

KUBERNETES CON LOAD BALANCER Y AWS



Hecho por Juan Medrano

INDICE

Paso 1: Preparación del Entorno (WSL2).....	3
Actualizar el sistema e instalar herramientas básicas.....	3
Instalar k3s (el motor de Kubernetes).....	3
Iniciar el servicio y esperar a que despierte.....	4
Paso 2: Creación de la Aplicación.....	4
Crear carpetas de trabajo.....	4
Crear el archivo de la aplicación.....	4
Crear la interfaz web.....	5
Verificar que todo está listo.....	5
Paso 3: Despliegue en Kubernetes.....	6
Crear y aplicar el Namespace.....	6
Paso 4: Verificación del Balanceo de Carga local.....	7
4.1 - Abrir el túnel (Port-Forward).....	7
4.2 - Probar en el Navegador.....	7
Paso 5: Conexión con AWS.....	8
Configuración en la Consola de AWS.....	8
El Túnel hacia la Nube.....	8
Paso 6: Escalado Dinámico.....	9

Paso 1: Preparación del Entorno (WSL2)

Aquí mostramos que el clúster de Kubernetes está activo. El comando nos devuelve el nombre de nuestra máquina con el estado 'Ready', lo que significa que el cerebro de Kubernetes ya está listo para recibir órdenes.

```
juan@A6Alumno21:/mnt/c/Windows/System32$ kubectl get nodes
NAME      STATUS    ROLES     AGE   VERSION
a6alumno21 Ready    control-plane   16m  v1.34.3+k3s1
juan@A6Alumno21:/mnt/c/Windows/System32$
```

Actualizar el sistema e instalar herramientas básicas

```
Setting up libpam-systemd:amd64 (255.4-1ubuntu8.12) ...
Setting up mesa-libgallium:amd64 (25.0.7-0ubuntu0.24.04.2) ...
Setting up cloud-init (25.2-0ubuntu1~24.04.1) ...
Installing new version of config file /etc/cloud/templates/sources.list.debian.deb822.tpl ...
Setting up dconf-gsettings-backend:amd64 (0.40.0-4ubuntu0.1) ...
Setting up libgbm1:amd64 (25.0.7-0ubuntu0.24.04.2) ...
Setting up libgl1-mesa-dri:amd64 (25.0.7-0ubuntu0.24.04.2) ...
Setting up libegl-mesa0:amd64 (25.0.7-0ubuntu0.24.04.2) ...
Setting up libglx-mesa0:amd64 (25.0.7-0ubuntu0.24.04.2) ...
Processing triggers for rsyslog (8.2312.0-3ubuntu9.1) ...
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for dbus (1.14.10-4ubuntu4.1) ...
Processing triggers for install-info (7.1-3build2) ...
Processing triggers for libc-bin (2.39-0ubuntu8.6) ...
juan@A6Alumno21:/mnt/c/Windows/System32$
```

```
juan@A6Alumno21:/mnt/c/Windows/System32$ sudo apt install -y curl wget git
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
curl is already the newest version (8.5.0-2ubuntu10.6).
wget is already the newest version (1.21.4-1ubuntu4.1).
wget set to manually installed.
git is already the newest version (1:2.43.0-1ubuntu7.3).
git set to manually installed.
The following package was automatically installed and is no longer required:
  liblvm19
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
juan@A6Alumno21:/mnt/c/Windows/System32$
```

Instalar k3s (el motor de Kubernetes)

```
juan@A6Alumno21:/mnt/c/Windows/System32$ curl -sfL https://get.k3s.io | K3S_KUBECONFIG_MODE="644" sh -
[INFO]  Finding release for channel stable
[INFO]  Using v1.34.3+k3s1 as release
[INFO]  Downloading hash https://github.com/k3s-io/k3s/releases/download/v1.34.3+k3s1/sha256sum-amd64.txt
[INFO]  Downloading binary https://github.com/k3s-io/k3s/releases/download/v1.34.3+k3s1/k3s
```

Iniciar el servicio y esperar a que despierte

```
[...]
INFO[0000] Closing database connections...
[1]+ Done sudo k3s server
juan@A6Alumno21:/mnt/c/Windows/System32$
```

Paso 2: Creación de la Aplicación

Crear carpetas de trabajo

```
juan@A6Alumno21:/mnt/c/Windows/System32$ mkdir -p ~/kubernetes-aws-practice/app
cd ~/kubernetes-aws-practice/app
juan@A6Alumno21:~/kubernetes-aws-practice/app$
```

