CAVARLE Théo   
DOUMALIN Kylian  
PICHAIMANI Sayan  
SU Brian

Projet Conteneurisation et Orchestration

Une image contenant texte

Description générée automatiquement

2022-2023

Sommaire :

Explications du projet :

L’utilisation des Conteneurs :

L’hébergement sur Kubernetes :

La création des Pods :

La Sécurité :

Explications du projet :

Le projet consiste à déployer, maintenir et mettre à jour plusieurs micro-services REST et des applications web dans un environnement conteneurisé sur Kubernetes pour garantir la haute disponibilité des applications. Les applications et services sont disponibles sur le Git du cours et sont décrits dans le fichier docker-compose.

Pour réaliser le projet, il faut créer les fichiers pour générer des images de conteneur de chaque application en utilisant les Dockerfiles fournis. Les images devront ensuite être envoyées vers un repo d'images privé. Ensuite, les applications doivent être déployées dans Kubernetes avec les paramètres nécessaires pour chaque application tels que la taille du cluster, le nombre de nœuds, le nombre de répliques, les réseaux virtuels, les volumes physiques, les pods, les services, les ingress, etc.

Le cluster de production doit être un cluster manager sur Azure (AKS). Pour assurer la performance, chaque pod doit avoir une limite de consommation CPU, mémoire et disque ainsi que des demandes de ressources définies. Les pods doivent également redémarrer automatiquement s'ils tombent.

Enfin, pour surveiller le cluster, il faut mettre en place des sondes de préparation et des sondes de vivacité, ainsi que des métriques pour mesurer l'état du cluster, les ressources des nœuds et des pods, et le travail du plan de contrôle. Les données seront affichées dans un Prometheus installé sur le cluster.

Il faut également s'assurer de la sécurité des applications en les rendant accessibles uniquement par le protocole https de l'extérieur. Pour cela, il faut générer un certificat SSL auto-signé et l'associer à votre cluster. Les applications envoient des logs dans la sortie standard, il faut donc mettre en place une pile complète (EFK ou ELK) pour une gestion de logs avancée.

Enfin, pour automatiser le déploiement sur d'autres clusters, il faut mettre en place un système de déploiement automatisé via HELM. Le paquet HELM doit contenir la création des pods, déploiements, services, ingress mais pas de contrôleur d'ingress.

L’hébergement sur Kubernetes :

L'hébergement sur Kubernetes consiste à déployer des applications dans un environnement conteneurisé et à les gérer à l'aide de l'orchestrateur Kubernetes. Kubernetes permet de gérer la répartition des ressources et des charges de travail sur un cluster de nœuds. Il s'agit d'un système de gestion de conteneurs qui permet de déployer, de mettre à jour et de surveiller les applications dans un environnement de production.

Pour héberger des applications sur Kubernetes, il est nécessaire de créer des images de conteneur pour chaque application et de les stocker dans un registre d'images. Ensuite, ces images peuvent être utilisées pour créer des pods qui sont des groupes de conteneurs liés qui fonctionnent ensemble sur un seul nœud de cluster. Les pods peuvent être répliqués pour permettre la haute disponibilité des applications.

Il est également nécessaire de configurer les ressources de chaque application, telles que la quantité de CPU et de mémoire nécessaire pour chaque pod, ainsi que les volumes de stockage nécessaires pour les données persistantes.

Kubernetes permet également de configurer les services, qui sont utilisés pour exposer les applications à l'extérieur du cluster. Les services peuvent être configurés pour diriger le trafic vers les pods qui exécutent l'application, et peuvent également être configurés pour équilibrer la charge du trafic entre les pods répliqués.

En somme, l'hébergement sur Kubernetes consiste à déployer des applications dans un environnement de production conteneurisé, à les configurer et à les gérer à l'aide de Kubernetes, afin de garantir la haute disponibilité et la scalabilité des applications.

