# Guía de Kubernetes avanzado en Minikube

## Prerrequisitos

### 1. Minikube y Docker

* Descarga el instalador de Minikube desde la [página oficial de Minikube](https://minikube.sigs.k8s.io/docs/start/).
* Sigue las instrucciones oficiales de [dockerdocs](https://docs.docker.com/engine/install/)

### 2. Kubectl

* Visita la [documentación oficial de kubectl](https://kubernetes.io/docs/tasks/tools/install-kubectl/) para seguir las instrucciones específicas de tu sistema operativo.

### 3. Verificación de instalación

# minikube versión  
 # kubectl version –client

# docker --version

**Configuración**

## 4. Instalar metrics server add-on e Iniciar minikube

# minikube addons enable metrics-server (Necesario para obtener métricas)

# minikube start

**Autoscaling**

**6. Archivo stress-deployment.yaml**

Crearemos un deployment utilizando stress-ng como método para saturar recursos y así testear el autoscaling de pods. Ejemplo básico para stress-deployment.yaml:

apiVersion: apps/v1

kind: Deployment

metadata:

name: memory-hpa-test

namespace: default

spec:

replicas: 1

selector:

matchLabels:

app: memory-hpa-test

template:

metadata:

labels:

app: memory-hpa-test

spec:

containers:

- name: pod-stress-test

image: polinux/stress-ng

command: [ "/bin/sh", "-c" ]

args: [ "stress-ng --cpu 1 --cpu-load 5 --timeout 300s" ]

#args: [ "stress-ng --cpu 10 --cpu-load 15 --timeout 300s" ] BUMP CPU

resources:

requests:

memory: "64Mi" # Minimum memory request

cpu: "250m" # Minimum CPU request

limits:

memory: "512Mi" # Maximum memory limit IF WE DISABLE THIS->OOMKilled

cpu: "400m" # Maximum CPU limit

**7. Archivo** **hpa.yaml**

Para trabajar con autoscaling de Pods en kubernetes, se utiliza un objecto llamado **Horizontal Pod Autoscaler** (HPA). A continuación se muestra un ejemplo básico de un archivo hpa.yaml:

apiVersion: autoscaling/v2

kind: HorizontalPodAutoscaler

metadata:

name: memory-hpa-test

namespace: default

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: memory-hpa-test

minReplicas: 1

behavior:

scaleDown:

policies:

- periodSeconds: 5

type: Percent

value: 400 # Hasta 4 veces el numero de current pods pueden ser quitados

selectPolicy: Max

scaleUp:

policies:

- periodSeconds: 5

type: Pods

value: 6 # Hasta 6 pods pueden ser agregados

- periodSeconds: 5

type: Percent

value: 400

selectPolicy: Max

#stabilizationWindowSeconds: 5

maxReplicas: 10

metrics:

- type: Resource

resource:

name: cpu

target:

type: Utilization

averageUtilization: 85 # Target CPU utilization percentage

### 8. Crear el deployment

# kubectl apply -f stress-deployment.yaml

# kubectl get pods (Ver status de pods)

# kubectl top pods (Ver metricas de pods)

## 9. Crear el HPA

# kubectl apply -f hpa.yaml

# kubectl get hpa (Ver status de HPA)

# kubectl describe hpa (Ver más detalles de HPA)

### 

### 10. Generar stress test

# kubectl get pods

# kubectl exec -it <POD\_NAME> -- bash (Meterse dentro del pod)

# stress-ng --cpu 10 --cpu-load 15 --timeout 300s (Generar stress test)

La prueba de stress hará subir los recursos del pod, lo cual generará que el HPA detecte este cambio y escale horizontalmente el deployment, agregando 1 o más pods.

Luego, cuando la prueba de stress finalice, el HPA esperará unos minutos debido a un proceso de cooling entre scale up y scale down y procederá a escalar hacia abajo cuando el consumo de los recursos se normalice.

En este tiempo podemos monitorear con los comandos get y describe del punto 9.

## 

## Crear un Ingress

Un Ingress de Kubernetes es un objecto que nos permite exponer un servicio hacia afuera del cluster, ya sea de forma privada o pública.

Funciona como Load Balancer lógico con reglas de balanceo que pueden apuntar a distintos servicios y apps dentro del cluster.

Se maneja con un Ingress Controller el cual enlaza la comunicación entre los servicios y el ingress. En nuestro caso es un controller de nginx instalado automáticamente por minikube, pero dependiendo del Cloud provider esto puede variar.

### 11. Archivo deployment-nginx-basic.yaml

Crearemos un deployment básico de nginx para la demo:

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 1

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

resources:

limits:

memory: "128Mi"

cpu: "500m"

requests:

memory: "64Mi"

cpu: "250m"

ports:

- containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

selector:

app: nginx

ports:

- protocol: TCP

port: 80

targetPort: 80

type: NodePort

Una vez que hayas creado el archivo, puedes crear el Deployment ejecutando el siguiente comando en tu terminal:

# kubectl apply -f deployment-nginx-basic.yaml

# kubectl get deployments

# kubectl get pods

# kubectl get service

### 

### 12. Archivo ingress.yaml

Crearemos el ingress con el siguiente archivo:

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: nginx-ingress

namespace: my-app

annotations:

kubernetes.io/ingress.class: "nginx" # Specify the ingress controller to use

nginx.ingress.kubernetes.io/rewrite-target: /

spec:

rules:

- host: nginx.example.com # Replace with your domain name

http:

paths:

- path: /test/app1

pathType: Prefix

backend:

service:

name: nginx-service # The Service exposing the app

port:

number: 80

- path: /test/app2

pathType: Prefix

backend:

service:

name: hw-nginx-service # The Service exposing the app

port:

number: 80

Aplicar los cambios:

# kubectl apply -f ingress.yaml

# kubectl get ingress

### 13. Cómo probar el ingress

Como se puede ver en el archivo ingress.yaml, el ingress necesita un hostname, en este caso nginx.example.com.

Como no tenemos un DNS creado para ese host, debemos crear una línea en nuestro archivo host local de Windows:

Abrir Notepad como admin -> C:\Windows\System32\drivers\etc\hosts

Agregar la IP de minikube (comando “minikube ip” para obtener), seguido del host.

Ejemplo:

192.168.58.2 nginx.example.com

Una vez hecho esto, podremos probar el acceso a nuestra app con curl:

# curl http://nginx.example.com/test/app1

Notaremos que esto funciona, en cambio, /test/app2 no funcionará porque no la tenemos definida.

### 13. Deployment de la segunda app, archivo deployment-nginx-hw.yaml

### apiVersion: apps/v1

### kind: Deployment

### metadata:

### name: hw-deployment

### spec:

### replicas: 1

### selector:

### matchLabels:

### app: hw-nginx

### template:

### metadata:

### labels:

### app: hw-nginx

### spec:

### containers:

### - name: hw-nginx

### image: nginx

### resources:

### limits:

### memory: "128Mi"

### cpu: "500m"

### requests:

### memory: "64Mi"

### cpu: "250m"

### ports:

### - containerPort: 80

### volumeMounts:

### - name: nginx-html

### mountPath: /usr/share/nginx/html

### volumes:

### - name: nginx-html

### configMap:

### name: hw-nginx-html

### ---

### apiVersion: v1

### kind: ConfigMap

### metadata:

### name: hw-nginx-html

### data:

### index.html: |

### <!DOCTYPE html>

### <html lang="en">

### <head>

### <meta charset="UTF-8">

### <meta name="viewport" content="width=device-width, initial-scale=1.0">

### <title>Hello World</title>

### </head>

### <body>

### <h1>Hello, World!</h1>

### <p>Welcome to your NGINX server running in Kubernetes.</p>

### </body>

### </html>

### ---

### apiVersion: v1

### kind: Service

### metadata:

### name: hw-nginx-service

### spec:

### selector:

### app: hw-nginx

### ports:

### - protocol: TCP

### port: 80

### targetPort: 80

### type: NodePort

Probar app2:

### # kubectl apply -f deployment-nginx-hw.yaml

### # curl http://nginx.example.com/test/app2

## Limpieza

# kubectl delete -f deployment-nginx-hw.yaml

# kubectl delete -f ingress.yaml

# kubectl delete -f deployment-nginx-basic.yaml

# kubectl delete -f hpa.yaml

# kubectl delete -f stress-deployment.yaml

### Detener Minikube

# minikube stop

Este comando detiene la máquina virtual de Minikube y libera la memoria y otros recursos que se estaban utilizando. Si ya no planeas usar Minikube, también puedes eliminarlo completamente con:

# minikube delete

Este comando eliminará la máquina virtual y todos los datos asociados, asegurando que tu entorno esté completamente limpio.