301 – Kubernetes

DEVOPS

WIK-DEVOPS-301

Intervenant : Jeremy Trufier <jeremy@wikodit.fr>

`v1.0-beta.0`

wikodit

Prérequis

- WIK-DEVOPS-201 – Docker
- WIK-DEVOPS-202 – Ansible

Introduction

Préparation

A. Avec minikube en local

Minikube

Minikube démarre une VM, donc inutile de le faire à l'intérieur d'une VM !!!

Installation

Pour Mac :

```
brew install minikube kubernetes-cli
```

Pour Linux : cf. Doc selon votre Linux
* Installer kubectl
* Installer minikube

Pour Windows (installation manuelle) :
1. téléchargez depuis le GitHub officiel : minikube-windows-amd64
2. Renommer le en minikube.exe
3. Télécharger kubectl (manuellement)

Lancement

Puis pour démarrer minikube

```
minikube start
```

Possible de spécifier le type de VM au premier démarrage (hyperv a du mal):

```
minikube start --vm-driver=virtualbox
```

Tester :

```
kubectl get pods
==> No resources found in default namespace.
```

B. Dans le cloud

Au préalable

Préparation

- Utilisez l'offre étudiante de Azure (100$), AWS (75$) ou Google Cloud (200$)
- Assurez-vous d'avoir trois machines distinctes qui constitueront votre cluster Kubernetes
- Assurez-vous d'avoir une machine de contrôle (Linux ou Mac) : votre ordinateur, sous-système Linux sur Windows, une VM, etc...
- Important : utiliser les clés privées / clés publiques
- Important : au moins 1.5Gb par machine

N'oubliez pas d'arrêter les machines quand vous ne les utilisez pas entre les cours !

Initialisez les machines avec le Linux CoreOS !!

Au préalable

Préparation

- Utilisez l'offre étudiante de Azure (100$), AWS (75$) ou Google Cloud (200$)
- Assurez-vous d'avoir trois machines distinctes qui constitueront votre cluster Kubernetes
- Assurez-vous d'avoir une machine de contrôle (Linux ou Mac) : votre ordinateur, sous-système Linux sur Windows, une VM, etc...
- Important : utiliser les clés privées / clés publiques
- Important : au moins 1.5Gb par machine

N'oubliez pas d'arrêter les machines quand vous ne les utilisez pas entre les cours !

Initialisez les machines avec le Linux CoreOS !!

Projet Ops

- TD: Créez un projet ops, ex: k8s-ops
- Suivez le cours WIK-DEVOPS-202 – Ansible pour avoir la structure
- Les 3 machines doivent être joignable par Ansible avec ansible all -a "echo OK"

  SI UNE ERRREUR INDIQUANT QUE PYTHON N'EST PAS PRESENT SUR LES MACHINES DISTANTES, TOUT VA BIEN (python n'est pas présent par défaut sur coreOS)

Kubespray 1/4

Ensemble de scripts Ansible permettant d'installer, de scaler, de mettre à jour un cluster Kubernetes

1. Créez un dossier vendor, dans lequel nous initialiserons Kubespray

Pour cela, nous allons utiliser les sous-modules Git, ce qui nous permettra d'inclure Kubespray dans notre projet, et de pouvoir bénéficier de leur mises à jour futures.

2. git submodule add https://github.com/kubernetes-sigs/kubespray.git vendor/kubespray
Cette commande, clone le Git de Kubespray dans le dossier vendor. Vous pouvez checkout une version spécifique (il nous faut > v1.13)

3. Important : copiez les settings de vendor/kubespray/ansible.cfg dans le ansible.cfg de votre projet. Puis modifiez la variable roles_path pour inclure "vendor/kubespray/roles", et modifier également la variable library vers "vendor/kubespray/library"

4. Copiez le contenu de vendor/kubespray/inventory/sample/group_vars dans votre group_vars et completez votre fichier hosts.ini (ou inventory.ini) de façon à avoir un fichier similaire à vendor/kubespray/inventory/sample/hosts.ini
Vos trois machines doivent être dans les trois groupes [etcd], [kube-master] et [kube-node].

