



TRAVAUX PRATIQUES

Kubernetes

| | |
|----------------------------------|----|
| #cluster..... | 2 |
| setup..... | 2 |
| inspect..... | 2 |
| #tooling..... | 2 |
| vscode..... | 2 |
| helm-install..... | 3 |
| Correction..... | 3 |
| #pods..... | 3 |
| imperative-pod-run..... | 3 |
| imperative-deployment-check..... | 3 |
| declarative-deployment..... | 4 |
| environment-vars..... | 4 |
| debugging..... | 4 |
| cleanup..... | 5 |
| #services..... | 5 |
| deployment-and-service..... | 5 |
| back-and-front..... | 5 |
| ingress..... | 7 |
| #storage..... | 8 |
| emptydir..... | 8 |
| persistentvolume..... | 9 |
| #configuration..... | 10 |
| configmap..... | 10 |
| secret..... | 10 |
| # updating..... | 11 |
| create-instances..... | 11 |
| update..... | 11 |
| #sharing..... | 12 |
| namespaces..... | 12 |
| quotas..... | 13 |
| rbac..... | 13 |
| #helm..... | 15 |
| environments..... | 15 |
| #cloud..... | 16 |
| azure..... | 16 |

#cluster

setup

Objectif : avoir un cluster à disposition.

Vous devez créer un cluster Kubernetes de développement. Pour cela, vous avez plusieurs options :

- Installer Docker Desktop : <https://www.docker.com/products/docker-desktop>
- Installer Minikube : <https://kubernetes.io/docs/tasks/tools/install-minikube/>
 - Utiliser le driver docker :

```
minikube config set driver docker
```

- Dans ce cas, installez kubectl <https://kubernetes.io/docs/tasks/tools/install-kubectl/>
- Créer un cluster PaaS dans le Cloud

Une fois l'installation terminée, vous devez être en mesure d'exécuter la commande suivante :

```
kubectl get nodes
```

inspect

Objectif : voir les éléments constituant le cluster.

Exécutez la commande suivante pour voir les nodes de votre cluster :

```
kubectl get nodes -o wide
```

Affichez les détails de l'un des nodes à l'aide de la commande suivante :

```
kubectl describe node <node-name>
```

Répondez aux questions suivantes :

- Pouvez-vous voir l'état du kubelet sur le node ?
- Quelle est la RAM disponible ?
- Quelle est la quantité de CPU disponible ?
- Pouvez-vous voir la consommation RAM et CPU des pods qui s'exécutent ?

Exécutez la commande suivante pour voir les pods système :


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

#tooling

vscode

Objectif : installer Visual Studio Code et son extension Kubernetes (optionnel).

Pour installer Visual Studio Code, rendez-vous ici : <https://code.visualstudio.com/>

Une fois l'installation effectuée, affichez les extensions de Visual Studio Code en cliquant sur le bouton correspondant dans la barre d'outils : 

Chez l'extension **Kubernetes** de Microsoft (ms-kubernetes-tools.vscode-kubernetes-tools) et installez-la.

helm-install

Objectif : installer Helm s'il n'est pas encore installé.

Exécutez la commande suivante :

```
helm version
```

Si Helm est déjà installé, vous obtenez une sortie similaire à celle-ci :

```
version.BuildInfo{Version:"v3.0.0", ...}
```

Si Helm n'est pas installé, ajoutez l'exécutable helm à votre path. Cela peut être fait manuellement ou via un gestionnaire de package, les instructions sont ici : <https://github.com/helm/helm#install>

Correction

Le code corrigé des exercices est ici : <https://bitbucket.org/epobb/kubernetes-exercises>

#pods

imperative-pod-run

Objectif : prendre en main la création impérative de ressources.

L'image learnbook/k8s-pinger:0.6 envoie des pings réguliers vers www.google.com toutes les 5 secondes.

Créez un deployment nommé pinger. Il doit déployer l'image learnbook/k8s-pinger:0.6 en lui attribuant le label app=pinger.

Affichez les pods et vérifiez qu'il y a bien un pod exécutant learnbook/k8s-pinger:0.6.

Affichez les logs des pods et vérifiez que le pod envoie bien des pings.

imperative-deployment-check

Objectif : vérifier ce qui se passe en cas de défaillance d'un pod.

Affichez les deployments et les pods. Vérifiez qu'il y a bien une instance du pod pinger disponible.

Supprimez le pod ayant le label app=pinger grâce à la commande :

```
kubectl delete pod -l app=pinger
```

Affichez les deployments et les pods. Vérifiez que le pod précédent est en cours d'arrêt et qu'un nouveau pod a été démarré pour le remplacer.

