BUT: R5.D10 (bases de Kubernetes)

Jean-Marc Pouchoulon

Octobre 2025

Les containers se sont révélés comme des entités informatiques nativement adaptées aux applications CLOUD. Très proches de la vision applicative du système d'information il sont devenus rapidement des "first class citizen" des infrastructures actuelles.

En tant que tels ils leur faut un cadre de fonctionnement solide et adaptable, piloté par un chef d'orchestre : Kubernetes. Issu de Google, Kubernetes est devenu rapidement la référence en termes d'orchestration de containers.

L'API de Kubernetes est identique quelque soit les infrastructures sous-jacentes. C'est un atout important pour délivrer du code évoluant pour des raisons de sécurité informatique ou d'amélioration continue. C'est aussi la promesse d'un couplage faible entre les infrastructures et le code permettant une organisation claire entre Dev && Ops.

Kubernetes ou "k8s" [1] évolue rapidement et offre des solutions pour piloter des containers mais aussi des machines virtuelles. [2]

1 Compétences à acquérir lors du TP.

Compétences pré-requises:

— Enseignements précédents en particulier sur les containers applicatifs : Construire un container avec un Dockerfile , utiliser un registry, les commandes Docker...

Compétences principales:

L'objectif de ce TP est d'orchestrer des containers pour améliorer la disponibilité et la scalabilité des applications. La haute disponibilité du "control plane" de k8s ne sera pas traitée ici.
Les compétences visées:

— travailler avec une configuration Kubernetes sur son propre poste;
— utiliser la commande "kubectl";
— transformer les commandes "imperative" en "declarative";
— utiliser Docker et Kubernetes;

1. il est appelé ainsi car entre le K et le S de Kubernetes il y a 8 lettres
2. voir par exemple `https://www.vmware.com/fr/products/vsphere/projectpacific.html` et `https://www.weave.works/blog/firekube-fast-and-secure-kubernetes-clusters-using-weave-ignite`

— créer et gérer un "pod";
— créer et gérer un "pod init";
— limiter la consommation de ressources d'un pod;
— utiliser un label;
— utiliser les concepts de "taint & toleration";
— mettre un noeud k8s en maintenance et le remettre en production;
— générer une définition en "Yaml" d'un objet k8s à partir d'un "dry-run";
— "kubernétiser" une image Docker;
— rendre disponible son application en dehors du Cluster (services dont "NodePort" et "LoadBalancer", "ingress controller", "Bare metal loadbalancer","Ingress Controller");
— gérer le cycle de vie d'une application dans Kubernetes;
— utiliser le manager d'applications "Helm" for "fun & profit";
— stocker des données de façon permanente dans un cluster;

Savoir:

— comprendre la philosophie de Kubernetes;
— savoir énoncer les rôles du "Control plane" et du "Data Plane" dans Kubernetes;
— savoir décrire et énoncer les fonctions de quelques objets essentiels de Kubernetes ("NameSpaces" ,"pods" , "deployments", "services", "replicatsets"...);

Vous travaillerez individuellement pour ce workshop.
Vous changerez le nom de votre hôte afin de faire apparaître votre nom via la commande suivante suivi d'un reboot:

```
hostnamectl set-hostname votrenom-vm
```

Sur le compte-rendu figureront les numéros de questions, les réponses mais aussi les résultats de vos commandes montrant la réussite de vos actions. Le prompt modifié ci-dessus apparaîtra sur vos "output" N'oubliez pas votre nom sur vos compte-rendus.

2 "Crash course" sur Kubernetes

2.1 Approches globales de Kubernetes

Kubernetes peut être vu sous plusieurs angles:

— comme un système d'exploitation réparti sur plus nœuds ou hôtes du réseau. Il va piloter des ressources ("containers" ou "virtual machine").
— comme un pilote [1] sous forme d'une "boucle informatique" qui sans cesse vérifie qu'il respecte les ordres de son amiral. On peut voir les hôtes machines physiques ou virtuelles qui participent au cluster Kubernetes comme des vaisseaux (participant au *"data plane"* avec d'autres navires) dirigés par une cellule de commandement (le *"control plane"*) et remplissant des tâches en fonction des ordres données par ce dernier. A l'intérieur de chaque vaisseau les groupes de processus sous forme de PODS (un ensemble d'un ou plusieurs containers) sont des marins qui partagent un même but logiciel (un micro-service utile à l'ensemble de la flotte) et des ressources (Network , IPC, voire process "NameSpace").

