18 – Testando Escalabilidade no Kubernetes – Horizonal Pod Autoscaler

Os objetivos dessa prática são:

- Instalar o Metric Server;
- Fazer o deploy de uma aplicação básica;
- Fazer o deploy de um Horizontal Auto Scaler;
- Estressar a CPU do pod com a aplicação básica;
- Monitorar o HPA e o comportamento do Kubernetes.

Para realizar essa prática é preciso ter instalado um cluster Kubernetes.

Passo a Passo

Para configurar o Horizontal Auto Scaler é preciso, primeiro, fazer o deploy do servidor que faz a coleta das métricas dos Nodes e dos Pods. Para isso:

1) No terminal onde está o seu cluster do Kubernetes, digite o comando “kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml

```
% kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

2) Digite o comando “kubectl get pods -A” e verifique se o POD “metrics-server” está em “running”:

```
marceloortiz@MacBook-Air-de-Marcelo-2 hack % kubectl get pods -A
NAMESPACE            NAME                                       READY   STATUS    RESTARTS      AGE
kube-system          coredns-6d4b75cb6d-blprk                   1/1     Running   2 (12m ago)   2d5h
kube-system          coredns-6d4b75cb6d-tnrtg                   1/1     Running   2 (12m ago)   2d5h
kube-system          etcd-ortiz-control-plane                   1/1     Running   2 (12m ago)   2d5h
kube-system          kindnet-zxpm5                              1/1     Running   2 (12m ago)   2d5h
kube-system          kube-apiserver-ortiz-control-plane         1/1     Running   2 (12m ago)   2d5h
kube-system          kube-controller-manager-ortiz-control-plane   1/1     Running   2 (12m ago)   2d5h
kube-system          kube-proxy-nlm69                           1/1     Running   2 (12m ago)   2d5h
kube-system          kube-scheduler-ortiz-control-plane         1/1     Running   2 (12m ago)   2d5h
kube-system          metrics-server-678f4bf65b-4wg8p            0/1     Running   0             30s
local-path-storage   local-path-provisioner-9cd9bd544-wxr54     1/1     Running   2 (12m ago)   2d5h
```

3) Verifique os eventos do seu pod “metrics-server”, caso exista alguma falha com “HTTP probe failed with statuscode: 500” siga o passo 4, caso contrário siga para o passo 6:

   Para verificar os eventos do pod, digite: kubectl describe pods -n kube-system “nome do pod de métricas”:

```
kubectl describe pods -n kube-system metrics-server-678f4bf65b-4wg8p
```

   O evento com erro é:

```
Events:
  Type     Reason     Age                  From               Message
  ----     ------     ----                 ----               -------
  Normal   Scheduled  5m35s                default-scheduler  Successfully assigned kube-system/metrics-server-678f4bf65b-4wg8p to ortiz-control-plane
  Normal   Pulled     5m34s                kubelet            Container image "k8s.gcr.io/metrics-server/metrics-server:v0.6.1" already present on machine
  Normal   Created    5m34s                kubelet            Created container metrics-server
  Normal   Started    5m34s                kubelet            Started container metrics-server
  Warning  Unhealthy  25s (x32 over 5m5s)  kubelet            Readiness probe failed: HTTP probe failed with statuscode: 500
```

4) Para corrigir esse erro, será preciso editar o POD “metrics-server” e adicionar as linhas abaixo na especificação do containers:

   command:
   - /metrics-server
   - --kubelet-insecure-tls
   - --kubelet-preferred-address-types=InternalIP

   Para isso digite “kubectl edit deploy -n kube-system metrics-server:

```
kubectl edit deploy -n kube-system metrics-server
```

   Pressione a tecla “i” e insira as linhas acima logo abaixo da linha “- --metric-resolution=15s”

