09

Deploy da Aplicação

Transcrição

Feita a configuração do *cluster* no Google Cloud, falta passarmos a ele os arquivos de configuração YAML. Todos os arquivos criados até então foram testados em nosso ambiente local, de desenvolvimento, a partir do `minikube`. Agora, precisaremos apenas fazer alguns ajustes para enviá-los ao ambiente de produção.

Para não alterarmos o que já foi feito, criaremos outro diretório, denominado `prod` (para remeter à "produção") para onde passaremos estes arquivos e em que faremos as devidas alterações. Vamos digitar no terminal:

```
mkdir prod && cp -r app prod && cp -r db prod
```

Com o comando `ls`, serão listados os diretórios `app` e `db`, utilizados no ambiente de desenvolvimento para testes com o `minikube`, e o recém criado `prod`, cujo `db` acessaremos da seguinte forma:

```
cd prod/
cd db/
```

Usando `ls` mais uma vez, teremos todos os arquivos referentes ao diretório `db`, localizados no ambiente de produção, que serão levados ao Google Cloud.

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

Para realizarmos a integração com o Google Cloud, utilizaremos a versão `5.5` do MySQL, informação existente no container que está sendo abstraído pelo `Pod`, o qual por sua vez está sendo abstraído pelo `StatefulSet`, que acessaremos para especificar esta versão. Abriremos o Atom digitando `atom statefulset.yaml` no terminal.

Neste arquivo, alteraremos a linha `image: mysql` para `image: mysql:5.5`, e depois o salvaremos. Esta é a única alteração que precisaremos fazer no nosso projeto. E, no terminal, usaremos comandos similares àqueles utilizados com o `minikube`:

```
kubectl create -f statefulset.yaml
```

Obteremos a informação de que este objeto foi criado no *cluster* com sucesso. Em seguida, criaremos o objeto de serviço, para a abstração do acesso a este `Pod` com MySQL:

```
kubectl create -f servico-banco.yaml
```

O objeto de serviço `db` foi criado! É necessário configurarmos, também, o objeto das permissões de acesso ao volume persistente:

```
kubectl create -f permissoest.yaml
```

Receberemos a informação de que o `persistentvolumeclaim` foi criado. Passada a configuração do banco de dados ao *cluster* do Google Cloud, voltaremos ao diretório `app`, em que faremos os mesmos procedimentos:

```
cd ../app
```

Com `ls` listamos os objetos ali presentes:

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

Passaremos o objeto `Pod` que está abstraindo o container da aplicação web, que por sua vez está sendo abstraído pelo `deployment`:

```
kubectl create -f deployment.yaml
```

Assim como o banco de dados, para acessarmos o `Pod` com a aplicação web, é necessário o `servico-aplicacao.yaml` para a abstração do acesso. Digitaremos, portanto:

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

Vamos garantir que tudo isso foi realmente criado no *cluster*? Voltaremos ao painel do Google Cloud e clicaremos em "*Workloads*" no menu lateral esquerdo, o que nos mostrará os objetos criados por nós. Encontram-se ali `aplicacao-deployment` e `statefulset-mysql`, ambos com status "OK".

Atualmente, o `aplicacao-deployment`, que abstrai o `Pod` com a aplicação web, possui apenas um *pod*. Faremos uma escalonação para termos três deles, nos aproximando do cenário testado no `minikube`, em que tínhamos três *pods* com a aplicação web e um com MySQL.

No terminal, usaremos o comando `kubectl scale deployment aplicacao-deployment --replicas=3`, e com "Enter" teremos a informação de que a aplicação foi escalonada. Confirmaremos isto voltando ao Google Cloud e atualizando a página referente a "*Workloads*".

Legal, agora `aplicacao-deployment` aparece como contendo `3/3 Pods`. Se formos à linha de comando no terminal e digitarmos `kubectl get pods`, já teremos os quatro *pods* rodando como gostaríamos.

Falta criarmos as tabelas neste MySQL para testarmos a aplicação. Acessaremos o `Pod` do banco de dados:

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

Acessaremos então o MySQL com usuário "root" a partir de:

```
mysql -u root
```

Devemos fazer a configuração do banco `loja`, criado anteriormente, com `use loja`, e então acrescentaremos os comandos SQL que os diretores da Alura Sports nos passaram previamente, que copiaremos e colaremos no terminal.

Com isto, toda a configuração foi feita! Precisaremos testar a aplicação no Google Cloud. Digitaremos `exit` e acessaremos a aplicação, `app`, que está sendo abstraída pelo objeto de serviço, isto é, é preciso saber o endereço IP que o Google forneceu a ele.

Para tal, usaremos `kubectl get service`, e são retornadas informações como o "CLUSTER-IP", endereço de acesso interno no *cluster* de cada objeto listado, bem como o endereço externo, público, que indica que `35.198.57.152` refere-se ao `servico-aplicacao`. Copiaremos este IP, abriremos um *browser* e testaremos a aplicação colando-o na barra de endereço.

A nossa aplicação está no ar, rodando no ambiente de produção, ou seja, no Google Cloud! Qualquer pessoa que fizer o mesmo, quer dizer, colar este IP na barra de endereço do *browser*, conseguirá o acesso.

Testaremos o funcionamento da aplicação cadastrando alguns produtos: "Camiseta seleção brasileira", com "200" e "Camiseta verde e amarela", e "Raqueteira", com "150", "Raqueteira marca Alura" e categoria "Tenis". Ambos foram cadastrados sem problemas - vamos alterar o preço do primeiro, de "200" para "250", o que conseguimos também sem problemas!

Isto quer dizer que toda a aplicação está sendo bem gerenciada pelo Kubernetes. Temos, ao todo, três *pods* em execução na aplicação web, cujo `Pod` utilizado não nos importa (como usuários finais), desde que o trabalho seja bem feito e o balanceamento de cargas esteja ocorrendo corretamente.

O Kubernetes sabe qual é o estado desejado da aplicação, então se por um acaso ocorrer algum problema com um deles, o que poderemos simular deletando um dos *pods* a partir de `kubectl delete pods aplicacao-deployment-4219871081-22gr0`, teremos que o estado desejado foi alterado.

Deste modo, como sabemos, o Kubernetes criará um novo *pod* para substitui-lo, o que pode ser visto ao digitarmos `kubectl get pods`. Vamos, então, simular um problema com o `Pod` do MySQL, e analisar se as informações serão de fato mantidas:

```
kubectl delete pods statefulset-mysql-0
```

Feito isto, usaremos `kubectl get pods` e veremos em "STATUS" que o último *pod* ainda está sendo criado ("*ContainerCreating*"). Assim que isso ocorre, por conta do mapeamento de volumes, esperamos que as nossas informações tenham sido mantidas.

Voltando à aplicação no *browser* e clicando em "Produtos" na aba superior, observamos que tudo se mantém e nada foi perdido. Vamos tentar inserir outro produto! "Camiseta seleção italiana", com "200" e "Camiseta azul" é cadastrado com sucesso! Desta forma, há três produtos listados, conforme esperado.

Podemos experimentar deletando produtos, o que também ocorre sem problemas. **Atenção!!** Após o término do exercício no Google Cloud, é importante voltarmos ao painel geral e clicar em "*Container clusters*" no menu lateral para **removermos o cluster**, marcando seu *checkbox* e apertando a opção "*DELETE*" na parte superior.

Isto porque estamos consumindo as máquinas no Google Cloud graças à bonificação que recebemos quando nos cadastramos, e cujo crédito está sendo consumido continuamente. Para não o gastarmos desnecessariamente, recomendo fortemente que depois do exercício seja feita a remoção do *cluster*.