DJEMAI, Massyl

HADDED, Sawssen

Kubernetes : Services vs Ingress
Rapport d’éxposé
Programmation Comparée

Introduction

Aujourd'hui plus que jamais, il existe une multitude d'applications fortement sollicitées qui engendre naturellement moult sujets de scalabilité, de gestion des flux et des ressources.

Déployer et maintenir des applications à grande échelle sans se doter d'une infrastructure adaptée est assez compliqué. C'est pour cette raison que des technologies répondant à ces problématiques sont apparues ainsi que bon nombre d'outils permettant d'accomplir ces tâches de façon optimale et sûre.

Dans le cadre de cet exposé, nous nous intéresserons à la gestion de la conteneurisation et son orchestration, à travers Kubernetes. En mettant l'accent sur les différentes manières d'exposer des ressources dans un réseau.

Après une mise en contexte et une introduction aux concepts fondamentaux à la compréhension de Kubernetes, nous mettrons en évidence le clivage entre les notions de Service et d'Ingress en donnant dans chacun des cas les avantages et les inconvénients de l'approche.

La conteneurisation

Lorsque nous développons un logiciel ou une application, une bonne pratique consiste à utiliser les conteneurs, on parle également de "dockerisation" (Docker étant la plateforme de référence pour la conteneurisation).

Un conteneur est l'exécution par un gestionnaire de conteneurs d'une Image décrivant les éléments nécessaires à l'exécution d'un code source. L'image est représentée par un fichier décrivant les dépendances de l'application que l'on souhaite déployer.

Cette exécution est comparable à une virtualisation d'un système d'exploitation, mais avec un avantage qui est la portabilité et la légèreté.

Le fait de partager le noyau de la machine hôte réduit considérablement le poids que représente un conteneur (surtout si on le compare à une machine virtuelle).

Lorsque l'application que l'on souhaite déployer comporte plusieurs serveurs applicatifs ou briques logicielles indépendantes, une bonne façon de faire consiste a les isoler dans des conteneurs distincts.

Kubernetes

Kubernetes est une plateforme open source d'orchestration de conteneurs servant principalement à déployer, gérer et mettre à l'échelle des applications conteneurisées.

Kubernetes a été développé par des ingénieurs de Google qui se sont inspirés de Borg (système d'automatisation de déploiement interne à Google).

C'est en 2015 que Kubernetes devient open-source, Google en fit don à la Cloud Native Computing Fondation qu'il fonde en partenariat avec The Linux Foundation.

Il est possible de décrire l'architecture d'une instance de kubernetes (un cluster) avec le schéma simplifié suivant :

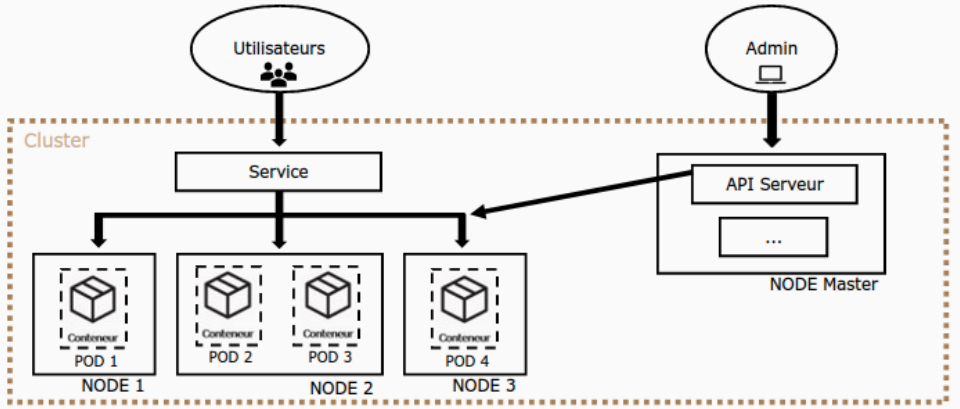

Figure 1 - Architecture d'un Cluster Kubernetes

Kubernetes se base sur un certain nombre d'abstractions qui incluent les POD, les SERVICES ou encore INGRESS.

Un POD représente une instanciation d'un déploiement, il encapsule un conteneur qui exécute le code source. Quant aux SERVICES et aux INGRESS, ils servent à exposer les PODS et donc l'application déployée.

L'exécution d'un cluster Kubernetes se fait sur des NODES qui sont les machines (physiques ou virtuelles) sur lesquelles les objets de Kubernetes se partagent les ressources.

Afin d'administrer le cluster Kubernetes, un NODE Maître est nécessaire, il est interrogeable à partir d'une API que consomme l'administrateur du cluster via un client web (par exemple kubectl).

La mise en œuvre d'un cluster Kubernetes passe par une approche déclarative via des fichiers au format YAML ou JSON. Le système de Kubernetes œuvre à maintenir l'état décrit par ces fichiers.

Les Services

Les services sont des abstractions qui définissent des POD et une politique permettant d'y accéder. Nous pouvons les voir comme des points d'entrées au cluster.

Kubernetes en dispose de trois catégories : NodePort, Load Balancer et ClusterIP.

ClusterIP (qui n'est pas l'objet principal du présent travail) s'occupe de la gestion de flux interne au sein du notre cluster, peut être utile pour la communication inter brique applicatives. Dans ce qui suit, une description plus détaillée des 2 types de services qui nous intéressent, à savoir NodePort et Load Balancer.

NodePort

NodePort est un Service qui permet d'aiguiller du trafic externe directement vers un POD en passant par un port unique réservé sur chaque Node du cluster. Ce port est choisi lors de la définition du Nodeport, entre 30000 et 32767.

Nodeport est utilisable nativement dans Kubernetes, il est implémenté directement dans un démon (k8s-proxy) présent dans chaque Noeud du cluster.

La répartition de charge entre les nœuds (s'il y en a besoin) est externalisée lorsqu'on utilise NodePort. Il faut donc passer par un répartiteur de charge tiers et le relier aux nœuds du cluster sur le port d'écoute du NodePort.

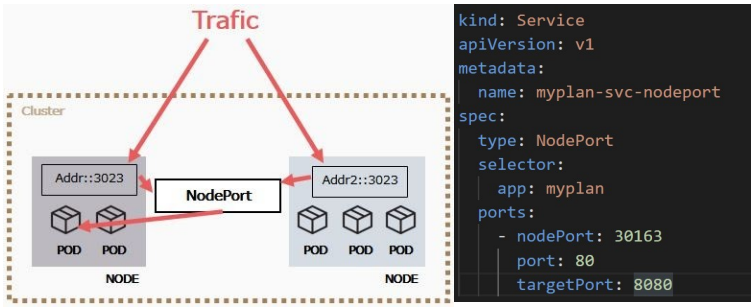

Figure 2 - Schéma du fonctionnement et de la déclaration du service NodePort

Load Balancer

Une alternative à la gestion de la répartition de charge en externe est l'utilisation du Service Load Balancer.

Comme pour Nodeport, Kubernetes réserve un port sur tous les nodes, et expose une ip et un port unique qui servent de point d'entrée au service qui se charge de répartir la charge.

Son utilisation suppose la présence d'une implémentation d'un répartiteur de charges répondant aux spécifications de Kubernetes que fournit généralement le fournisseur de Cloud.

Ce service prendra en compte l'éventuel changement d'IP des nœuds du cluster. Il est possible d'automatiser avec ce service la mise en échelle d'une application en spécifiant le nombre de réplicas nécessaires lors de la définition du service.