5. Spécifiez ansible_host dans le inventory.ini ou dans les host_vars (pas besoin de spécifier la variable ip)

Kubespray 2/4

Exécutez : pip install -r vendor/kubespray/requirements.txt

Éditez les configurations dans group_vars :

k8s-cluster/
- k8s-cluser.yml : utilisez au moins la version de kubernetes v1.13.2

all/
- all.yml changer bin_dir avec /opt/bin
- all.yml rajouter : ansible_python_interpreter = /opt/bin/python

addons/
- Activez le dashboard
- Activez HELM

Ajouter dans ansible.cfg :

```
[privilege_escalation]
become          = True
become_method = sudo
become_user    = root
```

Kubespray 3/4

Rajoutez les variables suivantes
dans le dossier de variable all

```
kube_controller_cpu_requests: 20m
kube_scheduler_cpu_requests: 20m
kube_apiserver_cpu_requests: 20m
dashboard_cpu_requests: 20m
dns_cpu_requests: 20m
calico_node_cpu_requests: 20m
calicoctl_cpu_requests: 20m
```

En effet, l'addition des cpu_requests par défaut équivaut à environ 1000m, soit 1 CPU entier, sauf que le système a besoin de réserver 200m. Avec 1 CPU par noeud, il n'est donc pas possible de faire tourner Kubespray, car aucun nouveau service n'aura de CPU disponible pour se lancer.

Cette manipulation permet de ne "réserver" que 140m soit 0.14 CPU, ce qui est largement suffisant, surtout que c'est le "minimum" alloué à un service.

Si vous avez des machines disposant de 2 CPU, cette manipulation n'est pas obligatoire.

Kubespray 4/4

5. Vérifiez que chacune de vos machine possède la variables : ansible_host (facultatif : variable ip qui est l'ip privée)

6. Créez le play suivant (plays/k8s-setup.yml) :

7. Lancez le play !
ansible-playbook plays/k8s-setup.yml

Pour allez plus loin, vous pouvez utilisez le play pour configurer un réseau interne aux machines, et les faire communiquer ensemble par ce réseau.

```yaml
- hosts: k8s-cluster:etcd:calico-rr
  gather_facts: false
  vars:
    ansible_ssh_pipelining: false
  roles:
    - { role: kubespray-defaults}
    - { role: bootstrap-os, tags: bootstrap-os}

- hosts: k8s-cluster:etcd
  tasks:
    - name: "Stop auto-reboot"
      service:
        name: locksmithd
        state: stopped

- name: Setup kubernetes cluster
  import_playbook: ../vendor/kubespray/cluster.yml

- hosts: k8s-cluster:etcd
  tasks:
    - name: "Restore auto-reboot"
      service:
        name: locksmithd
        state: started
```

Locksmith est le process qui redémarre coreOS après une Maj, le désactiver pendant l'installation permet d'éviter des soucis...

Vérifier

```
ssh user@master-ip
sudo -i
kubectl get nodes
```

Authentification

- Il faut tout d'abord récupérer les jetons d'authentification administrateur depuis un node master

```
mkdir -p ~/.kube
ssh core@ip-of-master-node sudo cat /etc/kubernetes/admin.conf > ~/.kube/config
```

- Finalement, éditez ce fichier (~/.kube/config)

- C'est le fichier qui sera utilisé pour communiquer avec l'API kubernetes, sa localisation peut être changé grâce à la variable d'environnement KUBE_CONFIG

Le fichier Kube Config

- Ce fichier YAML permet de gérer différents clusters avec différent users, il permet de stocker :
  - Des users (avec mot de passe, jeton oauth/oidc, ou avec certificats)
  - Des clusters (certificat de validation + IP/adresse d'un master ou d'un LB)
  - Des contextes : un contexte est ce qui permet de se connecter à un namespace précis, d'un cluster précis, avec un utilisateur précis

```yaml
apiVersion: v1
kind: Config
current-context: null
clusters: []
contexts: []
users: []
```

<= Le context utilisé par défaut

Le fichier Kube Config

- Chaque user, context ou cluster du fichier possède un name qui permet de l'identifier au sein du fichier, ce name est totalement libre
- Le fichier ci-contre possède deux contextes, dont un qui permet d'administrer le cluster, et l'autre qui permet de communiquer avec le namespace de production
- Pour info le namespace par défaut si non spécifié est "default"

```yaml
apiVersion: v1
kind: Config

clusters:
  - name: mpd
    cluster:
      certificate-authority-data: L...LQo=
      server: https://ip-du-node-master-de-mon-projet-devops:6443

users:
  - name: mpd-admin
    user:
      client-certificate-data: L...0tLQo=
      client-key-data: LS0...Qo=

  - name: jeremy@wikodit.fr
    user:
      auth-provider:
        name: oidc
        config:
          client-id: 1...m
          client-secret: s...5
          id-token: eyJ...rg8g
          idp-issuer-url: https://accounts.google.com
          refresh-token: 1...c

contexts:
  - name: mpd-admin
    context:
      cluster: mpd
      user: mpd-admin

  - name: mpd-production
    context:
      cluster: mpd
      user: jeremy@wikodit.fr
      namespace: production

current-context: mpd-production
```

Utiliser le client

- Installer kubectl (le package s'appelle kubernetes-cli ou kubectl)

```
kubectl get nodes
```

Magique !