Crear el archivo de la aplicación

```
juan@A6Alumno21:~/kubernetes-aws-practice/app$ cat > app.py << 'EOF'
> from flask import Flask, jsonify, send_from_directory
import os
import socket
from datetime import datetime
import sys

app = Flask(__name__)
> POD_NAME = os.getenv('POD_NAME', 'Unknown Pod')
POD_NAMESPACE = os.getenv('POD_NAMESPACE', 'default')

@app.route('/')
def index():
    return send_from_directory('.', 'index.html')

@app.route('/pod-info')
def pod_info():
    return jsonify({
        'pod_name': POD_NAME,
        'namespace': POD_NAMESPACE,
        'hostname': socket.gethostname(),
        'timestamp': datetime.now().isoformat()
    })

@app.route('/health')
def health():
    return jsonify({'status': 'healthy', 'pod': POD_NAME}), 200

if __name__ == '__main__':
    print(f"[{POD_NAME}] Iniciando servidor Flask...", file=sys.stderr)
    app.run(host='0.0.0.0', port=5000, debug=False)
EOF
juan@A6Alumno21:~/kubernetes-aws-practice/app$
```

Crear la interfaz web

```
Juan@A6Alumno21:~/kubernetes-aws-practice/app$ cat > index.html << 'EOF'
<!DOCTYPE html>
<html>
<head>
    <title>Kubernetes Load Balancer</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            display: flex;
            justify-content: center;
            align-items: center;
            min-height: 100vh;
            margin: 0;
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
        }
        .container {
            background: white;
            padding: 50px;
            border-radius: 10px;
            box-shadow: 0 10px 25px rgba(0,0,0,0.2);
            text-align: center;
            max-width: 500px;
        }
        h1 { color: #667eea; margin: 0 0 30px 0; }
        .info {
            background: #f0f0f0;
            padding: 20px;
            border-radius: 5px;
            margin: 20px 0;
        }
        .pod-name {
            font-size: 28px;
            color: #764ba2;
            font-weight: bold;
            font-family: monospace;
        }
        .timestamp { color: #666; font-size: 13px; margin-top: 10px; }
        .instruction {
            background: #e0f2fe;
            padding: 15px;
            border-radius: 5px;
            color: #0369a1;
            font-size: 14px;
            margin-top: 20px;
        }
        .refresh-button {
            margin-top: 20px;
            padding: 10px 20px;
            background: #667eea;
            color: white;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            font-size: 14px;
        }
        .refresh-button:hover { background: #764ba2; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Kubernetes Load Balancer</h1>
        <div class="info">
            <p>This is a simple Kubernetes Load Balancer. It uses a single Pod containing a Flask application. The application displays the name of the Pod it is running on, the timestamp of its creation, and provides a refresh button to reload the page. The background of the page is a linear gradient from #667eea to #764ba2.</p>
        </div>
        <div class="pod-name">Juan</div>
        <div class="timestamp">Jan 13 10:16</div>
        <div class="instruction">Click here to refresh</div>
        <div class="refresh-button" onclick="location.reload()>Refresh</div>
    </div>
</body>
</html>
EOF
```

Verificar que todo está listo

En esta fase, hemos preparado el entorno de desarrollo creando los archivos fuente de la aplicación dentro del directorio `~/kubernetes-aws-practice/app`. Como se observa en la captura, disponemos de `app.py` (lógica del servidor Flask), `index.html` (interfaz visual) y `requirements.txt` (dependencias necesarias). Estos archivos serán inyectados posteriormente en los Pods de Kubernetes mediante un ConfigMap, evitando la necesidad de construir imágenes Docker complejas manualmente.

```
juan@A6Alumno21:~/kubernetes-aws-practice/app$ ls -l
total 12
-rw-r--r-- 1 juan juan  802 Jan 13 10:16 app.py
-rw-r--r-- 1 juan juan 2712 Jan 13 10:18 index.html
-rw-r--r-- 1 juan juan   29 Jan 13 10:18 requirements.txt
juan@A6Alumno21:~/kubernetes-aws-practice/app$
```

Paso 3: Despliegue en Kubernetes

Crear y aplicar el Namespace

```
juan@A6Alumno21:~/kubernetes-aws-practice$ cat > namespace.yaml << 'EOF'
apiVersion: v1
kind: Namespace
metadata:
  name: load-balancer-demo
EOF
juan@A6Alumno21:~/kubernetes-aws-practice$ kubectl apply -f namespace.yaml
namespace/load-balancer-demo created
```

Aquí estamos verificando que Kubernetes ha creado con éxito nuestro espacio de trabajo llamado `load-balancer-demo`. Esto nos asegura que nuestra práctica no interferirá con otros servicios del sistema.

```
juan@A6Alumno21:~/kubernetes-aws-practice$ kubectl get namespaces
NAME          STATUS   AGE
default       Active   79m
kube-node-lease Active   79m
kube-public   Active   79m
kube-system   Active   79m
load-balancer-demo Active  10s
juan@A6Alumno21:~/kubernetes-aws-practice$
```

