Container communication via localhost

Création d’images personnalisées

dockerfile

```
# Premier stage
FROM httpd:latest as s1
RUN apt-get update
RUN apt-get install -y curl
RUN apt-get install -y vim
RUN apt-get install -y netcat-openbsd
RUN apt-get install -y netcat-traditional
RUN apt-get install -y telnet
RUN apt-get install -y net-tools

# Deuxième stage
FROM ubuntu:latest as s2
RUN apt-get update
RUN apt-get install -y curl
RUN apt-get install -y vim
RUN apt-get install -y netcat-openbsd
RUN apt-get install -y netcat-traditional
RUN apt-get install -y telnet
RUN apt-get install -y net-tools
```

Explication du Dockerfile

Dans le Dockerfile fourni, vous avez deux stages, `s1` et `s2`. Ce sont des exemples de constructions multi-stage.

- **s1** : Ce stage commence à partir d'une image de `httpd:latest` et installe plusieurs paquets tels que `curl`, `vim`, `netcat-openbsd`, `netcat-traditional`, `telnet`, et `net-tools`.
- **s2** : Ce deuxième stage commence à partir d'une image de `ubuntu:latest` et installe les mêmes paquets que le premier stage.

Commandes pour Créer, Tagger et Pousser les Images

1. Premier Stage (s1)

Pour créer et taguer l'image du premier stage :

```
docker build --target s1 -t mon_nom_utilisateur/mon_image_s1:latest .


// Exple
docker build --target s1 -t elfn/apache .
```

Et pour pousser cette image :

```
docker push mon_nom_utilisateur/mon_image_s1:latest

// Exple
docker push elfn/apache
```

2. Deuxième Stage (s2)

Pour créer et taguer l'image du deuxième stage :

```
docker build --target s2 -t mon_nom_utilisateur/mon_image_s1:latest .

// Exple
docker build --target s2 -t elfn/ubuntu .
```

Et pour pousser cette image :

```
docker push mon_nom_utilisateur/mon_image_s2:latest

// Exple
docker push elfn/ubuntu
```

Notes Importantes

1. **Login** : N'oubliez pas de vous connecter à Docker Hub ou à tout autre registre Docker que vous utilisez avant de pousser les images avec la commande `docker login`.
2. **Nettoyage** : Pour rendre votre Dockerfile plus efficace et propre, vous pourriez fusionner les commandes `RUN` en une seule, ce qui réduit le nombre de couches créées dans votre image :

   ```
   RUN apt-get update && apt-get install -y \
       curl vim netcat-openbsd netcat-traditional telnet net-tools
   ```

3. **Nom d'Utilisateur et Image** : N'oubliez pas de remplacer `mon_nom_utilisateur`, `mon_image_s1`, et `mon_image_s2` par votre nom d'utilisateur Docker et les noms d'images souhaités.

4. **Netcat Package** : Il semble que les paquets `netcat-openbsd` et `netcat-traditional` sont installés dans les deux stages. Habituellement, vous choisiriez l'un ou l'autre, selon vos préférences ou vos besoins spécifiques.

Création du pod `networking-pod1` contenant les conteneurs cont1 et cont2 issues des images créés et pushées.

monpod.yml

```
kind: Pod
apiVersion: v1
metadata:
  name: networking-pod1
spec:
  containers:
    - name: cont1
      image: elfn/ubuntu
      command: ["/bin/bash", "-c", "while true; do echo Learning kubernetes networking; sleep 5; done"]
    - name: cont2
      image: elfn/apache # Apache image
      ports:
        - containerPort: 80
```

Explication du YAML

- `kind: Pod` : Il indique que la ressource à créer est un Pod.
- `apiVersion: v1` : Il spécifie la version de l'API à utiliser pour créer le Pod.
- `metadata` : Il contient des métadonnées sur le Pod, telles que son nom.
  - `name: networking-pod1` : Il spécifie le nom du Pod.
- `spec` : Il définit la spécification du Pod.
  - `containers` : Il définit la liste des conteneurs à créer dans le Pod.
    - `name: cont1` : Le premier conteneur a pour nom `cont1`.
      - `image: elfn/ubuntu` : Ce conteneur est basé sur l'image `elfn/ubuntu`.
      - `command` : Le conteneur exécutera cette commande au démarrage. Ici, il exécute une boucle infinie qui imprime un message toutes les 5 secondes.
    - `name: cont2` : Le deuxième conteneur a pour nom `cont2`.
      - `image: elfn/apache` : Ce conteneur est basé sur l'image `elfn/apache`.
      - `ports` : Il spécifie les ports du conteneur.

- `containerPort: 80` : Le port 80 du conteneur est exposé.

Utilisation du YAML

Pour créer ce Pod dans votre cluster Kubernetes, sauvegardez le YAML dans un fichier, par exemple `monpod.yaml`, puis appliquez-le avec `kubectl apply` :

```
shCopy code
kubectl apply -f monpod.yaml
```

Vérification du Pod

Une fois le Pod créé, vous pouvez vérifier son état et ses conteneurs avec les commandes suivantes :

Pour voir l'état du Pod :

```
shCopy code
kubectl get pods networking-pod1
```

Pour voir les logs d'un conteneur spécifique dans le Pod :

```
shCopy code
kubectl logs networking-pod1 -c cont1
```

Ou pour le deuxième conteneur :

```
shCopy code
kubectl logs networking-pod1 -c cont2
```

Conclusion

Ce YAML décrit un Pod avec deux conteneurs ayant des rôles différents, l'un servant probablement une application web avec Apache et l'autre affichant un message en boucle. C'est un exemple typique de la façon dont vous pouvez déployer des applications multi-conteneurs dans Kubernetes.

Communication

Container access

Pour accéder aux conteneurs dans le Pod que vous avez créé, vous pouvez utiliser la commande `kubectl exec`. Voici comment vous pouvez le faire pour chacun des conteneurs spécifiés dans votre YAML.

1. Accéder au Premier Conteneur (`cont1`):

Vous pouvez exécuter une commande interactive, telle qu'un shell bash, dans le premier conteneur (`cont1`) avec la commande suivante :

```
kubectl exec -it networking-pod1 -c cont1 -- /bin/bash

kubectl exec testpod -c c00 -it /bin/bash

kubectl exec testpod -c c01 -it /bin/bash
```

Dans cette commande:

- `it` permet une interaction avec le terminal du conteneur.
- `c cont1` spécifie le nom du conteneur que vous souhaitez accéder dans le Pod.

2. Accéder au Deuxième Conteneur (`cont2`):

De même, pour accéder au deuxième conteneur (`cont2`), utilisez :

```
kubectl exec -it networking-pod1 -c cont2 -- /bin/bash
```

Vérification de l'Accès au Conteneur

Une fois que vous êtes à l'intérieur du conteneur, vous pouvez exécuter des commandes comme si vous étiez sur une machine Linux normale. Par exemple, pour vérifier la liste des processus en cours d'exécution, vous pouvez utiliser la commande `ps aux`.

Quitter le Shell du Conteneur

Pour sortir du shell du conteneur et revenir à votre terminal local, tapez `exit`.

Notes

- Assurez-vous que le shell bash est disponible dans les conteneurs. Si le shell bash n'est pas disponible, vous pourriez essayer avec `/bin/sh` ou un autre shell disponible dans le conteneur.
- Si vous ne spécifiez pas le conteneur avec l'option `c`, `kubectl exec` essayera d'accéder au premier conteneur dans le Pod par défaut.

Ces commandes vous permettent d'accéder et d'interagir directement avec les environnements d'exécution des conteneurs dans votre Pod Kubernetes.

Connexion and communication via localhost

```
// in cont1  and
telnet localhost 8081

//  in cont2
netcat -l -p 8081
```

Si vous exécutez `telnet localhost 8081` dans `cont1` et `netcat -l -p 8081` dans `cont2` au sein du même Pod Kubernetes, vous essayez d'établir une connexion entre ces deux conteneurs via le port `8081`. Voici comment cela fonctionne :

Dans `cont2` :

- `netcat -l -p 8081` :
  - `l` fait que `netcat` opère en mode écoute, attendant une connexion entrante.
  - `p 8081` spécifie le port sur lequel `netcat` écoute.

Donc, `cont2` écoute les connexions entrantes sur le port `8081`.

Dans `cont1` :

- `telnet localhost 8081` :
  - `telnet` est utilisé pour se connecter à des serveurs distants.
  - `localhost` indique que la connexion doit être établie sur la même machine (ou le même conteneur dans ce cas).
  - `8081` est le numéro du port auquel `telnet` doit se connecter.

Donc, en utilisant cette commande dans `cont1`, vous essayez de vous connecter au port `8081` de `cont1` lui-même, et non de `cont2`.

Problème Potentiel:

Étant donné que `telnet localhost 8081` dans `cont1` tente de se connecter à un service sur le port `8081` de `cont1`, et non de `cont2`, cette tentative de connexion échouera probablement, sauf si un service dans `cont1` écoute aussi sur le port `8081`.

Solution:

Si vous souhaitez que `cont1` se connecte à `cont2` sur le port `8081`, vous devez utiliser le nom du Pod ou l'adresse IP de `cont2` dans la commande `telnet`, plutôt que `localhost`. Cependant, les conteneurs au sein d'un même Pod partagent le même espace réseau, donc `localhost` devrait en théorie permettre à `cont1` de se connecter à `cont2`. Si ce n'est pas le cas, une vérification du réseau et de la configuration du Pod pourrait être nécessaire.

Conclusion:

Pour établir une connexion entre `cont1` et `cont2` sur le port `8081`, vous devrez démarrer `netcat` en mode écoute dans `cont2` et utiliser `telnet` pour vous connecter à ce port depuis `cont1`, en utilisant l'adresse appropriée pour atteindre `cont2` depuis `cont1`.

In cont1: Write “Bonjour”

In cont2: Receive “Bonjour”