Utiliser un contexte différent

```
kubectl --context=mpd-production get pods
```

Utiliser un namespace différent

```
kubectl -n kube-system get pods
```

Un namespace permet de segmenter le cluster
Un pod est un groupe de 1 ou plusieurs containers
kube-system est le namespace contenant les composants essentiels du cluster

Utiliser le client

La commande kubectl

kubectl -n kube-system action resource nom-de-resource

```
get <ressource> - Liste les entités
describe <ressource> <ressource name> - Affiche plus d'info sur une entité
create / delete / edit / patch
logs <pod name> [-c <container-name>] - Affiche les logs d'un container
exec <pod name> <options> - Execute une commande dans un container
proxy - Permet d'accéder à tous les services comme si en local
port-forward <options> - Permet de créer un tunnel vers un service
...
```

Ces actions seront détaillées plus tard

L'API Kubernetes

Les ressources et kind

- Les ressources sont des endpoints de l'API Kubernetes, ils permettent d'accéder aux Objets Kubernetes du type (kind) correspondant à la ressource, la resource `pod` ou `pods` permet de lister les objets kubernetes de type `pod`

- Il est possible d'ajouter des resources personnalisées, afin de déléguer des configurations à des scripts et controlleurs

- La référence de l'API se trouve ici : https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.13/

Les fichiers de configuration

- Avec kubernetes, il est possible de créer des resources à la main en utilisant "kubectl create ..." avec une flopée d'arguments derrière
- Il est également possible d'utiliser des fichiers de configuration YAML

mon-premier-pod.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: hello-world
  labels:
    name: hello-world
spec:
  containers:
    - name: hello-world
      image: dockercloud/hello-world
      ports:
        - containerPort: 80
```

1. appliquer la configuration :

```
kubectl apply -f mon-premier-pod.yml
```

2. créer un tunnel vers le container :