```
  spec:
    containers:
    - args:
      - --cert-dir=/tmp
      - --secure-port=4443
      - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
      - --kubelet-use-node-status-port
      - --metric-resolution=15s
      command:
      - /metrics-server
      - --kubelet-insecure-tls
      - --kubelet-preferred-address-types=InternalIP
```

   Cuidado com a indentação, pois as novas linhas precisam iniciar na mesma coluna das demais linhas.

   Saia salvando, pressionando a tecla “esc” e depois digitando “:wq”.

5) Verifique se agora o POD do “metrics-server” está em execução sem nenhum erro nos últimos eventos, para isso digite “kubectl describe pods -n kube-system “nome do pod do metric server”:

```
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  103s  default-scheduler  Successfully assigned kube-system/metrics-server-69cb89695d-t5ll5 to ortiz-control-plane
  Normal  Pulled     103s  kubelet            Container image "k8s.gcr.io/metrics-server/metrics-server:v0.6.1" already present on machine
  Normal  Created    103s  kubelet            Created container metrics-server
  Normal  Started    103s  kubelet            Started container metrics-server
```

6) Veja que agora é possível executar os comandos “kubectl top pods -A” e “kubectl top nodes”. Esse comando vai trazer a utilização de CPU e memória dos seus PODs e Nodes.

```
marceloortiz@MacBook-Air-de-Marcelo-2 hack % kubectl top nodes
NAME                 CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
ortiz-control-plane   240m         6%     747Mi           18%
```

```
marceloortiz@MacBook-Air-de-Marcelo-2 hack % kubectl top pods -A
NAMESPACE            NAME                                          CPU(cores)   MEMORY(bytes)
kube-system          coredns-6d4b75cb6d-blprk                      7m           12Mi
kube-system          coredns-6d4b75cb6d-tnrtg                      6m           12Mi
kube-system          etcd-ortiz-control-plane                      30m          48Mi
kube-system          kindnet-zxpm5                                 2m           10Mi
kube-system          kube-apiserver-ortiz-control-plane            82m          319Mi
kube-system          kube-controller-manager-ortiz-control-plane   31m          42Mi
kube-system          kube-proxy-nlm69                              1m           10Mi
kube-system          kube-scheduler-ortiz-control-plane            6m           16Mi
kube-system          metrics-server-69cb89695d-t5ll5               10m          18Mi
local-path-storage   local-path-provisioner-9cd9bd544-wxr54        3m           6Mi
```

Agora já podemos fazer o deploy de uma aplicação qualquer. No nosso caso vamos fazer um deploy de um nginx. Essa aplicação que vamos fazer a configuração do HPA.

7) Crie um arquivo com o nome “deploy.yaml” com as informações abaixo:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: scaling-deploy
spec:
 replicas: 1
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
     image: nginx
     resources:
       requests:
         memory: "50Mi"
         cpu: "500m"
```

8) Faça o deploy desse arquivo com o comando “kubectl apply -f deploy.yaml”:

```
marceloortiz@MacBook-Air-de-Marcelo-2 k8s % kubectl apply -f deploy.yaml
deployment.apps/scaling-deploy created
```

9) Verifique o consumo desse POD com o comando “kubectl top pods”:

```
marceloortiz@MacBook-Air-de-Marcelo-2 k8s % kubectl top pods
NAME                              CPU(cores)   MEMORY(bytes)
scaling-deploy-8f458dc5b-4z699    0m           6Mi
```

Veja que o consumo de CPU está bem baixo.