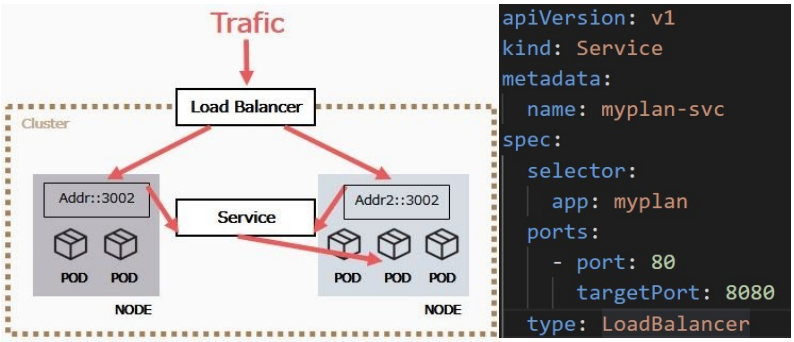

Figure 3 - Schéma du fonctionnement et de la déclaration du service Load Balancer

Ingress

Ingress est une ressource Kubernetes dont le rôle principal est de gérer le flux http/https provenant de l'extérieur du cluster et jouer le rôle de passerelle vers les services du cluster qui eux même ont leur stratégie de redirection vers les PODS comme vu précédemment.

Il est idéal pour l'architecture applicative orientée microservices. Sa configuration est proche de la configuration d'un proxy inversé, définissant un hôte virtuel avec sa propre ip et son propre

nom de domaine pour y accéder. La notion d'Ingress se rapproche d'ailleurs de celle de passerelle d'API (comme la librairie Apache Zuul).

Exemple concret : supposons qu'on ait une application de gestion de l'université avec différents composants : Inscription, Plannings horaires, Gestion de la bibliothèque.

Il est possible de créer 3 applications distinctes (avec des langages distincts), donc scalables séparément. Grâce à Ingress, il sera possible d'accéder à chacun des services de manière transparente par la même URL.

Son utilisation nécessite la présence d'un Ingress contrôler fourni généralement par fournisseur de cloud.

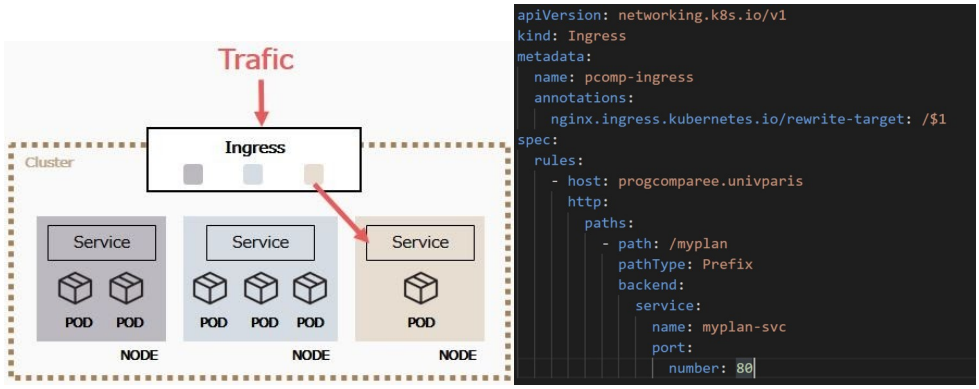

Figure 4 - Schéma du fonctionnement et de la déclaration du service Ingress

Conclusion

Kubernetes offre la possibilité d'exposer une application de diverses manières, en étudiant chacune d'entre elles, on a pu en conclure que chacune des approches était intéressante dans un certain contexte. Typiquement, si la priorité est la conception de micro-services alors l'approche par Ingress est idéale.

Pour résumer, ci-dessous un tableau comparatif des principales caractéristiques des ressources Kubernetes que nous avons étudié dans le cadre de ce travail :

	NodePort	Load Balancer	Ingress
Transmettre une requête a un POD	✓	✓	✓
Garantit la transmission	✓	✓	✓
Dépendant du cloud provider (coût)	✗	✓	✓
API Gateway	✗	✗	✓
Microservice	✗	✗	✓

Figure 5 - Tableau récapitulatif

Références

Kubernetes documentation

Kubernetes NodePort vs LoadBalancer vs Ingress ? When should I use what ?

Network Security Protocols