Supprimez le deployment pinger.

```
kubectl delete deployment <nom>
```

Affichez les deployments et les pods. Vérifiez que les pods pingr sont supprimés.

declarative-deployment

Objectif : créer des ressources de manière déclarative.

Créez un fichier YAML décrivant un deployment de l'image learnbook/k8s-pinger:0.6 en lui attribuant le label app=pinger. Utilisez la commande kubectl apply pour créer le deployment correspondant dans le cluster.

Affichez les pods et les deployments pour vérifier que le deployment a bien été créé, ainsi que le pod correspondant.

environment-vars

Objectif : mettre à jour des ressources créées de manière déclarative. Pour cela vous allez passer des variables d'environnement aux containers.

Modifiez le fichier YAML décrivant le deployment pour ajouter les variables d'environnement suivantes :

- target: www.qwant.com
- delay : 30

Appliquez la modification.

Affichez les pods et les deployments pour vérifier que le pod a bien été mis à jour.

Affichez les logs du pod en sélectionnant ceux ayant le label app=pinger. Vérifiez que l'on voit bien les pings sur www.qwant.com.

Durant un moment on voit des pings vers www.google.com dans les logs. Pourquoi ?

debugging

Objectif : vérifier le comportement de Kubernetes en cas d'erreur et apprendre à identifier la source des erreurs.

Créez un fichier YAML décrivant un deployment de l'image learnbook/debugging1:0.6 en lui attribuant le label app=debugging. Utilisez la commande kubectl apply pour créer le deployment correspondant dans le cluster.

Affichez les pods et les deployments pour vérifier que le deployment a bien été créé, ainsi que le pod correspondant.

Dans quel état est le deployment ? Dans quel état est le pod ?

Utilisez la commande kubectl describe pour comprendre ce qui s'est passé dans le pod.

Créez un fichier YAML décrivant un deployment de l'image mongo-express:0.49.0 en lui attribuant le label app=debugging2. Utilisez la commande kubectl apply pour créer le deployment correspondant dans le cluster.

Affichez les pods et les deployments pour vérifier que le deployment a bien été créé, ainsi que le pod correspondant.

Vous pouvez observer que le pod redémarre sans cesse. Déterminez la raison.

cleanup

Objectif : libérer les ressources créées.

Affichez l'ensemble des deployments et pods créés.

Vous devez supprimer l'ensemble de ces ressources en utilisant la commande `kubectl delete` et les fichiers YAML précédemment créés.

#services

deployment-and-service

Objectif : exposer une application web à l'extérieur.

Créez un deployment de `learnbook/k8s-front:0.5` nommé `front`, et portant un label `app=front`. Il s'agit d'une application web écoutant sur le port 80.

Exposez l'application `front` à l'extérieur à l'aide d'un service de type `LoadBalancer` qui écoute sur le port 8080.

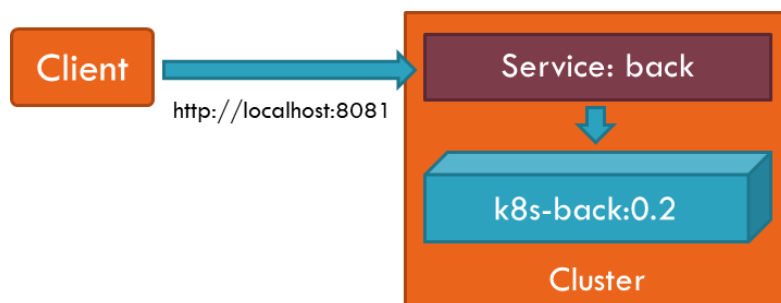
Affichez <http://localhost:8080> dans le navigateur. Vérifiez que l'application est accessible.

Supprimez toutes les ressources créées.

back-and-front

Objectif : exposer une API de manière interne au cluster.

Dans un premier temps, nous souhaitons obtenir l'architecture applicative suivante :

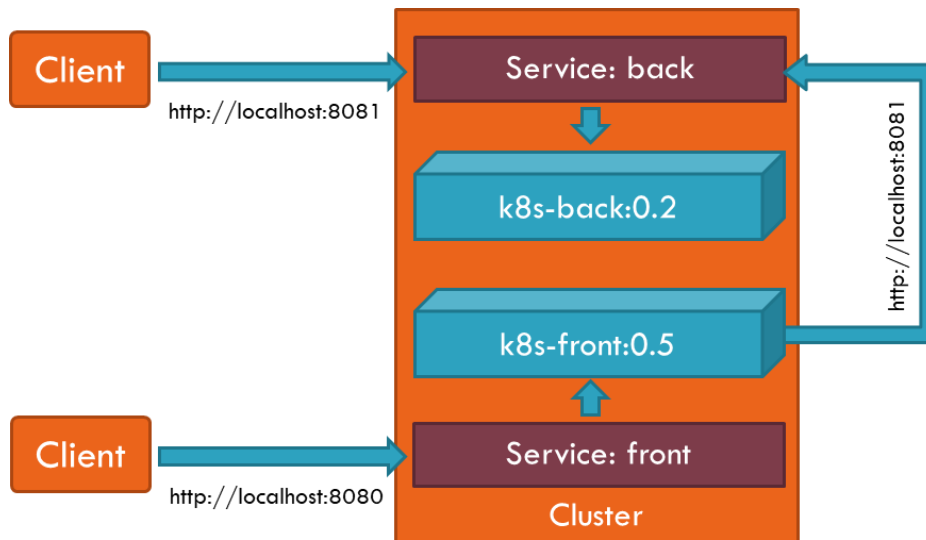


Créez un deployment de `learnbook/k8s-back:0.2` nommé `back`, et portant un label `app=back`. Il s'agit d'une API écoutant sur le port 80.

Exposez l'application `back` à l'extérieur à l'aide d'un service de type `LoadBalancer` qui écoute sur le port 8081.

Affichez <http://localhost:8081/v1/currenttime> dans le navigateur. Vérifiez que l'API répond.

Nous souhaitons maintenant déployer l'application web `front` qui utilisera ce service via son adresse publique. L'architecture applicative sera la suivante :



Créez un deployment de learnbook/k8s-front:0.5 nommé front, et portant un label app=front. Ce deployment doit passer les variable d'environnement suivantes au conteneur :

- API_EXTERNAL_URL : <http://localhost:8081> (ou pour minikube, celle qu'il vous attribue)
- API_CLUSTER_URL : <http://back-inside>

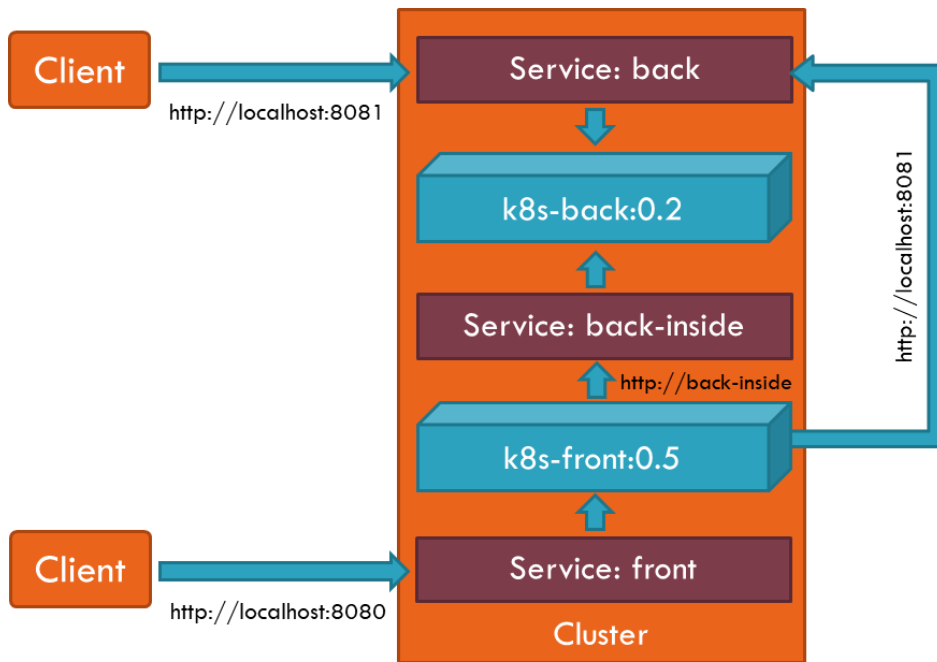
C'est grâce à la variable API_EXTERNAL_URL que l'application front saura comment invoquer l'API. La variable API_CLUSTER_URL ne correspond pour l'instant à aucune adresse existante.

Exposez l'application front à l'extérieur à l'aide d'un service de type LoadBalancer qui écoute sur le port 8080.

Affichez <http://localhost:8080> dans le navigateur. Vérifiez que l'application est accessible.

A ce stade, l'application doit afficher des valeurs dans sa zone « API call outside cluster » mais pas de valeur dans sa zone « API call inside cluster ».