3 Installation du cluster Kubernetes avec Kind

Il existe de nombreuse façon de travailler avec Kubernetes sur son poste de travail. On peut travailler en local avec le client "kubectl" et se connecter à un cluster de développement distant. On peut aussi installer des noeuds Kubernetes sur son laptop sous forme de machine virtuelle ou de containers... Docker.

On citera:

1. En grec ancien Kubernetes signifie le pilote. Le mot Cybernétique vient de là. C'était la minute culturelle de ce workshop

— minikube;
— micro-k8s (Ubuntu);
— Docker Desktop;
— k3d;
— kind (Kubernetes in Docker);
— ...

Dans ce workshop nous allons utiliser "kind". Les noeuds du cluster Kubernetes sont des containers Docker et le "plugin" réseaux de Kind a été bâtie pour être compatible avec Docker.

— installer kubectl sur votre machine [1];
— installez en suivant la documentation issue de `https://github.com/kubernetes-sigs/kind`;
— créer votre cluster kind en clonant le repository
`https://github.com/pushou/substrat-labs-k8s.git`
et lancez le script create-kind-cluster-with-ec.sh;

Vous pouvez visualisez les nodes Kubernetes qui sont ici des containers Docker via la classique commande

```
docker ps
```

et vous rendre sur le noeud Kubernetes via

```
docker exec -it id-container-noeud bash
```

Les pages suivantes:

— `https://kubernetes.io/docs/tasks/`;
— `https://kubernetes.io/fr/docs/reference/kubectl/cheatsheet/`;
— `https://kubernetes.io/docs/tasks/debug-application-cluster/crictl/`;

vous seront d'une grande aide. La commande "k explain" suivi du sujet peut aussi vous aider:

```
k explain pod # afficher l'essentiel
k explain pod.apiVersion # détailler un sujet
k explain pod.apiVersion --recursive # lister tous les champs
```

Il est recommandé d'installer fzf afin de faciliter l'accès à l'historique de commandes:

```
~fzf/install
```

Le client kubernetes kubectl permet d'interagir avec "l'API server". Le format de communication des échanges est le format "json". Ce client est installé sur le poste de l'administrateur et se sert d'un fichier "config" situé dans /.kube qui contiendra l'IP et le port de "l'API server".

kubectl supporte plusieurs formats de sortie dont les formats "json" et "yaml". Vous installerez jq via apt et jid [2] pour traiter du json. Passez la commande suivante afin de récupérer les capacités de votre cluster:

```
kubectl get nodes -o json |
jq ".items[] | {name:.metadata.name} + .status.capacity"
```

Créer cet alias essentiel dans votre .bashrc et activer la complétion pour kubectl:

1. `https://kubernetes.io/fr/docs/tasks/tools/install-kubectl/`
2. `https://github.com/fiatjaf/jiq`

```
alias k=kubectl
echo 'source <(kubectl completion bash)' >>~/.bashrc
kubectl completion bash >/etc/bash_completion.d/kubectl
echo 'alias k=kubectl' >>~/.bashrc
echo 'complete -F __start_kubectl k' >>~/.bashrc
source ~/.bashrc
```

kind permet de charger image depuis un registry pour l'ensemble des noeuds du cluster. Par exemple la commande suivante permet de charger l'image registry.iutbeziers.fr/pythonapp:latest.

```
# Chargement de l'image en local
docker pull registry.iutbeziers.fr/pythonapp:latest
# upload de l'image sur les containers Docker
kind --name tp1k8s load docker-image registry.iutbeziers.fr/pythonapp:latest
```

Vous pourrez utiliser cette fonctionnalité si le registry est privé.

4 Premiers pas avec Kubernetes

4.1 Configuration des objets en mode déclaratif et impératif

Quand on utilise [1] un mode déclaratif on déclare dans un "manifest yaml" les éléments de configurations. Ces éléments décrivent un état souhaité que l'on peut faire évoluer. [2] En "mode impératif" on donne des ordres via la ligne de commande.Ce mode ne garde pas la mémoire des configurations.

