Kubernetes

Para Quem Não Entende Nada

Aula 02

Volumes, NodePort, Ingress
e Comunicação Interna

Services

Faltou uma coisa na última aula, mostrar o describe do serviço e os endpoints.

Namespaces

Um namespace é um mecanismo para isolar logicamente grupos de recursos dentro de um cluster Kubernetes.

Namespaces

Existem vários namespaces dentro do kubernetes, podemos analisá-los com:

```
kubectl get namespaces #(ou ns)
```

Por padrão utilizamos o namespace “default” mas podemos modificar este comportamento, por exemplo:

```
Kubectl get pods -n kube-system
```

Namespaces

Criar um namespace é simples.
Todo comando suporta -n ou
--namespace:

```
$ kubectl create ns meu-ns
$ kubectl create deploy apache --image httpd -n meu-ns
$ kubectl get all -n meu-ns
```

Volumes

Todos container é efêmero por natureza. Seus dados serão perdidos se não estiverem em um volume externo caso o contêiner seja removido de alguma forma.
Um contêiner pode ter muitos volumes.

Volumes

O Kubernetes não fornece em sua instalação padrão uma forma de provisionar volumes dinamicamente. Felizmente, o Minikube sim.

Volumes

Existem vários tipos de volumes,
mas vamos analisar três deles:

- PersistentVolumeClaim
- hostPath
- emptyDir

PVCs e PVs

O PersistentVolumeClaim é o tipo mais utilizado, através dele fazemos um pedido para o Kubernetes que – em uma configuração dinâmica – nos entrega um PersistentVolume.

PersistentVolumeClaim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

PVs e PVCs

Para listar os PVCs e os PVs ambos podemos executar:

```
$ kubectl get pvcs
$ kubectl get pvs
```

PVs e PVCs

O PersistentVolume é global, não pertence a nenhum namespace. Já o PersistentVolumeClaim pertence, ele é o objeto que utilizamos em nossos pods.

garbage collector

Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
spec:
  ...
  template:
    spec:
      containers:
      - name: mysql
        volumeMounts:
        - mountPath: /var/lib/mysql
          name: mysql
      volumes:
      - name: mysql
        persistentVolumeClaim:
          claimName: mysql
```

Adicionar Volume

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
spec:
  ...
  template:
    spec:
      containers:
      - env:
        - name: MYSQL_ROOT_PASSWORD
          value: kub3rn3t3s
        - name: MYSQL_PASSWORD
          value: lua
        - name: MYSQL_USER
          value: lua
        - name: MYSQL_DATABASE
          value: lua
```

Adicionar Variáveis

emptyDir e hostPath

O emptyDir é um volume temporário utilizado para compartilhar dados entre contêineres de um mesmo pod.
O hostPath é um diretório na máquina compartilhado com os pods.

NodePort

Por padrão os serviços são criados com o tipo ClusterIP. Um serviço deste tipo pode ser acessado apenas dentro do cluster, para expor nossos serviços podemos utilizar o tipo NodePort.

Criar e expor Lua App

Utilizando o “expose” podemos criar um serviço do tipo NodePort:

```
$ kubectl create deploy lua --image \
docker.io/hectorvido/lua-app --port 8080

$ kubectl expose deploy/lua --type=NodePort --port=8080
```

Criar e expor Lua App

Um nodePort por padrão utiliza uma porta entre 30000-32767:

```
$ kubectl get svc lua

NAME   TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
lua    NodePort   10.97.181.68   <none>        8080:31168/TCP   3m10s
```

NodePort

O NodePort tem a facilidade de funcionar na instalação mais simples do Kubernetes, mas sua maior desvantagem é o desconhecimento de portas já utilizadas, causando confusão quando utilizado com frequência.

Ingress

O Ingress é uma forma de expor aplicações HTTP/HTTPS para fora do cluster.

Ingress

Existem vários Ingresses, normalmente fazemos a instalação destes como se fossem aplicações comuns, mas no caso do Minikube:

```
$ minikube addons enable ingress
```

Ingress

Valide se o Ingress foi habilitado com sucesso no namespace ingress-nginx:

```
$ kubectl get pods -n ingress-nginx

NAME                                        READY   STATUS      RESTARTS   AGE
ingress-nginx-admission-create-ftd4k        0/1     Completed   0          96s
ingress-nginx-admission-patch-dr7bb         0/1     Completed   0          96s
ingress-nginx-controller-7c6974c4d8-hskcd   1/1     Running     0          96s
```

Ingress

Vamos criar a primeira “url” da nossa aplicação em Lua:

```
$ kubectl create ingress lua-primeiro \
--rule=lua.example.local/*=lua:8080

$ minikube ip # 192.168.39.166

$ curl -v -H 'Host: lua.example.local' 192.168.39.166
```

Ingress

Vamos criar a segunda “url” da nossa aplicação em Lua:

```
$ kubectl create ingress lua \
--rule=lua.192-168-39-166.nip.io/*=lua:8080

$ curl -L lua.192-168-39-166.nip.io
```

O DNS Interno

Uma das facilidades do Kubernetes é a comunicação interna entre as aplicações. Devemos usar o nome dos Services ao invés de qualquer outro IP como o do próprio Service ou dos Pods, pois podem mudar.

O DNS Interno

Os pods se comunicam entre sí pelos IPs, mas a facilidade está em resolver o nome dos serviços:

<nome>.<ns>.svc.cluster.local

O DNS Interno

Podemos usar o nome completo ou apenas um pedaço dele:

```
kubectl expose deploy/mysql --port 3306

kubectl exec -ti deploy/lua --sh

nslookup mysql.default.svc.cluster.local
curl mysql.default.svc.cluster.local:3306
curl mysql:3306
```