02

StatefulSet

Transcrição

Atenção: Vamos utilizar o Mysql na versão 5.7.19 que é compatível com a aplicação web. A definição da imagem fica: `image: mysql:5.7.19`

Conseguimos acessar a aplicação web, porém são exibidas mensagens de erro também, porque nossa aplicação está tentando se comunicar com o banco de dados ainda não configurado.

Dividimos a aplicação em dois containers: um com a parte web e outro para o banco de dados. Como vimos anteriormente, no Kubernetes não existe um objeto denominado `Container`.

O menor objeto existente é o `Pod`, para o qual teremos que abstrair este container do banco de dados. É necessário um novo arquivo YAML para criá-lo em nosso *cluster*!

Portanto, voltaremos ao terminal, em que digitaremos `cd kubernetes` para acessarmos o `kubernetes`, e depois solicitaremos a listagem dos arquivos existentes com o comando `ls`. Obteremos:

```
aplicacao.yaml   deployment.yaml   servico-aplicacao.yaml
```

Tais arquivos se referem à aplicação web e, neste momento, vamos trabalhar com o banco de dados. Para não ficarmos misturando os arquivos, criaremos um novo diretório em `kubernetes`, chamado "app", para referenciar os arquivos que colocaremos ali, aproveitando para mover todos os `.yaml` para dentro dele.

Feito isto, criaremos outro diretório para o banco de dados (`db`), em que consolidaremos todas as informações acerca de sua configuração. Depois, chamaremos o Atom e criaremos o novo arquivo:

```
mkdir app && mv *.yaml app
mkdir db
cd db/
atom pod-banco.yaml
```

Já estamos mais acostumados a criar este arquivo, não é mesmo?

Vamos relembrar que, acessando "Área de trabalho > Projeto > loja > conecta.php", temos "db", "root", "", "loja" (ou seja, banco de dados, usuário `root`, senha vazia e o banco `loja`) sendo utilizados pela nossa aplicação web para a realização da persistência, do cadastro dos produtos no banco de dados.

Assim como fizemos no arquivo `docker-compose.yaml`, passaremos estas informações como variáveis de ambiente (`env`, de "*environment*", em inglês), de modo que teremos:

```
apiVersion: v1
kind: Pod
metadata:
  name: mysql
```

```
spec:
  containers:
    - name: container-mysql
      image: mysql:5.7.19
      ports:
        - containerPort: 3306
      env:
        - name: MYSQL_DATABASE
          value: "loja"
        - name: MYSQL_USER
          value: "root"
        - name: MYSQL_ALLOW_EMPTY_PASSWORD
          value: "1"
```

Acabamos de configurar o `Pod` que irá abstrair o container do MySQL!

Sabendo que o `Pod` é o objeto mais básico do Kubernetes, sem oferecer muitos recursos, o ideal é que façamos a abstração deste objeto em outro que, além destes recursos, adicione uma camada de estado desejado da nossa aplicação, para que o Kubernetes sempre saiba que queremos um `Pod` voltado ao banco de dados rodando no *cluster*.

Até o momento, tínhamos o container do banco de dados, que foi abstraído pelo `Pod`, que por sua vez precisará ser abstraído em outro. Diferentemente do que fizemos na parte de aplicação web, no entanto, aqui estamos preocupados com as informações que serão armazenadas no banco, pois não queremos perdê-las caso haja algum problema.

Temos que encontrar uma forma de fazermos uma espécie de mapeamento de volumes. Para isso, existe um outro objeto do Kubernetes, mais semântico, para justamente trabalhar nestes cenários em que nos preocupamos com as informações a serem persistidas em nosso banco.

Trata-se do ***StatefulSet***, o qual criaremos para a abstração do `Pod`, e que oferecerá recursos adicionais indicando ao Kubernetes sobre o estado desejado da aplicação. Assim, conseguiremos trabalhar com esta questão de volumes, mapeando-os no container do banco de dados para outro lugar externo.

Para isto, criaremos outro arquivo de configuração YAML. No Atom, usaremos o atalho "Ctrl + N" para abri-lo, aproveitando para salvá-lo em "db" com o nome "statefulset.yaml". Nele, digitaremos:

```
apiVersion: apps/v1beta1
kind: StatefulSet
metadata:
  name: statefulset-mysql
spec:
  template:
```

A partir daí, teremos toda a informação do `Pod` a ser abstraído pelo `StatefulSet` - algo que já fizemos! Toda a configuração se encontra em `pod-banco.yaml`. Vamos aproveitá-lo copiando tudo que vem após `metadata` e colando neste novo arquivo, não esquecendo de acrescentar `labels:` antes de `name`, e ajustando depois a indentação. Isso ficará da seguinte maneira:

```
apiVersion: apps/v1beta1
kind: StatefulSet
metadata:
  name: statefulset-mysql
spec:
```

```
  template:
    metadata:
      labels:
       name: mysql
    spec:
      containers:
        - name: container-mysql
          image: mysql:5.7.19
          ports:
            - containerPort: 3306
          env:
            - name: MYSQL_DATABASE
              value: "loja"
            - name: MYSQL_USER
              value: "root"
            - name: MYSQL_ALLOW_EMPTY_PASSWORD
              value: "1"
```

Com isso, criamos o objeto `StatefulSet` ! Trabalharemos a seguir com o mapeamento de volumes para o caso do `Pod` deixar de funcionar, impedindo a perda de informações guardadas no banco de dados.