1. Adaptez à votre contexte et passez les commandes suivantes pour vous familiariser avec Kubernetes et son mode impératif:

   ```
   kubectl run nginx-pod --image nginx
   kubectl exec -it nginx-pod -- sh
   kubectl create deployment hello-nginx --image nginx
   kubectl scale deployment hello-nginx --replicas 2
   kubectl expose deployment hello-nginx --type=LoadBalancer --port 80 --target-port 80
   ```

2. Visualisez les objets Kubernetes générés avec les commandes suivantes:

   ```
   # quelques commandes à tester
   k get nodes -o wide --show-labels
   k describe "ressources=node,pods, deploy,service..."
   k logs "pods"
   ```

   Ce mode est direct mais limité dans ses options. Il n'est pas orienté DevOps et "infrastructure as code". On peut lui préférer le mode "déclaratif" qui passe par l'édition d'un fichier yaml. Ces fichiers, cartes perforées des temps modernes sont envoyés à l'API de Kubernetes qui va les comprendre et s'efforcer de mettre en œuvre l'état demandé.

   Il existe aussi un mode impératif avec fichiers de commandes. Ce mode permet d'avoir de la persistence mais il n'est pas idempotent [3]. Deux personnes peuvent appliquées des changements et écraser les modifications de l'autre. Les ordres suivants sont caractéristiques de ce mode. (config objet=une configuration yaml):

1. voir aussi un article intéressant sur `https://atix.de/en/gitops-kubernetes-the-easy-way-part1/`
2. config objet=un objet K8S
3. https://fr.wikipedia.org/wiki/Idempotence

```
# Ne passez pas ces commandes c'est juste une information sur le mode impératif qui ne fonctionnera que quand
# vous aurez un fichier yaml valide (voir question suivante)
kubectl create -f "config-objet.yaml"
kubectl replace -f "config-objet.yaml"
kubectl delete -f "config-objet.yaml"
```

3. Utilisez la commande 'kubectl create –dry-run=client -o yaml "objet"' afin de générer les manifests des objets créés en mode impératif précédemment.
4. Créer via la commande Kubectl et l'option "–from-literal" une "configmap" nommée maconfigmap avec les couples clefs/valeurs suivants:
   — k8s=leprésent
   — virt=legacy

   Affichez ensuite les valeurs.
5. Créer de même un secret nommé monsecret avec la valeur mdp=torototo et affichez le "décodé" (il est en base 64).

4.2 Les "PODS" l'unité atomique de Kubernetes

Le pod est le plus petit objet déployable d'un cluster Kubernetes. Il est éphémère ("cattle" et pas "pet" !). Il est bâti à partir des namespaces, des cgroups et des capabilities.

Il est constitué par un ou plusieurs containers partageant les mêmes NameSpaces réseaux et IPC [1]. Les containers d'un POD sont aussi capables de partager entre eux des données au travers d'un partage sur l'hôte sur lequel ils s'exécutent. On distingue communément 3 patterns possibles [2]:

— Le plus connu est le pattern "side-car". Un container en "side-car" améliore ou étend les fonctionnalités d'un autre container. Par exemple les autres containers utilisent un partage recueillant les "logs" et qui va être traité par le container en "side-car".
— Le pattern "ambassadeur" va "proxyfier" les communications pour et à destination du POD. On peut imaginer un programme qui s'adresse à une IP et un port du POD (les containers dans un POD partage la même couche réseau) et qui en fait se connecte à un container jouant le role de proxy vers d'autres POD. Il est communément implémenté en "side-car". On peut lui déléguer les terminaisons SSL, l'authentification vers le micro-service du container...
— Le pattern "adapter" qui lui va normaliser le flux en sortie d'un POD par exemple en modifiant le format des logs exportées par un serveur web.

Un pod peut servir à initialiser un autre pod (récupération d'un fichier et partage).

