Horizontal Pod Autoscaler (HPA) no Kubernetes com Minikube

PSI5120 - Tópicos em Computação em Nuvem
Nome: Matheus Fortunato Alves
No USP: 4484262

Parte 1: Introdução e pré-requisitos

O que é o HPA?

O Horizontal Pod Autoscaler (HPA) ajusta automaticamente a quantidade de réplicas de um workload resource (como um Deployment ou StatefulSet) de acordo com a demanda de utilização de recursos (CPU, memória, etc).

- **Escalonamento horizontal:** aumenta/diminui a quantidade de Pods rodando.
- **Escalonamento vertical:** aumenta/diminui os recursos (CPU/memória) de cada Pod já existente.

O HPA garante que a aplicação responda automaticamente a mudanças de carga, aumentando os Pods quando a demanda cresce e reduzindo quando ela diminui.

Pré-requisitos do laboratório

Antes de começar, você precisa de:

- Docker Desktop instalado e configurado
- Kubernetes Cluster (usaremos o Minikube).
- kubectl configurado para se conectar ao cluster.
- Dois ou mais nodes no cluster (recomendado).
- Um Metric Server instalado e configurado.

Passo 1 – Instalar o Minikube

No Windows (PowerShell, com administrador):

```
choco install minikube kubernetes-cli
```

```
PS C:\Users\Fortu> minikube version
minikube version: v1.36.0
commit: f8f52f5de11fc6ad8244afac475e1d0f96841df1-dirty
```

Passo 2 – Iniciar o cluster Minikube

Crie o cluster Kubernetes local com pelo menos 2 nós que não sejam de controle:

```
minikube start --nodes 2 -p HPA-kubernetes --driver=docker
```

```
PS C:\Users\Fortu> minikube start --nodes 2 -p HPA-kubernetes --driver=docker
* [HPA-kubernetes] minikube v1.36.0 on Microsoft Windows 10 Home Single Language 10.0.19045.6093 Build 19045.6093
* Using the docker driver based on user configuration
* Using Docker Desktop driver with root privileges
* Starting "HPA-kubernetes" primary control-plane node in "HPA-kubernetes" cluster
* Pulling base image v0.0.47 ...
```

Verifique se o cluster está rodando:

```
kubectl get nodes
```

```
S C:\Users\Fortu> kubectl get nodes
AME                 STATUS   ROLES           AGE     VERSION
pa-kubernetes        Ready    control-plane   6m5s    v1.33.1
pa-kubernetes-m02    Ready    <none>          3m21s   v1.33.1
```

Passo 3 – Verificar a instalação do kubectl

Confirme que o `kubectl` está configurado corretamente:

```
kubectl version --client
kubectl get pods -A
```

```
PS C:\Users\Fortu> kubectl get pods -A
NAMESPACE     NAME                                       READY   STATUS    RESTARTS        AGE
kube-system   coredns-674b8bbfcf-9vlzr                   1/1     Running   0               6m35s
kube-system   etcd-hpa-kubernetes                        1/1     Running   0               6m40s
kube-system   kindnet-lrzcr                              1/1     Running   0               4m4s
kube-system   kindnet-t9zpt                              1/1     Running   0               6m35s
kube-system   kube-apiserver-hpa-kubernetes              1/1     Running   0               6m42s
kube-system   kube-controller-manager-hpa-kubernetes     1/1     Running   0               6m47s
kube-system   kube-proxy-hd4pp                           1/1     Running   0               6m36s
kube-system   kube-proxy-t8zv2                           1/1     Running   0               4m4s
kube-system   kube-scheduler-hpa-kubernetes              1/1     Running   0               6m41s
kube-system   storage-provisioner                        1/1     Running   1 (6m10s ago)   6m34s
PS C:\Users\Fortu> kubectl version --client
Client Version: v1.32.2
Kustomize Version: v5.5.0
```

Passo 4 - Confirmar papéis dos nós

Para checar os detalhes de cada nó:

```
kubectl describe node hpa-kubernetes | findstr "Roles"
kubectl describe node hpa-kuberentes-m02 | findstr "Roles"
```

```
PS C:\Users\Fortu> kubectl describe node hpa-kubernetes | findstr "Roles"
Roles:              control-plane
PS C:\Users\Fortu> kubectl describe node hpa-kubernetes-m02 | findstr "Roles"
Roles:              <none>
```

Passo 4 – Habilitar o Metrics Server no Minikube

No Minikube, habilite o **metrics-server** com:

```
minikube addons enable metrics-server -p HPA-kubernetes
```

```
PS C:\Users\Fortu> minikube addons enable metrics-server -p HPA-kubernetes
* metrics-server is an addon maintained by Kubernetes. For any concerns contact minikube on GitHub.
You can view the list of minikube maintainers at: https://github.com/kubernetes/minikube/blob/master/OWNERS
  - Using image registry.k8s.io/metrics-server/metrics-server:v0.7.2
* The 'metrics-server' addon is enabled
```

Verifique se o deployment foi criado:

```
kubectl get deployment metrics-server -n kube-system
```

E espere até os Pods ficarem `Running`:

```
kubectl get pods -n kube-system | grep metrics-server
```

```
PS C:\Users\Fortu> kubectl get deployment metrics-server -n kube-system
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
metrics-server   1/1     1            1           105s
PS C:\Users\Fortu> kubectl get pods -n kube-system | findstr metrics-server
metrics-server-7fbb699795-cm88n          1/1     Running   0              107s
```

Parte 2 – Implantação da aplicação de exemplo (php-apache)

Nesta etapa, vamos criar um **Deployment** que roda a imagem `registry.k8s.io/hpa-example` (um servidor Apache com PHP) e expor esse Deployment como um **Service** dentro do cluster.

Passo 1 – Aplicar o manifesto pronto

Execute o comando abaixo para criar o Deployment e o Service:

```
kubectl apply -f https://k8s.io/examples/application/php-apache.yaml
```

application/php-apache.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-apache
spec:
  selector:
    matchLabels:
      run: php-apache
  template:
    metadata:
      labels:
        run: php-apache
    spec:
      containers:
      - name: php-apache
        image: registry.k8s.io/hpa-example
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: 500m
          requests:
            cpu: 200m
---
apiVersion: v1
kind: Service
metadata:
  name: php-apache
  labels:
    run: php-apache
spec:
  ports:
  - port: 80
  selector:
    run: php-apache
```

```
PS C:\Users\Fortu> kubectl apply -f https://k8s.io/examples/application/php-apache.yaml
deployment.apps/php-apache created
service/php-apache created
```

Passo 2 – Verificar se os recursos foram criados

Confira se o Deployment e o Service estão ativos:

```
kubectl get deployments
kubectl get pods
kubectl get svc
```

```
PS C:\Users\Fortu> kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
php-apache-6487c65df8-rd2sj   1/1     Running   0          99s
PS C:\Users\Fortu> kubectl get deployments
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
php-apache   1/1     1            1           102s
PS C:\Users\Fortu> kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP   12m
php-apache   ClusterIP   10.103.244.28   <none>        80/TCP    104s
PS C:\Users\Fortu>
```

Passo 3 – Testar o acesso à aplicação

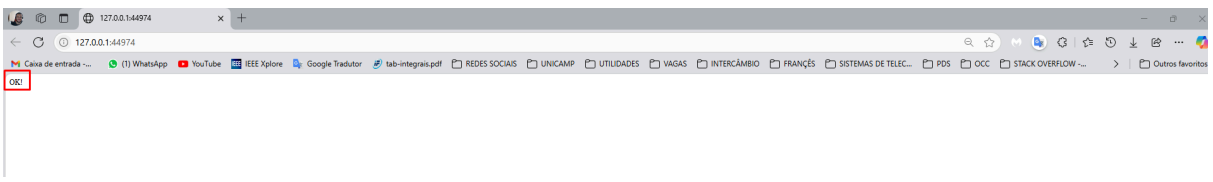
Como o serviço criado é do tipo **ClusterIP** (acessível apenas de dentro do cluster), vamos expor a aplicação para conseguir acessá-la a partir da máquina local.

No **Minikube**, você pode usar:

```
minikube service php-apache -p HPA-kubernetes
```

Isso abrirá automaticamente o navegador apontando para o endereço do serviço (um *tunnel* é criado entre sua máquina e o cluster).

```
PS C:\Users\Fortu> minikube service php-apache -p HPA-kubernetes
|-----------|------------|-------------|--------------|
| NAMESPACE |    NAME    | TARGET PORT |     URL      |
|-----------|------------|-------------|--------------|
| default   | php-apache |             | No node port |
|-----------|------------|-------------|--------------|
* service default/php-apache has no node port
! Services [default/php-apache] have type "ClusterIP" not meant to be exposed, however for local development minikube allows you to access this !
* Starting tunnel for service php-apache.
|-----------|------------|-------------|------------------------|
| NAMESPACE |    NAME    | TARGET PORT |          URL           |
|-----------|------------|-------------|------------------------|
| default   | php-apache |             | http://127.0.0.1:44974 |
|-----------|------------|-------------|------------------------|
* Opening service default/php-apache in default browser...
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```



Parte 3 – Criar o HorizontalPodAutoscaler (HPA)

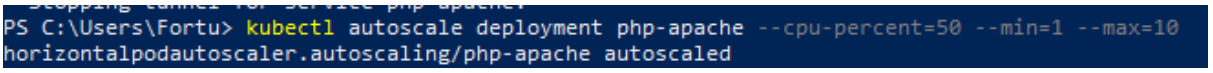
O HPA **ajusta automaticamente o número de réplicas do Deployment** `php-apache` para manter a média de utilização de CPU em **50%**.

- Número mínimo de réplicas: **1**
- Número máximo de réplicas: **10**
- Cada Pod do Deployment solicita **200m CPU**, então a meta de 50% significa 100m CPU em média.

Passo 1 - Criar o HPA

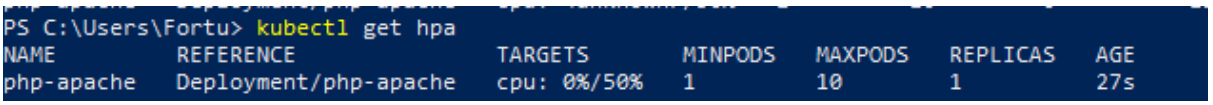
No PowerShell, execute:

```
kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
```



Passo 2 - Verificar o HPA criado

```
kubectl get hpa
```



Nota: Como ainda não há clientes enviando requisições, o TARGET aparece como 0%.

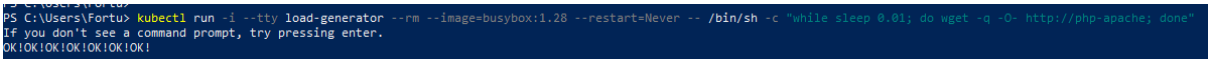
Parte 4 – Aumentar a carga e observar o HPA

O objetivo é simular aumento de carga na aplicação para que o HPA aumente o número de réplicas automaticamente.

Passo 1 - Criar um Pod gerador de carga

Abra um novo terminal PowerShell, e execute o comando:

```
kubectl run -i --tty load-generator --rm --image=busybox:1.28 --restart=Never -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"
```



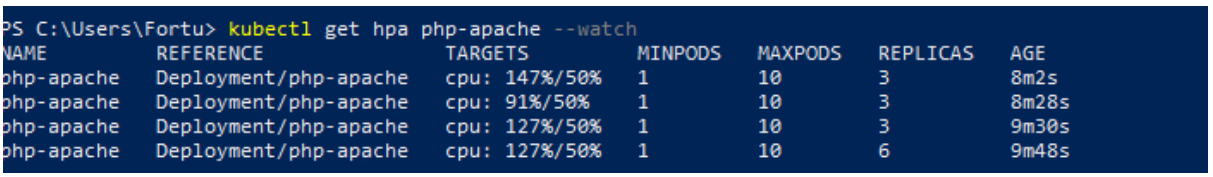
Explicação:

- `load-generator` → **nome do Pod cliente que gera carga.**
- `busybox:1.28` → **imagem leve que permite rodar comandos shell.**
- `while sleep 0.01; do wget -q -O- http://php-apache; done` → **loop infinito que envia requisições para o serviço PHP-Apache a cada 0.01s.**
- `--rm` → **remove o Pod quando você finalizar (`Ctrl+C`).**
- `--restart=Never` → **o Pod não será reiniciado automaticamente.**

Passo 2 - Monitorar o HPA

Em outro terminal, rode:

```
kubectl get hpa php-apache --watch
```

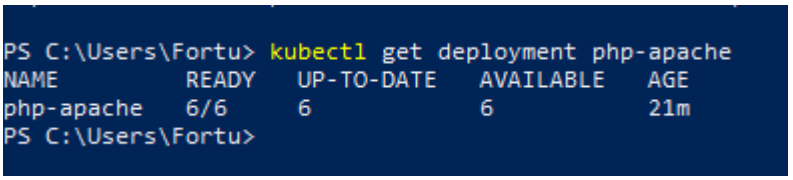


Note que o número de réplicas aumentou, indicando que o HPA está atuando para balancear a carga entre os pods.

Passo 3 - Verificar o Deployment

Depois que o HPA ajustar as réplicas, confira o Deployment:

```
kubectl get deployment php-apache
```

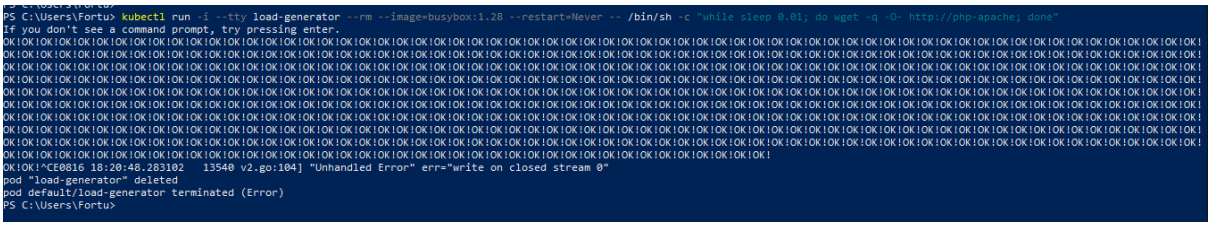


Parte 5 – Parar a geração de carga e observar a redução automática

Passo 1 - Parar o Pod gerador de carga

No terminal onde você executou o Pod `load-generator`, pressione:

```
Ctrl + C
```



- Isso encerra o Pod que estava enviando requisições continuamente para o serviço `php-apache`.
- Como resultado, a carga no cluster **vai cair para 0% de CPU**.

Passo 2 - Monitorar a redução do HPA

Ainda em outro terminal, você pode continuar monitorando o HPA:

```
kubectl get hpa php-apache --watch
```

```
PS C:\Users\Fortu> kubectl get hpa php-apache --watch
NAME         REFERENCE                TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   cpu: 0%/50%   1         10        6          16m
php-apache   Deployment/php-apache   cpu: 0%/50%   1         10        6          18m
php-apache   Deployment/php-apache   cpu: 0%/50%   1         10        3          18m
php-apache   Deployment/php-apache   cpu: 0%/50%   1         10        3          19m
php-apache   Deployment/php-apache   cpu: 0%/50%   1         10        1          19m
```

Passo 3 - Verificar o Deployment

Confirme que o Deployment também voltou ao estado mínimo:

```
kubectl get deployment php-apache
```

```
php-apache   6/6     6            6           28m
PS C:\Users\Fortu> kubectl get deployment php-apache
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
php-apache   1/1     1            1           30m
PS C:\Users\Fortu>
```

Parte 6 – Autoscaling com múltiplas métricas e métricas customizadas

O HPA padrão usa apenas **CPU** para escalar os Pods. Com a versão `autoscaling/v2`, é possível:

- Usar **mais de uma métrica** para decidir o número de réplicas.
- Trabalhar com **métricas customizadas**, como métricas de Pods ou de objetos (Ingress, Services, etc.)

Passo 1 - Obter o YAML atual do HPA

No PowerShell, salve o HPA em formato autoscaling/v2

```
kubectl get hpa php-apache -o yaml > C:\temp\hpa-v2.yaml
```

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  creationTimestamp: "2025-08-16T21:07:58Z"
  name: php-apache
  namespace: default
  resourceVersion: "2827"
  uid: f118de81-4dac-4083-a510-d660b4e8a215
spec:
  maxReplicas: 10
  metrics:
  - resource:
      name: cpu
      target:
        averageUtilization: 50
        type: Utilization
    type: Resource
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
status:
  conditions:
  - lastTransitionTime: "2025-08-16T21:12:56Z"
    message: recommended size matches current size
    reason: ReadyForNewScale
    status: "True"
    type: AbleToScale
  - lastTransitionTime: "2025-08-16T21:08:13Z"
    message: the HPA was able to successfully calculate a replica count from cpu resource
      utilization (percentage of request)
    reason: ValidMetricFound
    status: "True"
    type: ScalingActive
  - lastTransitionTime: "2025-08-16T21:27:08Z"
    message: the desired replica count is less than the minimum replica count
    reason: TooFewReplicas
    status: "True"
    type: ScalingLimited
  currentMetrics:
  - resource:
      current:
        averageUtilization: 0
        averageValue: 1m
      name: cpu
    type: Resource
  currentReplicas: 1
  desiredReplicas: 1
  lastScaleTime: "2025-08-16T21:27:08Z"
```

Note que `targetCPUUtilizationPercentage` foi substituído por um array `metrics`.