KUBERNETES-OPS

CONCERNANT CES SUPPORTS DE COURS

SUPPORTS DE COURS RÉALISÉS PAR OSONES

https://osones.com

Logo Osones

- Sources : https://github.com/Osones/formations/
- HTML/PDF : https://osones.com/formations/

Licence Creative Commons BY-SA 4.0

KUBERNETES : INSTALLATION

INSTALLATION DE KUBERNETES

- De nombreuses ressources présentes pour le déploiement de Kubernetes dans un environnement de production
- Un des outils est kubeadm utilisé pour rapidement démarrer un cluster kubernetes

INSTALLATION DE KUBERNETES

- Certains pré-requis sont nécessaires avant d'installer Kubernetes :
  - Désactiver le swap
  - Assurer que les ports requis soient ouverts : https://kubernetes.io/docs/setup/independent/install-kubeadm/#check-required-ports
  - Installer Docker : https://docs.docker.com/install

INSTALLATION DE KUBERNETES

- Installer les composants Kubernetes (kubeadm, kubectl, kubelet) : https://kubernetes.io/docs/setup/independent/install-kubeadm/
- Exécuter `sudo kubeadm init` sur le noeud master
- Exécuter `sudo kubeadm join` sur les autres noeuds (avec le token fournir par la commande `kubeadm init`)
- Copier le fichier de configuration généré par `kubeadm init`
- Installer le plugin Réseau (Dans notre cas nous utiliserons Weave Net)

INSTALLATION DE KUBERNETES

- Il existe des solutions managées pour Kubernetes :
  - Azure Kubernetes Service : https://azure.microsoft.com/en-us/services/kubernetes-service/
  - Google Kubernetes Engine : https://cloud.google.com/kubernetes-engine/
  - Elastic Kubernetes Services: https://aws.amazon.com/eks/
  - Docker for mac : https://docs.docker.com/docker-for-mac/kubernetes/

INSTALLATION DE KUBERNETES

- Via Ansible : kubespray https://github.com/kubernetes-incubator/kubespray
- Via Terraform : https://github.com/poseidon/typhoon
- Il existe aussi des projets open source basés sur le langage Go :
  - kube-aws : https://github.com/kubernetes-incubator/kube-aws
  - kops : https://github.com/kubernetes/kops

INTRODUCTION À SONOBUOY

- Outil de conformité de clusters Kubernetes
- Permet de facilement générer des données de diagnostics pour les applications déployées
- https://github.com/heptio/sonobuoy/

KUBECTL : ADVANCED USAGE

- Il est possible de mettre à jour un service sans incident grâce ce qui est appelé le *rolling-update*.
- Avec les *rolling updates*, les resources qu'expose un objet `Service` se mettent à jour progressivement.
- Seuls les objets `Deployment`, `DaemonSet` et `StatefulSet` support les *rolling updates*.
- Les arguments `maxSurge` et `maxUnavailabe` définissent le rythme du *rolling update*.
- La commande `kubectl rollout` permet de suivre les *rolling updates* éffectués.

KUBECTL : ADVANCED USAGE

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  selector:
    matchLabels:
      app: frontend
  replicas: 2
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
    type: RollingUpdate
  template:
    metadata:
      name: nginx
      labels:
```

KUBECTL : ADVANCED USAGE

```
$ kubectl create -f nginx.yaml --record
deployment.apps/nginx created
```

KUBECTL : ADVANCED USAGE

- Il est possible d'augmenter le nombre de pods avec la commande kubectl scale :

```
kubectl scale --replicas=5 deployment nginx
```

- Il est possible de changer l'image d'un container utilisée par un *Deployment* :

```
kubectl set image deployment nginx nginx=nginx:1.15
```

KUBECTL : ADVANCED USAGE

- Dry run. Afficher les objets de l'API correspondant sans les créer :

```
kubectl run nginx --image=nginx --dry-run
```

- Démarrer un container en utiliser une commande différente et des arguments différents :

```
kubectl run nginx --image=nginx \
--command -- <cmd> <arg1> ... <argN>
```

- Démarrer un Cron Job qui calcule π et l'affiche toutes les 5 minutes :

```
kubectl run pi --schedule="0/5 * * * ?" --image=perl --restart=OnFailure \
-- perl -Mbignum=bpi -wle 'print bpi(2000)'
```

KUBECTL : ADVANCED USAGE

- Se connecter à un container:

```
kubectl run -it busybox --image=busybox -- sh
```

- S'attacher à un container existant :

```
kubectl attach my-pod -i
```

- Accéder à un service via un port :

```
kubectl port-forward my-svc 6000
```

KUBECTL : LOGGING

- Utiliser kubectl pour diagnostiquer les applications et le cluster kubernetes :

```
kubectl cluster-info
kubectl get events
kubectl describe node <NODE_NAME>
kubectl  logs (-f) <POD_NAME>
```

KUBECTL : MAINTENANCE

- Obtenir la liste des noeuds ainsi que les informations détaillées :