Un pod est dit "statique" quand il est piloté par le daemon Kubelet de l'hôte. On ne peut pas rajouter dynamiquement un container dans un "pod" existant ce qui est parfois limitant. [3]

Les Pods sont constitués de containers qui vont être mis en œuvre au travers d'un "Container Runtime" qui implémentent tous une interface(C.R.I.). "dockerd" n'est plus le runtime de Kubernetes, les acteurs du marché l'ayant trouvé trop général pour être efficient. Son sous-ensemble: containerd est par contre largement utilisé avec celui de RedHat : CRI-O.

En conséquence l'interface de commande de Docker ne peut plus être utilisé sur chaque noeud Kubernetes. La commmande "ctr" permet d'interagir avec les pods sur les nœuds K8S mais il y en a d'autres comme crictl et nerdctl.

1. Le "process namespace du container" peut aussi être partagé voir `https://kubernetes.io/docs/tasks/configure-pod-container/share-process-namespace/#understanding-process-namespace-sharing`
2. Voir Design patterns for container-based distributed systems de Brendan Burns et David Oppenheimer (Google)
3. En fait c'est possible au travers d'un container éphémère dans les dernières versions de Kubernetes voir `https://github.com/kubernetes-sigs/kind/issues/1210` pour la mise en œuvre en version alpha

4.2.1 Opérations basiques sur les PODS

1. Générez la configuration d'un pod debian à l'aide de la commande suivante:

```
k run --dry-run=client  debianpod --image=registry.iutbeziers.fr/debianiut:latest -o yaml > monpremierpod.yml
```

   Créez ce manifest sur votre cluster afin de créer votre premier pod avec "kubectl create".
2. Vérifiez l'état de votre pod ?
3. Sur quel nœud votre pod s'exécute-t-il ?
4. Dans quel NameSpace votre Pod s'exécute-t-il ?
5. Accédez au container en utilisant "kubectl exec". Démarrez un server apache dans le container.
6. Pouvez-vous accéder au serveur web Apache qui s'exécute dans ce Pod ? Accéder au serveur Apache depuis votre client Kubernetes en utilisant "kubectl port-forward". Quel est le principe de cette commande ?

4.2.2 Manipulation des Pods

1. Supprimez le pod et recréez-le cette fois-ci en utilisant un "kubectl apply -f votre-manifeste". Quelle est la différence entre "apply" et "create" ?
2. Recréer le POD avec en sus un container issu d'une image busybox. Rattachez-vous au pod via "kubectl exec" en précisant avec l'option -c le container nom donné au container busybox.
3. Modifiez le manifeste du pod afin de limiter le cpu et la mémoire consommer par ce pod via des "limits".
4. Générer un autre pod sur un autre "node" K8s et effectuez un traceroute entre les deux pods. Quelles différences constatez-vous avec la gestion réseaux des containers Docker ?

4.2.3 Utilisation des utilitaires des "Container runtime" et de kubectl pour manipuler des containers

1. Lancez un process bash dans le pod créé avec kubectl.
2. Lister les images des containers utilisées par votre cluster Kubernetes à l'aide de kubectl [1].
3. Utilisez les utilitaires nerdctl [2] et crictl [3] pour récupérer la liste des images sur le noeud "worker" qui exécute le container. [1]
4. Lancez un process bash dans le pod avec crictl [4]. Faite de même avec l'utilitaire nerdctl.
5. Pour information il est possible de lancer un processus dans un container à l'aide de ctr l'utilitaire standard de containerd

   Les étapes sont les suivantes:

```
# On retrouve l'ID du container (attention contrairement à la commande docker on ne peut pas le tronquer pour sélectionner un container)
ctr   --namespace=k8s.io  c ls
# On lance un shell dans notre container avec l'ID non tronqué du container debian
ctr  --namespace=k8s.io  task exec --exec-id 1223  247248398b849c2e0c172d060827a77f7bbeef01ccabc502236a5b93407ec625 \
/bin/bash -c 'ps -ef'
```

   Pour nerdctl vous utiliserez aussi le namespace k8s.io

4.2.4 Implémentation d'un pattern commun: l'init pod

1. installation d'un init pod Inspirez-vous de :

   `https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-initialization/`
   afin de créer un init pod qui récupère un fichier html sur un autre serveur web pour l'afficher.