La création des Pods :

Une image contenant texte

Description générée automatiquement

Api-applicants

Api-identity

Api-jobs

Database

Rabbitmq

User-data

Web

Chaque fichier à une utilité :

api-applicants : Ce fichier est utilisé pour configurer et déployer une API pour la gestion des candidatures. Il peut inclure des informations telles que les URL de base, les ports, les configurations de sécurité, les informations de connexion à la base de données, etc.

api-identity : Ce fichier est utilisé pour configurer et déployer une API d'authentification et d'autorisation. Il peut inclure des informations telles que les URL de base, les ports, les configurations de sécurité, les informations de connexion à la base de données, etc.

api-jobs : Ce fichier est utilisé pour configurer et déployer une API pour la gestion des offres d'emploi. Il peut inclure des informations telles que les URL de base, les ports, les configurations de sécurité, les informations de connexion à la base de données, etc.

database : Ce fichier est utilisé pour configurer et déployer une base de données. Il peut inclure des informations telles que le type de base de données, le nom d'utilisateur, le mot de passe, le port, l'hôte, etc.

rabbitmq : Ce fichier est utilisé pour configurer et déployer un serveur RabbitMQ, qui est un système de messagerie asynchrone. Il peut inclure des informations telles que les URL de base, les ports, les configurations de sécurité, etc.

user-data : Ce fichier est utilisé pour stocker des informations sur les utilisateurs de l'application, telles que leur nom, leur adresse e-mail, leur adresse, etc.

web : Ce fichier est utilisé pour configurer et déployer une application web. Il peut inclure des informations telles que les URL de base, les ports, les configurations de sécurité, les informations de connexion à la base de données, etc.

Une image contenant texte

Description générée automatiquement



Une image contenant texte

Description générée automatiquement

Explication de ce code : (Ici il s’agit de l’applicants-api)

Ce code est utilisé pour créer un déploiement et un service pour une application appelée "applicants-api". Le déploiement spécifie les détails pour créer les répliques de conteneurs de l'application, tels que le nom du conteneur, l'image Docker à utiliser et les variables d'environnement pour la connexion à la base de données RabbitMQ. Le déploiement utilise également les spécifications pour configurer les limites et les demandes de ressources pour le conteneur.

Le projet demandait d’allouer des ressources spécifiques à chaque pods.

On utilise resources pour le spécifier :

Requests : pour les ressources à allouer

Limits : pour la limite de ressource

La dernière ligne "restartPolicy" spécifie que les conteneurs doivent être relancés en cas d'échec.

Pour les affinités, on utilise affinity :

PodAffinity : pour spécifier des règles d’affinité entre les pods et d’autres pods dans un même cluster.

RequiredDuringSchedulingIgnoredDuringExecution : Le pod est planifié sur un nœud qui respecte l’affinité demandé, si l’affinité ne peut être respectée lors de l’exécution, elle est ignorée.

Labelselector : définie les pods ciblés pour l’affinité

MatchExpressions : liste d’expression de correspondance lors de la sélection d’étiquette

Key : clé de l’étiquette pour la correspondance

Operator : l’opérateur de correspondance (in)

Values : la liste des valeurs à utiliser (ici le nom de nos pods)

TopologyKey : la clé de topology à utiliser pour l’affinité (ici kubernetes.io/hostname, pour spécifier que les pods doivent être planifiés sur les mêmes nœuds.

Le service spécifie le nom du service et les détails pour créer un point de terminaison pour accéder à l'application. Il spécifie également comment les connexions doivent être acheminées vers les conteneurs. Le service est configuré pour écouter sur le port 80 pour le trafic entrant et sortant. Ensemble, le déploiement et le service garantissent que l'application "applicants-api" est correctement déployée et accessible.

Hébergement sur Lens :

Lens est un outil de gestion de conteneurs Kubernetes qui permet de gérer, déployer et surveiller les conteneurs Docker. Il permet d’observer une couche d'abstraction supplémentaire pour la gestion des conteneurs et des clusters de conteneurs à grande échelle. Elle permet d’observer si nos pods ont bien été déployer en local. Grace a la partie graphique, on peut facilement observer l’état des conteneurs, des déploiements ainsi que les erreurs des déploiements.

Pour pouvoir héberger un pods ou un service, il faut lancer la commande suivante :

  
Cette commande permet de créer ou de mettre à jour une ressource Ingress, qui permet de configurer des règles pour diriger le trafic entrant vers des services dans le cluster Kubernetes.