```
kubectl get nodes
kubectl describe nodes
```

KUBECTL : MAINTENANCE

- Marquer le noeud comme *unschedulable* (+ drainer les pods) et *schedulable* :

```
kubectl cordon <NODE_NAME>
kubectl drain <NDOE_NAME>
kubectl uncordon <NODE_NAME>
```

MONITORING

LIMITRANGES

- l'objet LimitRange permet de définir les valeurs minimum et maximum pour les ressources utilisées par les containers et les pods
- l'objet LimitRange ne s'applique dans un seul namespace mais peut être utilisé pour d'autres namespaces
- les limites spécifiées s'appliquent à chaque pod/container créé dans le namespace
- le LimitRange ne limite pas le nombre total de ressources disponibles dans le namespace

LIMITRANGES

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: limit-example
spec:
  limits:
  - default:
      memory: 512Mi
    defaultRequest:
      memory: 256 Mi
    type: Container
```

RESOURCEQUOTAS

- un objet ResourceQuota limite le total des ressources de calcul consommées par les pods ainsi que le total de l'espace de stockage consommé par les PersistentVolumeClaims dans un namespace
- il permet aussi de limiter le nombre de pods, PVC et autres objets qui peuvent être créés dans un namespace

RESOURCEQUOTAS

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: cpu-and-ram
spec:
  hard:
    requests.cpu: 400m
    requests.memory: 200Mi
    limits.cpu: 600m
    limits.memory: 500Mi
```

KUBERNETES : SECURITÉ ET CONTROLE D'ACCÈS

AUTHENTICATION & AUTORISATION

- RBAC (Role Based Access Control)
- ABAC (Attribute-based access control)
- WebHook
- Certificates
- Token

RBAC

3 entités sont utilisées :

- Utilisateurs représentés par les Users ou les ServiceAccounts
- Resources représentées par les Deployments, Pods, Services, etc...
- les différentes opérations possibles :
  create, list, get, delete, watch, patch

SERVICE ACCOUNTS

- Objet Kubernetes permettant d'identifier une application s'éxe
  pod
- Par défaut, un ServiceAccount par namespace
- Le ServiceAccount est formatté ainsi :
  system:serviceaccount:<namespace>:<service_acc

SERVICE ACCOUNTS

```
apiVersion: v1
kind: ServiceAccount
metadata:
    name: default
    namespace: default
```

ROLE

- L'objet Role est un ensemble de règles permettant de définir quelle opération (ou *verbe*) peut être effectuée et sur quelle ressource
- Le Role ne s'applique qu'à un seul namespace et les ressources liées à ce namespace

ROLE

```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

ROLEBINDING

- L'objet RoleBinding va allouer à un User, ServiceAccount ou un groupe les permissions dans l'objet Role associé
- Un objet RoleBinding doit référencer un Role dans le même namespace.
- L'objet roleRef spécifié dans le RoleBinding est celui qui crée le liaison

ROLEBINDING

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: jane
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

CLUSTERROLE

- L'objet ClusterRole est similaire au Role à la différence qu'il n'est pas limité à un seul namespace
- Il permet d'accéder à des ressources non limitées à un namespace comme les nodes

CLUSTERROLE

```yaml
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

CLUSTERROLEBINDING

```
 kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: salme-reads-all-pods
subjects:
- kind: User
  name: jsalmeron
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

RBAC

```
kubectl auth can-i get pods /
--namespace=default /
--as=spesnova@example.com
```

NETWORKPOLICIES

- La ressource NetworkPolicy est une spécification permettant de définir comment un ensemble de pods communiquent entre eux ou avec d'autres endpoints
- Le NetworkPolicy utilisent les labels pour sélectionner les pods sur lesquels s'appliquent les règles qui définissent le trafic alloué sur les pods sélectionnés
- Le NetworkPolicy est générique et fait partie de l'API Kubernetes. Il est nécessaire que le plugin réseau déployé supporte cette spécification

NETWORKPOLICIES

- DENY tout le trafic sur une application
- LIMIT le trafic sur une application
- DENY le trafic all non alloué dans un namespace
- DENY tout le trafic venant d'autres namespaces
- exemples de Network Policies : https://github.com/ahmetb/kubernetes-network-policy-recipes

NETWORKPOLICIES

- Exemple de NetworkPolicy permettant de blocker le trafic entrant :

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-deny-all
spec:
  podSelector:
    matchLabels:
      app: web
  ingress: []
```

PODSECURITYPOLICIES

- Nécessite d'être activés spécifiquement
- Permet de définir ce qui est alloué pour l'éxecution
- Il faut une PSP par défaut
- A utiliser dans un contexte multitenant
- Peut-être combiné avec le RBAC
- Attention: Activer cette fonctionnalité peut endommager votre environnement

ADMISSION CONTROLLERS

- Interceptes les requêtes sur l'API Kubernetes
- Peut effectuer des modifications si nécessaires
- Conception personnalisée possible

ADMISSION CONTROLLERS

- DenyEscalatingExec
- ImagePolicyWebhook
- NodeRestriction
- PodSecurityPolicy
- SecurityContextDeny
- ServiceAccount

CONCLUSION