1. voir `https://kubernetes.io/fr/docs/tasks/access-application-cluster/list-all-running-container-images/`
2. `https://github.com/containerd/nerdctl`
3. Passez en root pour lancer cette commande
1. Il faut être root sur le noeud et utiliser l'option –namespace=k8s.io
4. voir `https://asciinema.org/a/179047` pour son usage

Vous utiliserez l'image apache [1] standard [2].

5 Gestion du cycle de vie des applications d ans Kubernetes

Le POD est un objet Kubernetes qui a des limitations:

— il ne permet pas le passage à l'échelle. La scalabilité va être assurée par l'objet "Deployment".
— Le POD est éphémère et il n'est donc pas d'adresse IP persistente sur le réseau. L'objet "service" créé une adresse IP persistante pour les PODS et leur "deployment".

5.1 Déploiement de son application Kubernetes

5.1.1 Les "NameSpaces"

Les "NamesSpaces" sont une brique essentielle de la construction des containers et donc aussi pour les "pods".Il est fortement recommandé de s'en servir pour isoler les applicatifs entre eux afin de regrouper les "pods" créant des micro-service pour une même application. Ils permettent aussi de séparer ce qui relève de l'infrastructure des containers ("namespace kube-system") de ce qui relève de l'applicatif.

1. Créez un namespace "applicatif".
2. Editez le "manifest" de ce namespace avec kubectl et modifier son nom en "applicatifs".
3. Listez l'ensemble des namespaces de ce cluster kubernetes puis l'ensemble des objets kubernetes de votre cluster.
5. A quoi sert le namespace kube-system ? le name-space default ?

5.2 Déployer une application Python dans Kubernetes

5.2.1 "Deployment" de l'application

On va ré-utiliser l'application Python générée avec le TP Docker. L'U.R.I. /env permet de récupérer les variables d'environnement du contrôleur. Vous utiliserez la commande kubectl.

1. Créez un "deployment" avec "kubectl create" utilisant l'image registry.iutbeziers.fr/pythonapp:latest dans le namespace applicatifs.
2. Dumper la configuration de ce "deployment" dans un fichier yaml. Editez ce fichier et expliquez les différents objets qui le compose.
3. En modifiant le "deployment" passer à trois "pods" Python.

5.2.2 Utilisation des labels

1. Affichez les pods dont le label est "app=pythonapp". Rajoutez un label "env=dev". Vérifiez la prise en compte de votre action. Enlevez le label app. Détruisez les pods labellisés "app=pythonapp". Supprimez au final le déploiement.
2. Labellisez deux nodes "statusnode=maintenance". Drainer les nodes en maintenance en utilisant le label. [3]
3. En utilisant l'adresse obtenue lors de l'affichage du POD essayer d'accédez à l'application en CLI via l'utilitaire curl en dehors du cluster? que constatez-vous ? essayez depuis un noeud du cluster.
4. Modifiez le déployment pythonapp afin que le pod s'execute sur node3.
5. Taintez le node "master" afin que ne soit plus schedulé à l'avenir de pods dessus.
6. Utilisez les fonctionnalités de taint & toleration pour qu'un pod soit schedulé sur node1 - (Taintez différemment tout vos noeuds et lancez un pod avec une toleration pour node1).Expliquez quel est le principe de "taint & toleration".
   Pour lancer le pod:

1. `https://hub.docker.com/_/httpd`
2. afin de simplifier l'utilisation de wget utilisez un site en clair comme `http://store.iutbeziers.fr`
3. utilisez –force pour forcer le kill des pods et –ignore-daemonsets.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-1
  labels:
    security: s1
spec:
 containers:
 - image: registry.iutbeziers.fr/debianiut:latest
   name: debianiut
   command: [ "/bin/bash", "-c", "--" ]
   args: [ "while true; do sleep infinity; done;" ]
 tolerations:
 - key: "node"
    operator: "Equal"
    value: "1"
    effect: "NoSchedule"
```

5.3 Réaliser une migration de version de l'image Python

Créez un nouvelle image Docker de l'application Python (en générant un nouveau tag pour aller vite) ou en modifiant l'application que vous pouvez récupérer sur: Vous pouvez charger l'image avec "kind load" sur l'ensemble des noeuds. Vous devrez indiquer comme "pull policy" de l'image "ifnotpresent" [1]

