Formation Kubernetes - Travaux pratiques

TP2 - INSTALLATION ET CONFIGURATION DE KUBERNETES

Objectif : Au cours de ce TP nous allons passer rapidement en revue deux manières de mettre en place Kubernetes avec Minikube :

Découverte de Kubernetes
Installer le client CLI kubectl

kubectl est le point d'entrée universel pour contrôler tous les types de cluster kubernetes. C'est un client en ligne de commande qui communique en **REST** avec **l'API** d'un cluster.

Nous allons explorer **kubectl** au fur et à mesure des TPs. Cependant à noter que :

kubectl peut gérer plusieurs clusters/configurations et switcher entre ces configurations
La méthode d'installation importe peu.

Pour installer **kubectl** sur Ubuntu nous ferons simplement: **sudo snap install kubectl --classic**.

Pour installer kubectl sur Windows consulter la documentation officielle : https://kubernetes.io/fr/docs/tasks/tools/install-kubectl/

Faites **kubectl version** pour afficher la version du client kubectl.

Installer Minikube
Minikube est la version de développement de Kubernetes (en local) la plus répendue. Elle est maintenue par la cloud native foundation et très proche de Kubernetes upstream. Elle permet de simuler un ou plusieurs nœuds de cluster sous forme de conteneurs docker ou de machines virtuelles.

Pour installer minikube la méthode recommandée est indiquée ici: https://minikube.sigs.k8s.io/docs/start/

Nous utiliserons classiquement docker comme runtime pour minikube (les nœuds k8s seront des conteneurs simulant des serveurs). Ceci est, bien sur, une configuration de développement. Elle se comporte cependant de façon très proche d'un véritable cluster.

Si Docker n'est pas installé, installer Docker avec la commande en une seule ligne : **curl -fsSL https://get.docker.com | sh**, puis ajoutez-vous au groupe Docker avec sudo usermod -a -G docker <votrenom>, et faites sudo reboot pour que cela prenne effet.

Pour lancer le cluster faites simplement: **minikube start**.

Minikube configure automatiquement **kubectl** (dans le fichier ~/.kube/config) pour qu'on puisse se connecter au cluster de développement.

Testez la connexion avec **kubectl get nodes**.

Affichez à nouveau la version **kubectl** version. Cette fois-ci la version de **kubernetes** qui tourne sur le cluster actif est également affichée. Idéalement le client et le cluster devrait être dans la même version mineure par exemple 1.20.x.

Bash completion
Pour permettre à kubectl de compléter le nom des commandes et ressources avec <Tab> il est utile d'installer l'autocomplétion pour Bash :

Sur Linux

sudo apt install bash-completion

source <(kubectl completion bash)

echo "source <(kubectl completion bash)" >> ${HOME}/.bashrc

Sur windows 👉 https://kubernetes.io/fr/docs/tasks/tools/install-kubectl/

Explorons notre cluster k8s

Le cluster K8s est constitué d'objets divers qui permettent de décrire des applications, des tâches de calcul, services et droit d'accès. Nous allons dans un premier temps explorer le cluster :

Listez les nodes pour récupérer le nom de l'unique node **kubectl get nodes** puis affichez ses caractéristiques avec la commande **kubectl describe node/minikube**.

La commande **get** est générique et peut être utilisée pour récupérer la liste de tous les types de ressources.

De même, la commande **describe** peut s'appliquer à tout objet **k8s**. On doit cependant préfixer le nom de l'objet par son type **(ex : node/minikube ou nodes minikube)** car **k8s** ne peut pas deviner ce que l'on cherche quand plusieurs ressources ont le même nom.

Pour afficher tous les types de ressources à la fois que l'on utilise : **kubectl get all**

Il semble qu'il n'y a qu'une ressource dans notre cluster. Il s'agit du service d'**API Kubernetes**, pour qu'on puisse communiquer avec le cluster.

En réalité, il y en a généralement d'autres cachés dans les autres **namespaces**. En effet les éléments internes de **Kubernetes** tournent eux-mêmes sous forme de services et de daemons **Kubernetes**. Les namespaces sont des groupes qui servent à isoler les ressources de façon logique et en termes de droits (avec le Role-Based Access Control (RBAC) de Kubernetes).

Pour vérifier cela on peut :

Afficher les namespaces : **kubectl get namespaces**

Un cluster Kubernetes a généralement un namespace appelé default dans lequel les commandes sont lancées et les ressources créées si on ne précise rien. Il a également aussi un namespace kube-system dans lequel résident les processus et ressources système de k8s. Pour préciser le namespace on peut rajouter l'argument -n à la plupart des commandes k8s.

Pour lister les ressources liées au **kubectl get all -n kube-system**.

Ou encore : **kubectl get all --all-namespaces** (peut être abrégé en kubectl get all -A) qui permet d'afficher le contenu de tous les namespaces en même temps.

Pour avoir des informations sur un namespace : **kubectl describe namespace/kube-system**

Déployer une application en CLI

Nous allons maintenant déployer une première application conteneurisée. Le déploiement est un peu plus complexe qu'avec Docker, en particulier car il est séparé en plusieurs objets et plus configurable.

Pour créer un déploiement en ligne de commande, on peut lancer par exemple: **kubectl create deployment demonstration --image=monachus/rancher-demo**.

Cette commande crée un objet de type **deployment**. Nous pouvons étudier ce déploiement avec la commande **kubectl describe deployment/demonstration**.

Notez la liste des événements sur ce déploiement en bas de la description.

De la même façon que dans la partie précédente, listez les pods avec **kubectl**. Combien y en a-t-il ?

Agrandissons ce déploiement avec **kubectl scale deployment demonstration --replicas=5**

kubectl describe deployment/demonstration permet de constater que le service est bien passé à 5 replicas.

Observez à nouveau la liste des évènements, le scaling y est enregistré…

Listez les pods pour constater

A ce stade impossible d'afficher l'application : le déploiement n'est pas encore accessible de l'extérieur du cluster. Pour régler cela nous devons l'exposer grace à un service :

kubectl expose deployment demonstration --type=NodePort --port=8080 --name=demonstration-service

Affichons la liste des services pour voir le résultat: **kubectl get services**

Un service permet de créer un point d'accès unique exposant notre déploiement. Ici nous utilisons le type **Nodeport** car nous voulons que le service soit accessible de l'extérieur par l'intermédiaire d'un forwarding de port.

NB : Forwarding de port : rediriger des paquets réseaux reçus sur un port donné d'un ordinateur ou un équipement réseau vers un autre ordinateur ou équipement réseau sur un port donné.

Avec **minikube** ce forwarding de port doit être concrêtisé avec la commande **minikube service demonstration-service**. Normalement la page s'ouvre automatiquement et nous voyons notre application.

Sauriez-vous expliquer ce que l'app fait que nous venons de déployer ?

Pour le comprendre ou le confirmer, diminuez le nombre de réplicats à l'aide de la commande utilisée précédemment pour passer à 5 réplicats. Que se passe-t-il ?

Une autre méthode pour accéder à un service (quel que soit sont type) en **mode développement** est de forwarder le traffic par l'intermédiaire de **kubectl** (et des composants kube-proxy installés sur chaque noeuds du cluster).

Pour cela on peut par exemple lancer: **kubectl port-forward svc/demonstration-service 8080:8080 --address 127.0.0.1**

Vous pouvez désormais accéder à votre app via via **kubectl sur: http://localhost:8080**.

Quelle différence avec l'exposition précédente via minikube ?

Simplifier les lignes de commande k8s

Pour gagner du temps on dans les commandes Kubernetes on peut définir un alias: **alias kc='kubectl'**
(à mettre dans votre .bash_profile en faisant echo "alias kc='kubectl'" >> ~/.bash_profile, puis en faisant source ~/.bash_profile).

Vous pouvez ensuite remplacer **kubectl** par **kc** dans les commandes.

Également pour gagner du temps en ligne de commande, la plupart des mots-clés de type Kubernetes peuvent être abrégés :

services devient svc
deployments devient **deploy**
etc.

Liste complète