Agora vamos criar nosso HPA com base na utilização da CPU, para isso:

10) Crie um arquivo chamado “hpa.yaml” com o conteúdo abaixo:

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
 name: scaling-deploy
spec:
 maxReplicas: 10
 minReplicas: 1
 scaleTargetRef:
  apiVersion: apps/v1
  kind: Deployment
  name: scaling-deploy
 targetCPUUtilizationPercentage: 70
```

11) Faça o deploy do HPA com o comando “kubectl apply -f hpa.yaml”:

```
marceloortiz@MacBook-Air-de-Marcelo-2 k8s % kubectl apply -f hpa.yaml
horizontalpodautoscaler.autoscaling/scaling-deploy created
```

12) Verifique se o HPA foi criado corretamente com o comando “kubectl get hpa”:

```
marceloortiz@MacBook-Air-de-Marcelo-2 k8s % kubectl get hpa
NAME             REFERENCE                   TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
scaling-deploy   Deployment/scaling-deploy   1%/70%    1         10        1          2m55s
```

Agora vamos simular a carga no nosso POD e verificar o comportamento do HPA. Para isso:

13) Conecte no seu POD com o comando “kubectl exec -it “nome do pod” -- bash":

```
% kubectl exec -it scaling-deploy-6686b77c59-9mj9s -- bash
```

14) Dentro do seu POD, execute o comando “apt update”:

```
root@scaling-deploy-6686b77c59-9mj9s:/# apt update
Get:1 http://deb.debian.org/debian bullseye InRelease [116 kB]
Get:2 http://deb.debian.org/debian-security bullseye-security InRelease [44.1 kB]
Get:3 http://deb.debian.org/debian bullseye-updates InRelease [44.1 kB]
Get:4 http://deb.debian.org/debian bullseye/main arm64 Packages [8069 kB]
Get:5 http://deb.debian.org/debian-security bullseye-security/main arm64 Packages [165 kB]
Get:6 http://deb.debian.org/debian bullseye-updates/main arm64 Packages [2600 B]
Fetched 8441 kB in 4s (2045 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
```

15) Dentro do seu POD, execute o comando “apt install stress -y”:

```
root@scaling-deploy-6686b77c59-9mj9s:/# apt install stress -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  stress
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 21.6 kB of archives.
After this operation, 54.3 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian bullseye/main arm64 stress arm64 1.0.4-7 [21.6 kB]
Fetched 21.6 kB in 3s (7725 B/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package stress.
(Reading database ... 7815 files and directories currently installed.)
Preparing to unpack .../stress_1.0.4-7_arm64.deb ...
Unpacking stress (1.0.4-7) ...
Setting up stress (1.0.4-7) ...
```

16) Dentro do seu POD, execute o comando “stress --cpu 1”:

```
root@scaling-deploy-6686b77c59-9mj9s:/# stress --cpu 1
stress: info: [318] dispatching hogs: 1 cpu, 0 io, 0 vm, 0 hdd
```

17) Abra um outro terminal no seu cluster Kubernetes e verifique o consumo do POD com o comando “kubectl top pods”:

```
MacBook-Air-de-Marcelo-2:k8s marceloortiz$ kubectl top pod
NAME                                CPU(cores)   MEMORY(bytes)
scaling-deploy-6686b77c59-694ls   6m           3Mi
scaling-deploy-6686b77c59-9mj9s   999m         32Mi
```

Veja que o POD está consumindo 999m de CPU e o HPA começou a criar novos PODs.

18) Verifique quantos nos PODs foram criados automaticamente com o comando “kubectl get pods”:

```
MacBook-Air-de-Marcelo-2:k8s marceloortiz$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
scaling-deploy-6686b77c59-694ls   1/1     Running   0          3m29s
scaling-deploy-6686b77c59-9mj9s   1/1     Running   0          10m
scaling-deploy-6686b77c59-pjxkb   1/1     Running   0          3m29s
```

No caso dos nossos testes foram criados 2 novos PODs.

19) Interrompa o “stress” que está sendo executado no primeiro pod digitando as teclas “crtl + c”. Aguarde até que o HPA verifique que o consumo voltou abaixo dos 70% de CPU e observe se os PODs que foram criados durante o pico de consumo foram destruídos.

20) Para limpar o ambiente:
    - Remover o HPA: kubectl delete -f hpa.yaml
    - Remover o nginx: kubectl delete -f deploy.yaml

- Remover o servidor de métricas: kubectl delete -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml