11

Criando serviço banco

Transcrição

Construímos o objeto `StatefulSet` que irá realizar a abstração do `Pod`, dentro do qual está o container do banco de dados.

Já sabemos que é possível escalonarmos a aplicação, aumentando ou diminuindo a quantidade de *pods*, e por esse motivo não conseguimos acessar um `Pod` diretamente, sendo necessário abstrairmos seu acesso em uma camada mais estabilizada, no caso o objeto `Service`.

Neste momento, ele será utilizado pela aplicação web para a realização do cadastro dos produtos da Alura Sports, ou seja, para a persistência de dados no banco.

Para criarmos este objeto, geraremos mais uma vez um arquivo YAML de configuração. No Atom, abriremos uma nova aba e salvaremos o arquivo em "db" com o nome "servico-banco.yaml".

O tipo de atuação do `Service` agora será como interface de comunicação entre a aplicação web e o container com o banco de dados, que estarão no *cluster*.

Não há necessidade de um recurso de acesso externo no `Service`, pois esta comunicação ocorrerá dentro do próprio *cluster*. Por este motivo, podemos configurar o `Service` para que seja acessado apenas por quem se encontra ali, de modo que usaremos `ClusterIP` como tipo.

O `Service` faz a abstração do `Pod` com o banco MySQL, então teremos que indicar a porta de acesso do MySQL para a realização da persistência dos dados (`3306`).

E não podemos nos esquecer de estabelecer o vínculo entre `Service` e o *pod* a ser abstraído, isto é, usar a chave seletora. O objeto `Service` estará abstraindo o acesso do `Pod` que, por sua vez, é abstraído pelo `StatefulSet`, identificado com a etiqueta `label` de nome `mysql`. Assim:

```
apiVersion: v1
kind: Service
metadata:
  name: db
spec:
  type: ClusterIP
  ports:
    - port: 3306
  selector:
    name: mysql
```

Salvaremos o arquivo! Para o objeto `StatefulSet` funcionar, ele será um tanto diferente do que foi feito com o objeto `Deployment`, já que o primeiro exige a referência bidirecional. É preciso indicar a ele o serviço responsável pela abstração do objeto `Pod`.

Para isso, em `statefulset.yaml`, logo abaixo de `spec`, acresentaremos `serviceName`, que será o nome do serviço colocado em `servico-banco.yaml`, ou seja, `db`:

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

Vamos salvar o arquivo e, com isso, basicamente terminamos a configuração do banco de dados! Agora, é necessário que estes objetos sejam levados ao nosso *cluster*. Voltaremos ao terminal e digitaremos `ls` para obtermos a listagem dos arquivos, que será:

```
permissoes.yaml   pod-banco.yaml   servico-banco.yaml   statefulset.yaml
```

O primeiro passo consiste em termos o `Pod` com o banco de dados do MySQL, informação abstraída pelo `StatefulSet`. Vamos solicitar a criação deste objeto no *cluster*, que já estará abstraindo o `Pod` do MySQL, através do qual teremos a informação de que precisamos.

Usaremos o comando `kubectl create -f statefulset.yaml`, e recebemos o seguinte retorno:

```
statefulset "statefulset-mysql" created
```

O que indica que o objeto foi criado com sucesso! Lembrando que a linguagem YAML é bem complicada: se erramos a indentação, o arquivo de configuração não é aceito.

Podemos criar o serviço que irá abstrair o acesso a este `Pod`, o que faremos a partir do comando `kubectl create -f servico-banco.yaml`. Obteremos a informação de que `db` foi criado.

Falta criarmos as permissões deste *pod* para o acesso do volume, garantindo que nossas informações não serão perdidas. Usaremos `kubectl create -f permissoes.yaml` para isto. Maravilha, aparentemente tudo está rodando no *cluster* conforme esperado.

No entanto, não podemos esquecer que, para rodarmos a aplicação, é necessário criarmos algumas tabelas no MySQL. Para tal, acessaremos o `Pod` que está abstraindo o MySQL.

Antes disso, verificaremos com o *cluster* quais os *pods* em execução. Esperamos que haja `4` deles, sendo `3` na aplicação web e `1` com o banco de dados.

```
kubectl get pods
```

De fato, temos `4` *pods*: `aplicacao-deployment-17598552-69q7g`, `aplicacao-deployment-17598552-74tm4`, `aplicacao-deployment-17598552-87q94` e `statefulset-mysql-0` - este último normalmente leva mais tempo para ser criado.

Iremos acessá-lo e criar as tabelas para fazermos o teste com nossa aplicação. O comando será bem parecido com aquele feito com o Docker, com a diferença de estarmos nos comunicando com o *cluster*. Digitaremos:

```
kubectl exec -it statefulset-mysql-0 sh
```

Com "Enter", iremos à linha de baixo, em que colocaremos:

```
mysql -u root
```

Feito isto, usaremos o banco `loja` com o comando `use loja` , já configurada na variável de ambiente, e criaremos as tabelas cujos arquivos SQL foram enviados para nós pelos diretores da Alura Sports. Vamos copiá-los e colar no terminal.

Legal! Se tudo correu bem, devemos ter a aplicação funcionando sem problemas ou mensagens de erro. Vamos confirmar isto digitando `exit` e pedindo ao `minikube` para que seja fornecido o endereço IP (a URL) vinculado ao primeiro objeto de serviço, que está abstraindo o acesso à parte web.

```
minikube service servico-aplicacao --url
```

Obteremos a URL `http://192.168.99.100:31503` , que copiaremos e colaremos na barra de endereços do *browser*. Parece que tudo está funcionando bem, vamos testar cadastrando alguns produtos.

O primeiro será "Camiseta seleção brasileira" no campo "Nome", "200" em "Preço", "Camiseta verde e amarela" em "Descrição". Em seguida, cadastraremos "Raqueteira", "300" e "Raqueteira marca Alura" na categoria "Tenis".

Maravilha, as informações estão sendo cadastradas! Agora, esperamos que o Kubernetes esteja gerenciando a aplicação. Utilizando o comando `kubectl get pods` , veremos que há três *pods* com a aplicação web rodando, dentre os quais não fazemos ideia, qual estamos acessando. O objeto de serviço é quem faz este balanceamento para nós.

O Kubernetes sabe qual o estado desejado da aplicação, então, caso deletemos o primeiro *pod* com o comando `kubectl delete pods aplicacao-deployment-17598552-69q7g` , ele perceberá que queremos sempre ter três *pods*, e não dois, portanto ele criará um novo `Pod` .

Usando o comando `kubectl get pods` novamente, observa-se que foi acrescentado à listagem `aplicacao-deployment-17598552-bz07z` , antes inexistente. Se perdermos o `StatefulSet` , o que acontecerá? Por conta do mapeamento de volumes que fizemos, não perderemos as informações previamente armazenadas.

Vamos confirmar isto digitando `kubectl delete pods statefulset-mysql-0` e, depois, `kubectl get pods` para verificarmos o que houve. O objeto `statefulset-mysql-0` permanece na listagem! A questão é sabermos se as informações cadastradas foram perdidas ou não.

Voltaremos à aplicação e clicaremos em "Produtos" no menu superior da página. Veremos que aqueles cadastrados anteriormente continuam ali, ou seja, mesmo removendo o `Pod` que estava abstraindo o container do MySQL, devido o mapeamento de volumes, tudo se mantém como se nada tivesse acontecido.

Vamos cadastrar outro produto: "Camiseta seleção brasileira" com preço "150" e "Camiseta azul" em "Descrição". Tudo está como gostaríamos, as informações se encontram ali, e a remoção de produtos também funciona corretamente.