```
kubectl port-forward pod/hello-world 8080:80
```

3. Accéder à http://localhost:8080

Info: lorsqu'un fichier est modifié, puis appliqué, Kubernetes mettra automatiquement l'objet à jour en fonction de son nom (ou le créera s'il n'existe pas)

Les fichiers de configuration

L'API kubernetes à utiliser, vous trouverez également des versions betas etc...

Le type d'Objet, nous verrons les différents types en détails

Les métadonnées, qui permettent d'identifier l'objet

Le nom de l'objet, doit être unique pour un même type de ressource au sein d'un namespace

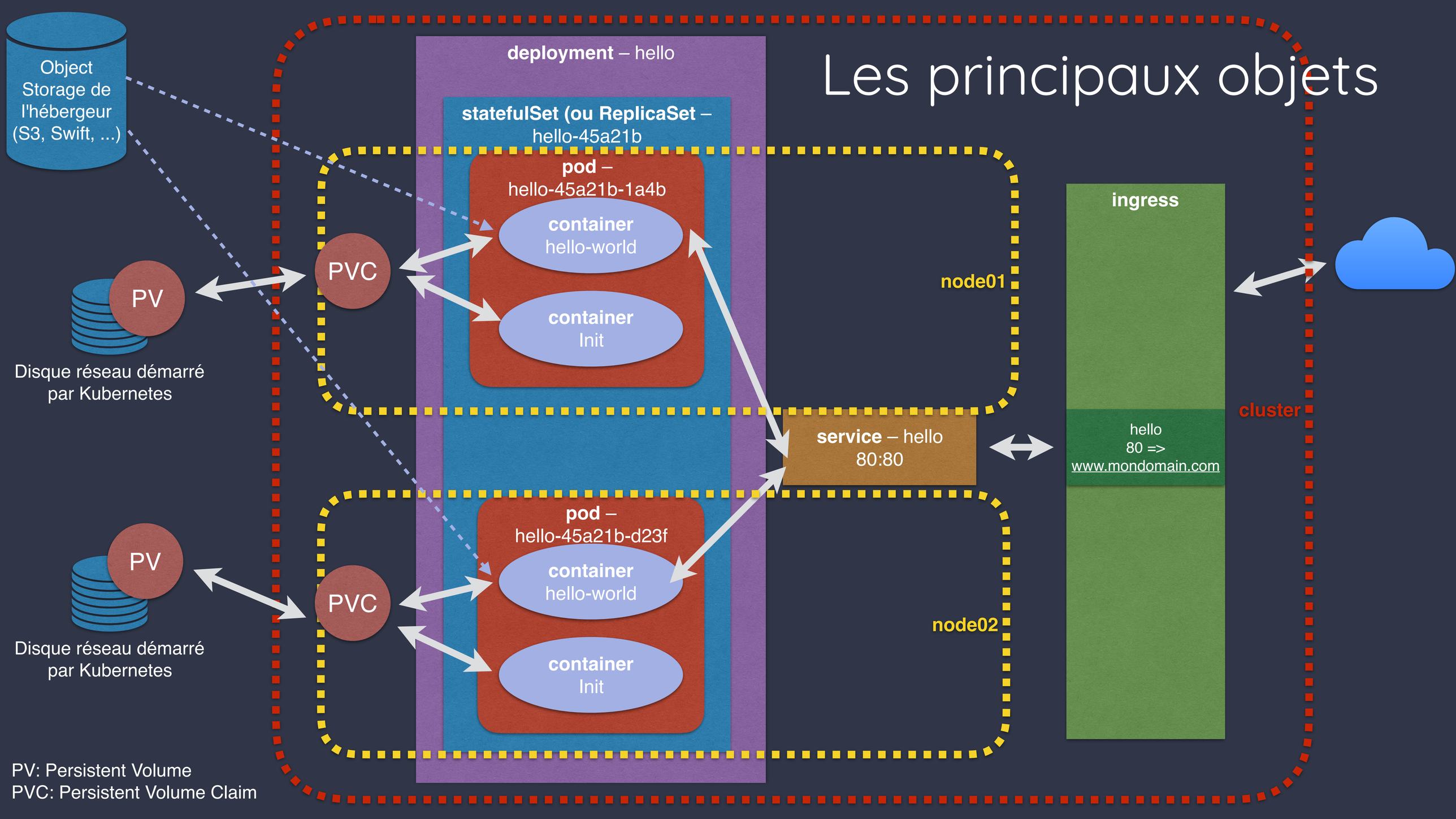
Les labels permettent d'organiser les ressources, ils sont totalement libres

Les annotations permettent de stocker des informations dynamique ou de passer des options de configurations pour d'autres objets qui utiliseront cet objet

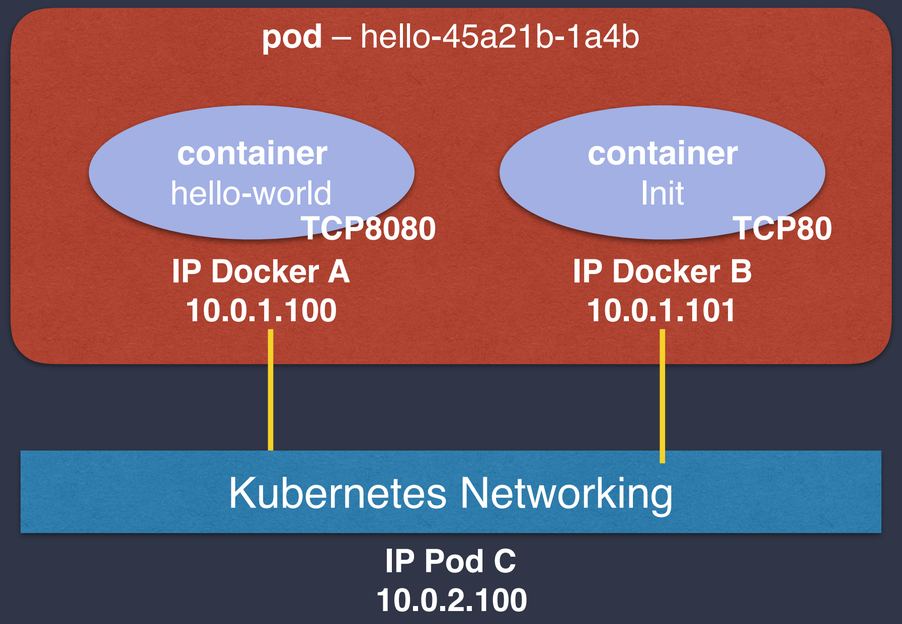
La spécification de l'objet, ici dans le cas du Pod, on y retrouvera la liste des containers

```yaml
apiVersion: v1

kind: Pod

metadata:

  name: hello-world


  labels:
    environment: production
    tier: frontend

 annotations:
    some-service/config: value


spec:
  containers:
    - name: hello-world
      image: dockercloud/hello-world
      ports:
        - containerPort: 80
```

Les principaux objets



Pod



- Unité de base Kubernetes
- Le master choisit sur quel machine lancer un Pod
- Un Pod contient un ou plusieurs container
- Un Pod possède une adresse IP qui expose les ports exposés par les containers
- (Potentiellement des containers d'initialisation ou de terminaison)
- Un pod est généralement démarré par un autre objet kubernetes

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: hello-world
  labels:
    name: hello-world
spec:
  containers:
    - name: hello-world
      image: dockercloud/hello-world
      ports:
        - containerPort: 80
      env:
        - name: TEST
          value: toto
      resources:
        limits:
          cpu: 100m
          memory: 128Mi
        requests:
          cpu: 100m
          memory: 128Mi
```