```
git clone https://github.com/pushou/Python3K8s.git
```

1. Migrez vers cette version en déployant cette nouvelle version.
2. Suivez cette migration à l'aide de la commande rollout.

6 Services

Un "deployment" permet de scaler une application mais ne rend pas l'application accessible même dans le cluster.

C'est la que la notion de **service** intervient: on expose les "deployment" pour les rendre accessibles.

Un service va fournir une adresse IP et un port, un enregistrement DNS et une liste de "endpoint" qui contiennent les IP des "pod". Ces pods sont sélectionnés grâce à un label déclaré dans le service.

On distingue 4 types de services:

— *ClusterIP* (type de service par défaut) utilisé pour communiquer en interne du cluster. (par exemple un service LDAP pour l'authentification).

— *NodePort*: il permet d'exposer le service sur un port aléatoire choisi entre 30000 et 32767 sur *chaque noeud* du cluster.(un port choisi pour tous les noeuds) Ce port sera joignable sur tous les noeuds du Cluster qu'il héberge un Pod éligible ou non.

Les nœuds d'un cluster K8S dans le CLOUD n'ont pas toujours une IP publique: il faut un équilibreur de charge classique en frontal ou utiliser un VPN pour s'y connecter.

— *LoadBalancer*: Les Services de type LoadBalancer sont utilisés quand le cluster est hébergé chez un « Cloud Provider ».

1. NOTE: The Kubernetes default pull policy is IfNotPresent unless the image tag is :latest or omitted (and implicitly :latest) in which case the default policy is Always. IfNotPresent causes the Kubelet to skip pulling an image if it already exists. If you want those images loaded into node to work as expected, please: don't use a :latest tag and / or: specify imagePullPolicy: IfNotPresent or imagePullPolicy: Never on your container(s).

Le cluster va demander la création du load balancer en le faisant pointer sur tous les noeuds du cluster sur un port attribué aléatoirement sur chaque noeud (comme pour un NodePort).

  - Un Load Balancer par service est coûteux.
  - « On premise » si on veut de l'équilibrage de charge il faut "tricher" avec une solution comme metallb.

— *ExternalName*: C'est un type de service qui fait le lien vers l'extérieur du cluster. Les ressources sont par exemple hébergées en dehors d'un cluster.

6.1 Service de type NodePort

1. On va utiliser la commande impérative suivante afin de générer le manifeste du service de type node port.

```
k expose deployment pythonapp --name pythonnp --target-port=5000 --port=9002 --type=NodePort --dry-run=client -o yaml > nodeport.ya
```

2. Modifiez le fichier en prenant comme nodeport 31234.
3. Quelle est la différence entre port/nodeport/target-port ? testez les nodes et le service.
4. Quels sont les inconvénients d'un service Node-Port ?Que manque-t-il dans cette architecture ?
5. Récupérez les "endpoints" reliés au service.
6. Complétez l'architecture avec un équilibreur de charge externe (un haproxy à installer par package) ?
7. Dessinez un schéma de cette architecture sur papier.

6.2 Service de type LoadBalancer

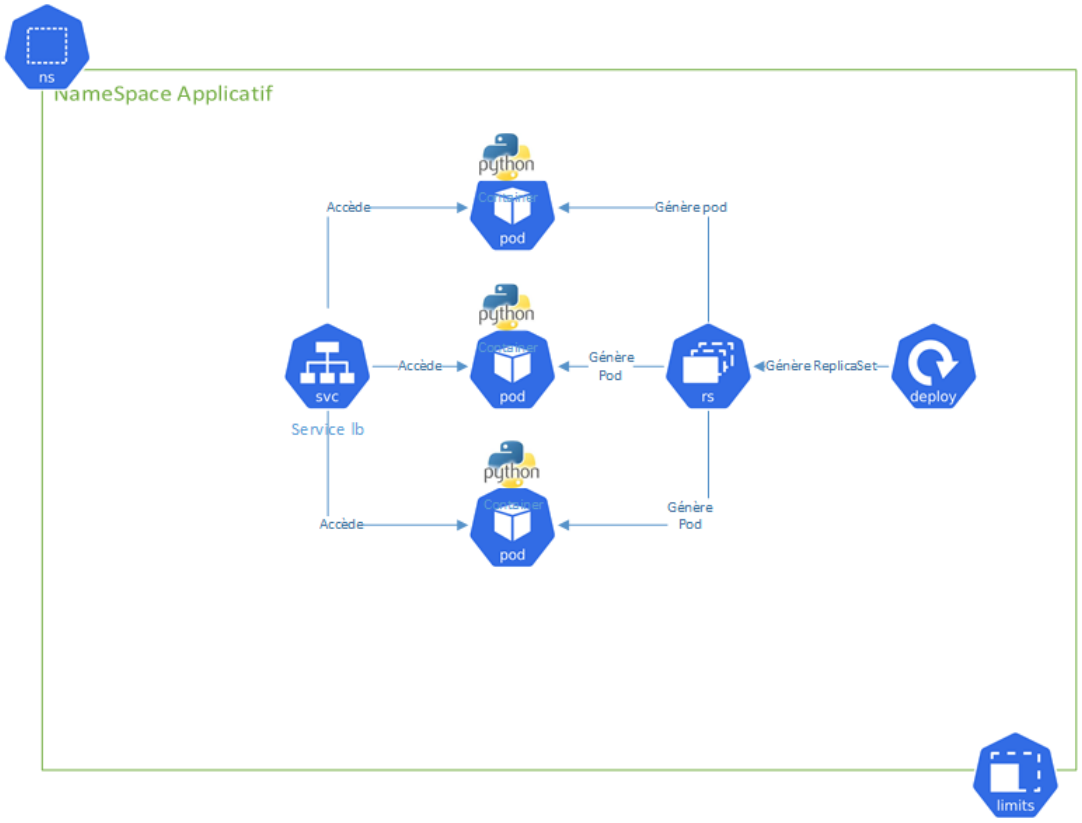

FIGURE 3 – Service

1. Créer un service de type loadbalancer. Pourquoi son EXTERNAL-IP est elle pending ?

2. Installer et utiliser Metallb [1] en mode L2 afin de rendre le service loadbalancer accessible depuis l'extérieur. [2]

   Vous utiliserez la méthode "installation par manifest" et vous configurerez une plage IP (voir "Defining the IPs to assign to the Load Balancer services")

7 Ingress

Une autre façon d'accéder à une application c'est d'utiliser un "ingress controller". Ce n'est pas un service comme précédemment mais un reverse proxy. Ce contrôleur se base sur l'en-tête http "Host" afin d'aiguiller la requête vers le bon "backend". "*haproxy-ingress*" est une des solutions possibles pour un "ingress controller". "*Nginx*" ou "*Traeffik*" en sont des autres.

1. Installer helm en suivant `https://helm.sh/docs/intro/install/`
2. Déployez le package de l' "ingress controller haproxy" en vous aidant de:
   `https://www.haproxy.com/fr/blog/use-helm-to-install-the-haproxy-kubernetes-ingress-controller/`
   et de
   `https://hub.helm.sh/charts/mirusresearch/haproxytech-haproxy-ingress`

   Le service du controller haproxy sera de type "LoadBalancer" afin de permettre à mettallb de publier l'adresse sur le réseau d'hôte.
3. Utilisez cet ingress afin de rendre accessible l'application Python.
4. Vérifiez que l'"ingress controller" fonctionne:

   ```
   http  pythonapp.172.18.0.250.nip.io/env
   ```

   NB: le domaine nip.io permet de résoudre tous les FQDN en adresse IP.
5. Visionnez dans votre navigateur la page de statistiques de l'"HAPROXY ingress". Faite un test de charge avec la commande ab (apachebench);
6. Vérifiez que l'"Ingress Controller" s'adapte si des pods disparaissent.
7. En suivant `https://github.com/jcmoraisjr/haproxy-ingress/tree/master/examples/deployment` générez un certificat auto-signé pour l'application Python et vérifiez en le bon fonctionnement.
8. Dessinez un schéma de l'architecture Ingress.

8 Stockage persistent et non réparti

La persistence des données est une difficulté commune des containers.

Les "ConfigMaps" et les "Secrets" permettent d'assurer la persistence des données mais pour des configurations et mots de passes. "HostPath" [3] permet de partager des fichiers ou des répertoires de l'hôte. "local-persistent-volumes" [4] permet de partager des disques.

Kind installe le "Rancher Localpath provisionner" [5] qui permet de provisionner du stockage avec un volume local dynamiquement réservé.

Bien sur l'utilisation d'un stockage réparti et persistent (NFS, CEPH, GLUSTERFS, ...) est possible. Il existe des solutions comme LongHorn, Rook , OpenEBS... qui installent du stockage persistent et réparti pour le cluster.

L'administateur du cluster va définir des classes de stockages (ssd, capacitif, nfs..). Le développeur lors de la création de ses pods fait une demande de volumes qui va s'appuyer sur un "persistent volume claim" qui pourra "piocher" de l'espace dans la classe de stockage défini précédemment.

Ainsi le développeur ne fait que demander de l'espace et ne s'occupe pas de sa gestion.

1. `https://metallb.universe.tf/`
2. degug de metallb via "kubetail -l component=speaker -n metallb-system" et " k logs -f controller-57f648cb96-khvk6 -n metallb-system"
3. `https://kubernetes.io/docs/concepts/storage/volumes/#hostpath`
4. `https://kubernetes.io/blog/2018/04/13/local-persistent-volumes-beta/`
5. `https://github.com/rancher/local-path-provisioner`

Dans votre environnement kind partage avec les noeuds Docker un espace dans votre directory "creation-cluster-kind" qui s'appelle shared-storage. Sa création est gérée par Kind et vous n'avez pas à la créer (c'est du Docker pas du Kubernetes).

Par défaut dans chaque noeud Docker, le répertoire /var/local-path-provisioner est utilisé pour stocker les volumes des PODS.

1. Quel est l'intérêt de faire du stockage local ? du stockage réparti.
2. Retrouvez la classe de stockage via kubectl ? y a t il des PV ou des PVC ?
3. En vous inspirant de la documentation Rancher Créez un PVC qui pointera sur la StorageClass "standard".
4. Créez un "deployment" nginx et donner le chemin /usr/share/nginx/html comme volume persistent. Vérifiez que le volume est conservé si vous supprimez le "deployment" ?pourquoi ne peut-on pas supprimer le pvc avant le pv ?

   NB: quelques commandes utiles pour vérifier le fonctionnement de la demande de stockage:

