02

Minikube

Transcrição

Começaremos a trabalhar com Kubernetes em nossa máquina local, com um projeto desenvolvido pelo pessoal do Kubernetes chamado ***Minikube***. Ao pesquisarmos este termo na internet, encontraremos [este link (http://kubernetes.io/docs/tasks/tools/install-minikube)](http://kubernetes.io/docs/tasks/tools/install-minikube) com as informações necessárias para sua instalação.

> Nos exercícios, será disponibilizado o passo a passo para a realização da instalação, a qual já foi feita na máquina local que está sendo utilizada no momento.

Após a instalação e inicialização do **Minikube** na máquina local, será criado um ambiente virtualizado (**VM**), em que haverá o *cluster*, a máquina Mestre - que estará recebendo as configurações do arquivo YAML - e a máquina *Node*, a receber a implementação dos containers que formam a aplicação.

A próxima etapa será criar estes arquivos de configuração para que a aplicação possa ser gerenciada pelo Kubernetes. Já que faremos alguns arquivos de configuração ao longo do curso, consolidaremos todos eles em um diretório na *Home*.

Vamos abrir o terminal e criar o diretório "kubernetes", em que consolidaremos todas as informações dos arquivos de configuração, digitando:

```
mkdir kubernetes
cd kubernetes
```

Seguiremos utilizando o Atom, e focaremos na parte web da nossa aplicação neste momento, portanto, teremos:

```
atom aplicacao.yaml
```

Com isto, o programa irá abrir e, na primeira linha do arquivo YAML colocaremos a versão da API do Kubernetes, responsável por criar este objeto que queremos no *cluster*, bem como seu tipo.

Como sabemos, há dois containers, porém, no Kubernetes não existe um objeto chamado `Container`. O menor objeto existente no Kubernetes recebe o nome de "***Pod***". Mas o que seria isto?

> Não se preocupe em decorar estas informações, elas estão contidas na documentação do Kubernetes.

```
apiVersion: v1
kind: Pod
```

Pods e containers

Temos trabalhado com containers em nossa aplicação: um para a parte web e outro para o banco de dados. No Kubernetes, porém, precisamos abstrair este container colocando-o no objeto denominado `Pod`, cujo tamanho é maior do que o de um container.

Isto é, **o `Pod` suporta um ou mais containers**, então, em um mesmo Pod, é possível ter um para banco de dados e outro para fazer a análise de um log, por exemplo.

Embora esta configuração seja possível, quando inserimos mais de um container em um mesmo objeto `Pod`, é gerado um **alto grau de acoplamento entre os containers, pois eles estarão dividindo memória e volume**.

Na nossa aplicação, criaremos um objeto destes para a web, e outro `Pod` para o banco de dados. Vamos voltar ao arquivo `aplicacao.yaml` e focar nossa atenção para a criação da parte web.

Tendo especificada a criação de um objeto de tipo `Pod`, há a possibilidade do Kubernetes estar gerenciando vários outros objetos de mesmo tipo. Teremos que dar um nome para identificá-lo.

Todas estas informações ficarão armazenadas em um campo chamado "metadata", nomeando-se com `aplicacao` e especificando-se como conteúdo `containers` no plural (pois pode ser mais de um), e a imagem base será a que já está no repositório. Diremos que a porta `80` deverá ser acessada. O arquivo ficará da seguinte maneira:

```
apiVersion: v1
kind: Pod
metadata:
  name: aplicacao
spec:
  containers:
    - name: container-aplicacao-loja
      image: rafanercessian/aplicacao-loja:v1
      ports:
        - containerPort: 80
```

Feito isto, queremos que seja criado no *cluster* o `Pod`, que abstrai nosso container da aplicação web. Voltaremos ao terminal e inicializaremos o minikube para isso a partir do comando `minikube start`.

O minikube cria uma máquina virtual, e teremos o *cluster* no ambiente virtualizado. Em seguida, precisaremos nos comunicar com o *cluster* para que então seja criado o objeto `Pod`.

Para estabelecermos esta comunicação, usaremos a linha de comando com `kubectl`, cujos passos para a instalação, assim como o `minikube`, deixarei disponíveis mais adiante. **Utilizaremos `kubectl` sempre que quisermos nos comunicar com o *cluster***.

Como queremos que seja criado o `Pod`, especificado em nosso arquivo `aplicacao.yaml`, digitaremos:

```
kubectl create -f aplicacao.yaml
```

Em seguida, apertaremos "Enter" e obteremos a informação de que no *cluster* inicializado pelo Kubernetes foi criado um `pod` chamado "aplicacao". Agora vamos perguntar ao *cluster* o que está sendo rodado nele, para confirmar se o `pod` está realmente sendo executado:

```
kubectl get pods
```

Veremos que o `pod` denominado "aplicacao" - nome que havíamos dado no campo "metadata" -, está com status "*Running*", isto é, está rodando perfeitamente.

Como já vimos antes, esperamos que o Kubernetes esteja gerenciando o objeto `Pod`. Significa que se por ventura o `Pod` parar de funcionar por algum motivo, esperamos que o Kubernetes perceba o estado do *cluster*, que estará diferente do que havíamos configurado inicialmente em `aplicacao.yaml`, e encontre alguma maneira de criar um novo `pod`, um container web.

Vamos ver isto em funcionamento?

Para simularmos um problema, removeremos o `Pod` do *cluster* e verificaremos a ação tomada pelo Kubernetes depois:

```
kubectl delete pods aplicacao
```

Com isso, informamos o *cluster* para que se delete, dentre os `pods` existentes, aquele cujo nome é `aplicacao`. Com "Enter", obtém-se a mensagem de que a remoção foi realizada com sucesso.

Usando o comando abaixo, pediremos que o *cluster* nos informe quais `pods` estão sendo rodados:

```
kubectl get pods
```

A mensagem retornada é a seguinte:

```
No resources found.
```

O que indica que nenhum objeto do tipo `pod` foi encontrado, ou seja, este objeto foi removido permanentemente! Poxa, por que o Kubernetes não está fazendo o que deveria? Veremos na sequência, vamos lá!