Deployment

- Le déploiement garantit le démarrage et l'organisation des pods par l'intermédiaire de ReplicaSet
- Il démarre généralement un ReplicaSet
- Si le ReplicaSet ou les Pods sont supprimés, le déploiement les recréera
- Un déploiement spécifie également la stratégie lors de la mise à jour
- Ainsi pour mettre à jour une app, il suffira simplement de patcher le déploiement avec la nouvelle image docker, le déploiement se chargera du reste

```yaml
kind: Deployment
metadata:
  name: hello-world
  labels:
    app: hello-world
    tier: devops
spec:
  selector:
    matchLabels:
      app: hello-world
  replicas: 3
  template:
    metadata:
      labels:
        app: hello-world
    spec:
      containers:
        - name: hello-world
          image: dockercloud/hello-world
          ports:
            - containerPort: 80
          resources:
            limits:
              cpu: 100m
              memory: 128Mi
            requests:
              cpu: 100m
              memory: 128Mi
```

Service

- Un service fournit une IP virtuelle (kubernetes networking)
- Un service s'occupe de LoadBalancer différents Pod

```yaml
kind: Service
apiVersion: v1
metadata:
  name: hello-world
spec:
  selector:
    app: hello-world
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

Ingress

config/helm-ingress-nginx.yml

```yaml
controller:
  kind: DaemonSet
  hostNetwork: true
  daemonset:
    useHostPort: true
  service:
    type: ClusterIP
```

Pour mettre en place un Ingress sur votre cluster exécutez :

```
helm repo add stable https://kubernetes-charts.storage.googleapis.com/
helm install ingress -f config/helm-ingress-nginx.yml stable/nginx-ingress
```

- Un ingress connecte un service à Internet
- Un ingress peut effectuer la terminaison SSL
- Un ingress permet de gérer les virtualHost pour rediriger vers un service en fonction d'un nom de domaine
- Différents Ingress existent : nginx, haproxy, etc...

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: hello-world
spec:
  rules:
    - host: hello.testdevops.tk
      http:
        paths:
          - path: /
            backend:
              serviceName: hello-world
              servicePort: 80
          - path: /lol
            backend:
              serviceName: service-lol
              servicePort: 8080
```

Le host est facultatif

ConfigMap

- Un ConfigMap peut stocker des fichiers qui peuvent être montés à des emplacements
- Avec un ConfigMap on peut aussi stocker des variables d'environnements, exemple :

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: hello-world-env
data:
  NODE_ENV: production
  PORT: 3300
  TEST: toto
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: hello-world
  labels:
    name: hello-world
spec:
  containers:
    - name: hello-world
      image: dockercloud/hello-world
      ports:
        - containerPort: 3300
      envFrom:
        - configMapRef:
            name: hello-world-env
      resources: {...}
```

Secret

- Un Secret est l'équivalent d'un ConfigMap, à la différence qu'il est utilisé pour stocker des données sensible
- Les données doivent être encodées en Base64
- à ce jour les Secret n'offrent pas plus de sécurité que le ConfigMap (à part au niveau des ACLs)

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: hello-world-env
data:
  NODE_ENV: cHJvZHVjdGlvbg==
  PORT: MzMwMA==
  TEST: dG90bw==
```

```
↳ echo -n 'production' | base64
cHJvZHVjdGlvbg==
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: hello-world
  labels:
    name: hello-world
spec:
  containers:
    - name: hello-world
      image: dockercloud/hello-world
      ports:
        - containerPort: 3300
      envFrom:
        - secretRef:
            name: hello-world-env
      resources: {...}
```

TD

- Créez un namespace avec un fichier de conf, assignez lui des quotas
- Mettre en place le dockercloud/hello-world sur le namespace, sur un nom de domaine ou un chemin
- (Cloud) Mettre en place un LoadBalancer IP sur votre hébergeur Cloud qui redirige vers toutes vos machines. En utilisant cette IP vous devriez pouvoir afficher le hello-world
- (Cloud) Éteindre une des machines, et vérifier que tout fonctionne toujours.

Templating et HELM

TP

Félicitations !!

Cours WIK-DEVOPS-301 burned :)