```
k -n local-path-storage logs -f -l app=local-path-provisioner
k get persistentvolumeclaim
k get persistentvolume
```

9 Utiliser Helm pour installer un dns unbound dans votre cluster.

1. Recherchez la charte pour unbound/stable et installez là:
2. Configurez un fichier yaml ([1]) qui permettra une résolution recursive depuis les hôtes Docker.

10 Containers éphémères pour débugguer

"Ephemeral container" [2] est une fonctionnalité qui permet de lancer un container dynamiquement dans le même POD que le container qui pose problème. Votre cluster kind dispose de cette "feature". [3] Testons la:

```
kubectl run mon-app --image=k8s.gcr.io/pause:3.1 --restart=Never
kubectl debug -it mon-app --image=busybox --target=mon-app
```

1. Expliquez le mode de fonctionnement de cette feature et les options ci-dessus.
2. Lancez un container nginx et débugguer le de la même façon.
3. Quel est le pid du process nginx ? pouvez-vous lister la configuration du container nginx sous cat /proc/pid-trouvé-par-ps/root/ ?
4. Rajoutez l'option –share-processes et –copy-to en remplacement de –target avec kubectl debug pour accéder totalement au container nginx. Affichez la configuration du fichier nginx.con.

11 Installation d'utilitaires..utiles

1. Installez et testez Krew.
2. Installez et testez kubetail.
3. Installez et testez stern.
4. Installez et testez kubectx.
5. Installez et testez contena-lentz (installation via snap).
6. Installez et testez le tableau de bord de Kubernetes.

1. voir `https://hub.helm.sh/charts/stable/unbound`
2. `https://kubernetes.io/docs/concepts/workloads/pods/ephemeral-containers/`
3. voir `https://martinheinz.dev/blog/49`