécute en tant que Sidecar	Langage indépendante

Divers	Travail ensemble avec les autres services spéciales de Netflix pour supporter des fonctionnalités supérieurs, par exemple, Netflix Ribbon, Netflix Zuul, etc.	Plus de fonctionnalités supérieurs, par exemple supporter plusieurs de centre de données, le stockage de clé/valeur, la communication sécuritée, 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.", Software Architecture - 11th European Conference, ECSA 2017, Canterbury, UK, , pages 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 of the 2002 International Conference on Dependable Systems and Networks DSN '02, 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, pages 51-59, June 2002.

[7] D. Ongaro, J. Ousterhout, "In search of an understandable consensus algorithm", USENIX ATC'14 Proceedings of the 2014 USENIX conference on USENIX Annual, Page 305 – 320, June 2014.

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

www.telecom-sudparis.eu