Une image contenant texte, écran, capture d’écran

Description générée automatiquement

Dans la partie Lens, quand on clique sur l’onglet « Pods », nous pouvons observer tous les pods qui ont été déployer

Une image contenant texte

Description générée automatiquement

La sécurité :

Dans ce projet, la sécurité consiste à rendre les applications accessibles uniquement par le protocole HTTPS depuis l'extérieur, en générant un certificat SSL auto-signé et en l'associant au cluster. Cela garantit que les communications entre le client et le serveur sont cryptées et sécurisées.

De plus, il est important de s'assurer que les images de conteneurs utilisées dans le cluster sont fiables et sécurisées. Il est donc recommandé de les construire à partir de sources sûres et de mettre en place des politiques de sécurité pour contrôler l'accès aux images.

Il est également recommandé de configurer des règles de sécurité pour protéger les ressources du cluster, telles que les secrets, les configurations et les fichiers de journalisation.

Enfin, il est important de maintenir le cluster à jour avec les dernières mises à jour de sécurité pour éviter les failles de sécurité connues.

Pour générer le certificat, j’ai tout d’abord installé Open SSL :

<https://slproweb.com/products/Win32OpenSSL.html>

Ensuite j’ai créé un fichier openssl.conf avec des informations pour le certificat

Une image contenant texte

Description générée automatiquement

Ensuite la commande pour générer les clés privée, publique et le certificat avec le fichier openssl.conf

Req : pour utiliser la gestion de requêtes Openssl.

-x509 : propriété pour spécifier que nous voulons générer un certificat auto-signé au lieu de générer une demande de signature de certificat.

-nodes : pas de mot de passe pour protéger la clé privée.

-days : propriété pour spécifier la durée de validité du certificat.

-newkey rsa:2048 : on génère une nouvelle paire de clés privée/publique RSA de 2048 bits.

-keyout tls.key : propriété pour spécifier le nom du fichier de la clé privée

-out tls.crt : propriété pour spécifier le nom du fichier pour le certificat

-config : spécifie le fichier de config OpenSSL

Une image contenant texte

Description générée automatiquement

Tls.key pour la clé privée et tls.crt pour le certificat

Une image contenant texte

Description générée automatiquement

Commande pour créer un objet Secret TLS dans Kubernetes :



On précise le fichier de clé privée et le fichier du certificat

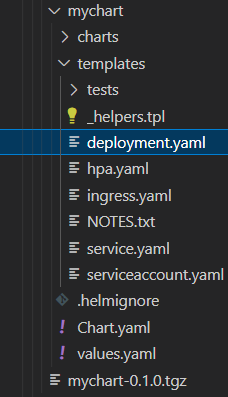
Helm :

Helm est un outil open source qui facilite la gestion des applications Kubernetes en permettant de les empaqueter, de les déployer et de les mettre à jour de manière reproductible et cohérente sur plusieurs clusters Kubernetes. Les paquets d'application créés avec Helm sont appelés "charts". Les charts sont des fichiers de configuration qui contiennent des informations sur l'application à déployer, ainsi que sur les ressources Kubernetes nécessaires pour exécuter cette application. Helm permet également de gérer les dépendances entre les différentes applications et de versionner les différentes versions d'une application. En résumé, Helm facilite grandement le déploiement et la gestion des applications sur Kubernetes.

Dans ce projet nous allons utiliser Helm pour automatiser les déploiements et aussi réaliser un bilan de santé avec Grafana et Prometheus.

Pour automatiser les déploiements il faudra créer un paquet Helm qui déploie des pods, des déploiements et des services.

Un dossier qui contiendra la chart se créer, puis on initialisera la chart avec la commande suivante : helm create mychart



Puis nous modifierons les fichiers deployment.yaml, service.yaml et ingress.yaml.

Ensuite nous vérifierons que notre chart est correctement formaté avec la commande : helm lint mychart

Et on emballera la chart avec la commande : helm package mychart

Ensuite le paquet est prêt à être déployer sur un cluster.

Pour l’installation de Grafana et Prometheus, nous utiliserons les commandes :

helm install prometheus stable/prometheus

helm install grafana stable/grafana



Une fois que vous avez installé Grafana et Prometheus, vous devez configurer Grafana pour se connecter à Prometheus en tant que source de données.