Nous souhaitons enfin que l'application front soit capable d'invoquer l'API back sans passer par une adresse publique mais directement par le cluster. C'est le but de la variable API_CLUSTER_URL passée précédemment. L'architecture applicative sera la suivante :



Exposez l'application back à l'intérieur du cluster à l'aide d'un service de type ClusterIP nommé back-inside qui écoute sur le port 80.

Affichez <http://localhost:8080> dans le navigateur. Vérifiez que l'application affiche maintenant des valeurs à la fois dans les zones « API call outside cluster » et « API call inside cluster ».

Supprimez toutes les ressources créées.

ingress

Objectif : exposer le front et le back chacun sur le port par défaut (80) et sur sa propre DNS.

En préalable, ajoutez les entrées suivantes à votre fichier hosts :

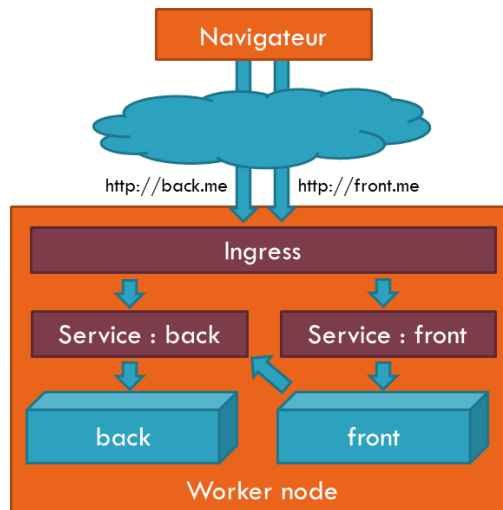
```
127.0.0.1    front.me
127.0.0.1    back.me
```

L'emplacement du fichier hosts dépend de votre machine. En général il s'agit de /etc/hosts sur Linux et C:\Windows\System32\drivers\etc sur Windows.

Si vous utilisez un cluster hors de votre machine ou Minikube, il faudra utiliser l'adresse IP du contrôleur Ingress plutôt que 127.0.0.1. Par exemple pour Minikube utilisez l'IP retournée par :

```
minikube ip
```

L'architecture visée est la suivante :



Créez un deployment de `learnbook/k8s-front:0.4` et un deployment de `learnbook/k8s-back:0.2`. Exposez chaque deployment par un service de type ClusterIP. Ils ne sont pas accessibles depuis l'extérieur du cluster.

Créez un objet ingress qui routes les requêtes <http://front.me> vers le service front et <http://back.me> vers le service back.

Ouvrez dans votre navigateur <http://front.me> et <http://back.me>. Aucune adresse ne répond car il n'y a pas de contrôleur ingress.

Sur Docker Desktop ou Azure AKS

Vous allez installer le contrôleur suivant : <https://kubernetes.github.io/ingress-nginx>

Pour cela, exécutez les commandes suivantes :

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update
helm install default-ingress ingress-nginx/ingress-nginx
```

Sur Minikube

Installez un contrôleur ingress avec la commande suivante :

```
minikube addons enable ingress
```

Ouvrez dans votre navigateur <http://front.me> et <http://back.me>. Les deux adresses répondent.

Bonus : faites en sorte que la page <http://front.me> affiche les appels à l'API aussi bien externes qu'internes au cluster.

Supprimez toutes les ressources créées.

Conservez les fichiers YAML utilisés, vous en aurez besoin lors du TP #helm / environments

#storage

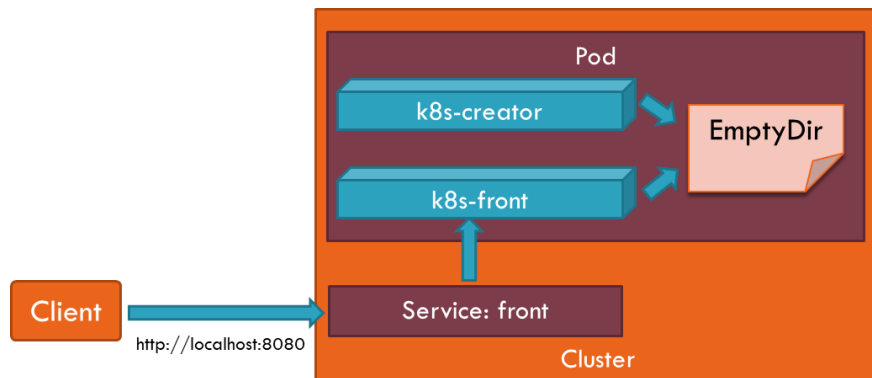
emptydir

Objectif : créer et utiliser un volume de type emptydir afin de partager des fichiers entre conteneurs dans un pod.

Une image `learnbook/k8s-creator:0.2` est capable de créer des fichiers au nom aléatoire dans son répertoire `/shared`. L'image `learnbook/k8s-front:0.5` est capable d'afficher les fichiers présents dans son répertoire `/shared`.

Vous allez exécuter ces deux images au sein d'un même pod afin que front puisse voir les fichiers créés par creator.

L'architecture visée est la suivante :



Créez un deployment contenant à la fois `learnbook/k8s-creator:0.2` et `learnbook/k8s-front:0.5` au sein du même pod.

Exposez l'application front à l'extérieur à l'aide d'un service de type LoadBalancer qui écoute sur le port 8080.

Affichez <http://localhost:8080/files> dans le navigateur. Aucun fichier n'est présent dans le répertoire `/shared`. Pourquoi ?

Ajoutez un volume de type `emptyDir` dans le pod, et exposez-le simultanément à front et creator en tant que répertoire `/shared`.

Affichez <http://localhost:8080/files> dans le navigateur. Les fichiers apparaissent.

Supprimez toutes les ressources créées.

Créez de nouveau le pod avec les deux conteneurs partageant le même volume et le service.

Affichez <http://localhost:8080/files> dans le navigateur. Il n'y a plus les fichiers précédemment créés. Pourquoi ?

persistentvolume

Objectif : utiliser un `PersistentVolume` pour conserver les fichiers lorsque le pod meurt.

Utilisez le même déploiement que précédemment mais remplacez le `emptyDir` par une référence à un `PersistentVolumeClaim` que vous créez.

Affichez <http://localhost:8080/files> dans le navigateur. Fichiers.

Supprimez le déploiement et le service.

Recréez le déploiement et le service.

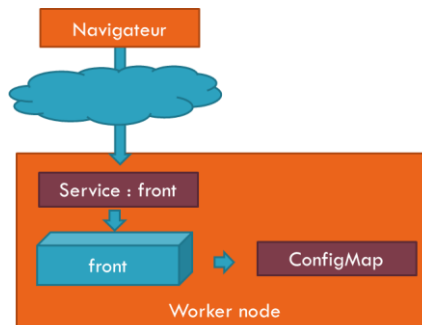
Affichez <http://localhost:8080/files> dans le navigateur. Cette fois les fichiers précédents sont affichés, ils n'ont pas été perdus lors de la mort du pod.

Supprimez toutes les ressources créées.

#configuration

configmap

Objectif : créer et utiliser un objet ConfigMap de différentes manières. L'architecture cible est la suivante :



Créez un objet ConfigMap avec les clés et valeurs suivantes :

- message: "Hello world"
- config: "Debug"
- magicNumber: "42"

Créez un deployment de learnbook/k8s-front:0.4 exposé par un service.

Affichez <http://localhost:8080/files> dans le navigateur. Aucun fichier.

Modifiez le deployment pour qu'il utilise en tant que répertoire /shared le ConfigMap créé.

Affichez <http://localhost:8080/files> dans le navigateur. Les clés sont affichées.

Modifiez le deployment pour que toutes les entrées du ConfigMap créé soient injectées en tant que variables d'environnement dans le conteneur.

Affichez <http://localhost:8080/details> dans le navigateur. Les trois variables « message », « config » et « magicNumber » du ConfigMap doivent apparaître.

Supprimez toutes les ressources créées.

secret

Objectif : créer et utiliser un objet Secret pour un paramétrage confidentiel.

Créez un objet Secret avec les clés et valeurs suivantes :

- message: "Hello secret world"
- config: "Hidden"
- magicNumber: "0"

Créez un deployment de learnbook/k8s-front:0.4 exposé par un service.

Affichez <http://localhost:8080/files> dans le navigateur. Aucun fichier.

Modifiez le deployment pour qu'il utilise en tant que répertoire /shared le Secret créé.

Affichez <http://localhost:8080/files> dans le navigateur. Les clés sont affichées.

Modifiez le deployment pour que toutes les entrées du Secret créé soient injectées en tant que variables d'environnement dans le conteneur.

Affichez <http://localhost:8080/details> dans le navigateur. Les trois variables « message », « config » et « magicNumber » du ConfigMap doivent apparaître.

Supprimez toutes les ressources créées.

updating

create-instances

Objectif : observer le comportement du ReplicaSet lorsque plusieurs replicas sont demandées.

Créez un deployment de learnbook/k8s-front:0.4 exposé sur le port 8080 par un service de type LoadBalancer.

Affichez <http://localhost:8080> dans le navigateur et notez le nom du serveur qui exécute le pod.

Affichez les ReplicaSets et observez celui correspondant à votre application.

Lancez un autre terminal dans lequel vous exécutez la commande suivante :

```
kubectl get pods -l app=front --watch
```

Modifiez le deployment de learnbook/k8s-front:0.4 afin que l'application soit hébergée sur 10 pods. Observez dans l'autre terminal comme les pods sont créés.

Affichez les ReplicaSets et observez celui correspondant à votre application. Vérifiez qu'il y a bien 10 pods qui exécutent l'application.

Affichez <http://localhost:8080> dans le navigateur et notez le nom du serveur qui exécute le pod. Rafraîchissez la page et vérifiez que le nom du serveur change. S'il ne change pas, vous pouvez utiliser une query string aléatoire sur l'URL, par exemple <http://localhost:8080/?iudf>

Modifiez le deployment de learnbook/k8s-front:0.4 afin que l'application soit hébergée sur 3 pods. Observez dans l'autre terminal comme les pods sont détruits.

Affichez les ReplicaSets et observez celui correspondant à votre application. Vérifiez qu'il y a bien 3 pods qui exécutent l'application.

Supprimez toutes les ressources créées.

update

Objectif : procéder à des mises à jour de type rolling update en vérifiant le comportement du cluster.

Créez un deployment de learnbook/k8s-front:0.5 tournant sur 3 pods et exposé sur le port 8080 par un service de type LoadBalancer. Pour cela vous pouvez reprendre la configuration utilisée pour l'exercice précédent.

Lancez un autre terminal dans lequel vous exécutez la commande suivante :

```
kubectl get pods -l app=front --watch
```

Modifiez le deployment de `learnbook/k8s-front:0.5` en lui passant un message d'accueil dans la variable d'environnement « `WELCOME_MESSAGE` ».

Vérifiez dans le deuxième terminal que tous les pods sont bien mis à jour successivement.

Affichez <http://localhost:8080> dans le navigateur. Vérifiez que le message d'accueil est bien affiché.

Modifiez le deployment de `learnbook/k8s-front:0.5` pour qu'il utilise maintenant `learnbook/k8s-front:invalid`.

Vérifiez dans le deuxième terminal qu'un seul un pod supplémentaire est créé en erreur, et que les 3 anciens pods tournent toujours.

Affichez <http://localhost:8080> dans le navigateur. Vérifiez que l'application fonctionne toujours avec sa version précédente.

Affichez les objets de type `ReplicaSet` pour l'application et vérifiez qu'il y en a deux dont l'un n'a créé qu'un seul pod.

Modifiez le deployment de `learnbook/k8s-front:0.5` pour qu'il utilise de nouveau `learnbook/k8s-front:0.5`.

Vérifiez dans le deuxième terminal que seul le un pod en erreur est supprimé, et que les 3 anciens pods tournent toujours.

Supprimez toutes les ressources créées.

#sharing

namespaces

Objectif : créer deux déploiements parallèles de l'application front ; l'un de production qui écoute sur le port 8080, l'autre de test qui écoute sur le port 8082.

Créez un deployment de `learnbook/k8s-front:0.4` exposé sur le port 8080 par un service de type `LoadBalancer`.

Affichez <http://localhost:8080/files> dans le navigateur. Cliquez sur le bouton « Add file » pour ajouter quelques fichiers.

Créez un namespace nommé « tests ».

Créez dans le namespace « tests » un deployment de `learnbook/k8s-front:0.4` exposé sur le port 8082 par un service de type `LoadBalancer` lui-même aussi dans le namespace « tests ».

Minikube : vous obtiendrez l'URL de ce service avec :

```
minikube service front --url --namespace
```

Affichez <http://localhost:8082/files> dans le navigateur. Assurez-vous que les fichiers de l'autre application ne sont pas affichés. Cliquez sur le bouton « Add file » pour ajouter quelques fichiers.

Affichez tous les deployments et services créés en utilisant le switch `--all-namespaces` sur la commande `kubectl get`.

Supprimez toutes les ressources créées.

quotas

Objectifs : limiter les ressources utilisables dans un namespace.

Créez un namespace nommé *quotas-test*.

Ajoutez au namespace *quotas-test* l'objet ResourceQuota suivant :

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: ram-quotas
spec:
  hard:
    requests.memory: 500Mi
    limits.memory: 1Gi
```

Exécutez la ligne de commande suivante pour afficher la consommation du quota :

```
kubectl get resourcequota ram-quotas -n=quotas-test --output=yaml
```

Vous pouvez constater que l'utilisation actuelle du quota est de 0.

Ajoutez dans le namespace *quotas-test* un déploiement de l'image *learnbook/k8s-pinger:0.6* nommé *pinger1*, avec les contraintes de mémoire suivantes :

```
resources:
  requests:
    memory: "300Mi"
  limits:
    memory: "400Mi"
```

Exécutez la ligne de commande suivante pour afficher la consommation du quota :

```
kubectl get resourcequota ram-quotas -n=quotas-test --output=yaml
```

Vous pouvez constater que l'utilisation actuelle du quota tient compte du Deployment.

Ajoutez dans le namespace *quotas-test* un déploiement de l'image *learnbook/k8s-pinger:0.6* nommé *pinger2*, avec les mêmes contraintes de mémoire que précédemment :

```
resources:
  requests:
    memory: "300Mi"
  limits:
    memory: "400Mi"
```

Affichez les ReplicaSets du namespace *quotas-test* avec *get* puis *describe*. Vous pouvez constater que le deuxième Deployment n'a pas abouti par manque de mémoire.

Exécutez la ligne de commande suivante pour afficher la consommation du quota :

```
kubectl get resourcequota ram-quotas -n=quotas-test --output=yaml
```

Supprimez le namespace nommé *quotas-test*. Il est normal que la commande prenne du temps, le temps que les pingers soient arrêtés.

rbac

Objectif : accéder à l'API Kubernetes depuis un Pod.

Créez un deployment de *learnbook/k8s-gotty:0.2* (il écoute sur le port 8080) exposé sur le port 8088 par un service de type LoadBalancer.

Affichez <http://localhost:8088> dans le navigateur. Vous obtenez un terminal connecté au pod *gotty*.

Exécutez la commande suivante dans le terminal du navigateur (vous pouvez coller avec Shift-Insert) :

```
curl https://kubernetes/api/v1/namespaces/default/secrets/ --insecure
```

Vous obtenez une erreur *401 Unauthorized*. Cette erreur est normale : vous venez de tenter un accès non identifié à l'API Kubernetes.

Vous allez maintenant utiliser l'identité du Pod pour accéder à l'API Kubernetes.

Exécutez les commandes suivantes dans le terminal du navigateur :

```
TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)
curl -H "Authorization: Bearer $TOKEN"
https://kubernetes/api/v1/namespaces/default/secrets --insecure
```

Vous obtenez une erreur *403 Forbidden*. Cette erreur est normale : le ServiceAccount *default* avec lequel votre Pod tourne n'a pas le droit de lister les secrets.

Docker Desktop : Attention : si à ce stade vous n'obtenez pas de *Forbidden*, il s'agit d'un bug connu¹. La solution consiste à exécuter cette commande (dans votre ligne de commande normale, pas au sein du terminal *gotty*) :

```
kubectl patch clusterrolebinding docker-for-desktop-binding --type=json --
patch '[{"op": "replace", "path": "/subjects/0/name",
"value": "system:serviceaccounts:kube-system"}]'
```

Récupérez dans un fichier local le YAML situé ici : <https://bitbucket.org/epobb/kubernetes-exercises/src/master/sharing/rbac/serviceaccount.yaml> et appliquez-le avec `kubectl apply`.

3 objets sont créés :

- Le ServiceAccount nommé *gotty*.
- Le Role nommé *list-secrets* qui définit le droit de lister les objets Secret dans le namespace *default*.
- Le RoleBinding qui connecte ce ServiceAccount à ce Role.

Exécutez les commandes suivantes dans le terminal du navigateur (ce sont les mêmes que ci-dessus) :

```
TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)
curl -H "Authorization: Bearer $TOKEN"
https://kubernetes/api/v1/namespaces/default/secrets --insecure
```

Vous obtenez toujours l'erreur 403 car le Pod *gotty* utilise le ServiceAccount *default*.

Modifiez le Deployment pour que le Pod *gotty* utilise le ServiceAccount *gotty* créé précédemment.

```
spec:
  serviceAccountName: gotty
  containers:
  - image: learnbook/k8s-gotty:0.2
```

Attendez que le terminal du navigateur se recharge, puis exécutez les commandes suivantes dans le terminal du navigateur (ce sont les mêmes que ci-dessus) :

```
TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)
curl -H "Authorization: Bearer $TOKEN"
https://kubernetes/api/v1/namespaces/default/secrets --insecure
```

¹ <https://github.com/docker/for-mac/issues/4774>

La liste des secrets est maintenant affichée.

Exécutez la commande suivante dans le terminal du navigateur (elle liste les Pods au lieu de lister les Secrets) :

```
curl -H "Authorization: Bearer $TOKEN"  
https://kubernetes/api/v1/namespaces/default/pods --insecure
```

Vous obtenez une erreur 401 *Unauthorized*. Cette opération n'a pas été autorisée dans le Role *list-secrets*.

#helm

environments

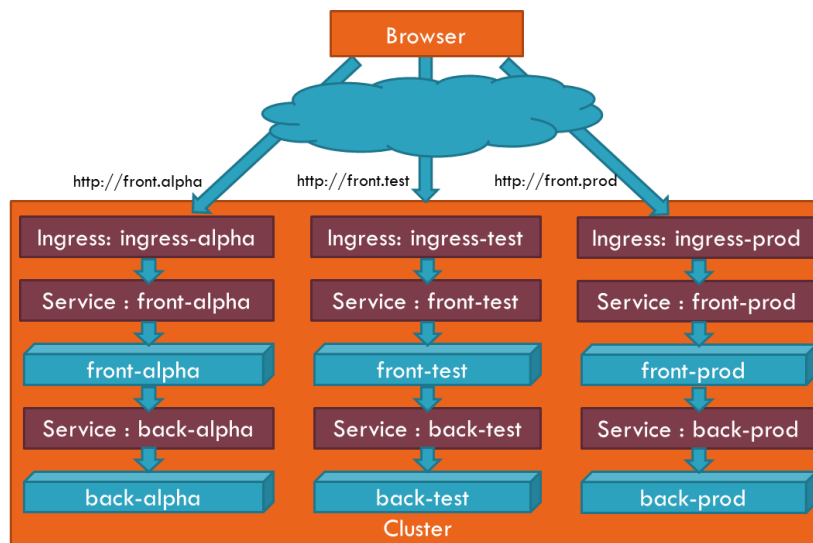
Objectif: transformer la stack front-back en chart Helm, puis en exécuter trois instances.

En préalable, ajoutez les entrées suivantes à votre fichier hosts :

```
127.0.0.1 front.alpha  
127.0.0.1 back.alpha  
127.0.0.1 front.test  
127.0.0.1 back.test  
127.0.0.1 front.prod  
127.0.0.1 back.prod
```

L'emplacement du fichier hosts dépend de votre machine. En général il s'agit de `/etc/hosts` sur Linux et `C:\Windows\System32\drivers\etc` sur Windows.

L'architecture visée est la suivante :



Pour faciliter le déploiement et la mise à jour, vous devez créer un chart Helm contenant toute la pile front (deployment et service), back (deployment et service) et ingress.

Afin de gagner du temps, reprenez les fichiers YAML créés lors du TP #services / ingress.

Lors de l'installation du chart, ce dernier doit afficher un texte contenant les DNS des services. Par exemple : « Vous pouvez invoquer `http://front.alpha` et `http://back.alpha`. »

Le chart doit prendre en paramètre le nom de l'environnement : « alpha », « test » ou « prod ».

Une fois votre chart créé, déployez-le 3 fois pour créer les 3 environnements.

Vérifiez que vous pouvez bien accéder aux 3 environnements avec votre navigateur, et créez quelques fichiers dans chacun pour vérifier qu'ils sont bien indépendants.

Supprimez toutes les ressources créées.

#cloud

azure

Objectif : créer et utiliser un cluster AKS.

Installez la CLI Azure ou bien utilisez le Shell Azure <https://shell.azure.com/>

Si vous utilisez la CLI sur votre poste, connectez-vous à votre compte Azure avec la commande suivante :

```
az login
```

Dans le cas où vous avez plusieurs abonnements, listez-les puis sélectionnez-en :

```
az account list
az account set --subscription "<name here>"
```

Créez un groupe de ressources dans la région de votre choix. La région peut être **westeurope**.

```
az group create --name AKS-TEST --location westeurope
```

Créez un cluster dans le groupe :

```
az aks create --name cloudk8s --resource-group AKS-TEST --node-count 2 --
generate-ssh-keys
```

Enfin, ajoutez un contexte à kubectl :

```
az aks get-credentials --name cloudk8s --resource-group AKS-TEST
```

Note : c'est le user view qui est ajouté par défaut.

Testez la connexion au cluster :

```
kubectl get nodes
```

inspect

Objectif : voir les éléments constituant le cluster. Ce TP est le même que celui effectué au début.

Exécutez la commande suivante pour voir les nodes de votre cluster :

```
kubectl get nodes -o wide
```

Affichez les détails de l'un des nodes à l'aide de la commande suivante :

```
kubectl describe node <node-name>
```

Répondez aux questions suivantes :

- Pouvez-vous voir l'état du kubelet sur le node ?
- Quelle est la RAM disponible ?
- Quelle est la quantité de CPU disponible ?
- Pouvez-vous voir la consommation RAM et CPU des pods qui s'exécutent ?

Exécutez la commande suivante pour voir les pods système :

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