Esta es la captura más importante del despliegue. Aquí vemos el ecosistema completo funcionando: 1) Los **3 Pods** están en estado 'Running', 2) El **Service** tiene una IP asignada para actuar como Load Balancer y 3) El **Deployment** confirma que las 3 réplicas deseadas están operativas. Kubernetes ya está gestionando nuestra aplicación automáticamente.

```
juan@A6Alumno21:~/kubernetes-aws-practice$ kubectl get all -n load-balancer-demo
NAME                           READY   STATUS    RESTARTS   AGE
pod/web-app-549f797496-5sddl   1/1     Running   0          27s
pod/web-app-549f797496-r4kgw   1/1     Running   0          27s
pod/web-app-549f797496-xg6rx   1/1     Running   0          27s

NAME                  TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/web-app-service   LoadBalancer   10.43.100.248  <pending>    80:30701/TCP  14s

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/web-app     3/3     3           3           27s

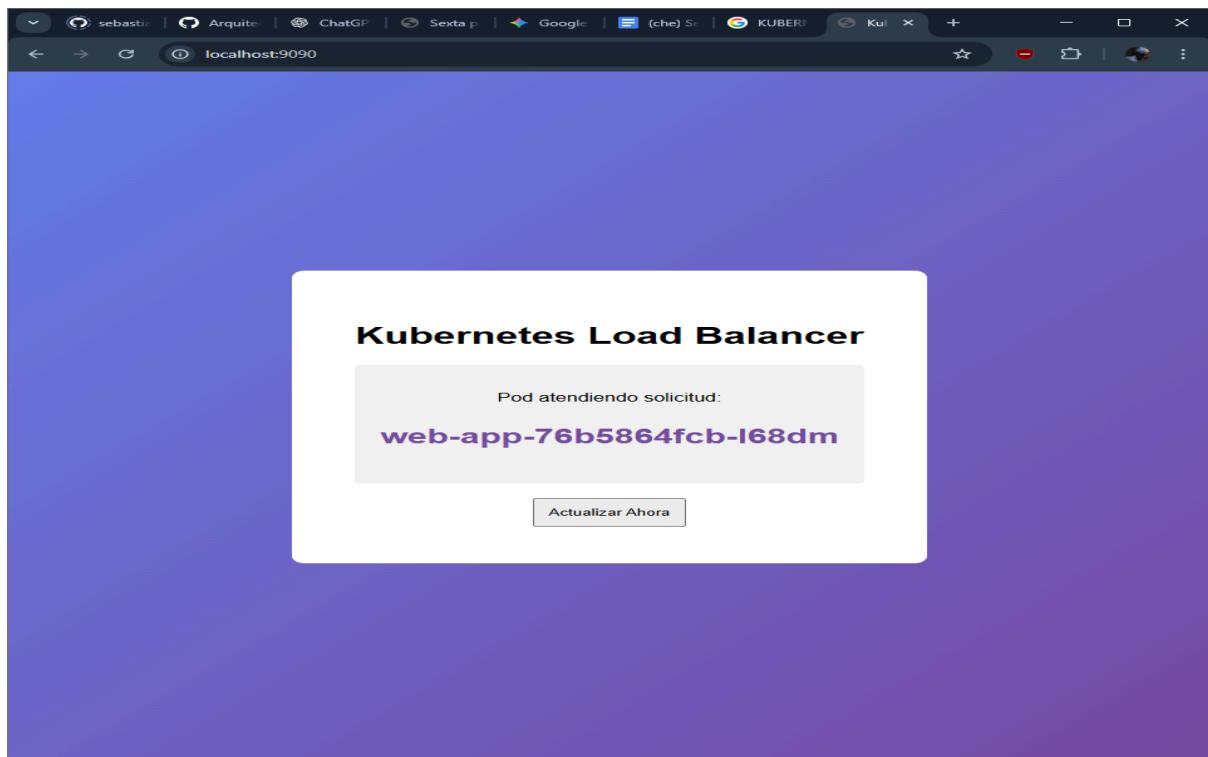
NAME                           DESIRED   CURRENT   READY   AGE
replicaset.apps/web-app-549f797496  3         3         3        27s
juan@A6Alumno21:~/kubernetes-aws-practice$
```

Paso 4: Verificación del Balanceo de Carga local

4.1 - Abrir el túnel (Port-Forward)

```
NAME           DESIRED   CURRENT   READY   AGE
replicaset.apps/web-app-549f797496   3         3         3        27s
juan@A6Alumno21:~/kubernetes-aws-practice$ kubectl port-forward -n load-balancer-demo svc/web-app-service 8080:80
```

4.2 - Probar en el Navegador



Paso 5: Conexión con AWS

Configuración en la Consola de AWS

The screenshot shows the AWS CloudFront Rules configuration interface. It displays two rules:

Regla de salida 1

Tipo	Información	Protocolo	Información	Intervalo de puertos	Información
SSH		TCP		22	
Tipo de destino	Información	Destino	Información	Descripción: opcional	
Anywhere-IPv4		0.0.0.0/0			
		0.0.0.0/0	X		

Regla de salida 2

Tipo	Información	Protocolo	Información	Intervalo de puertos	Información
HTTP		TCP		80	
Tipo de destino	Información	Destino	Información	Descripción: opcional	
Anywhere-IPv4		0.0.0.0/0			
		0.0.0.0/0	X		

El Túnel hacia la Nube

```
[7]+ Killed ssh -i ~/labsuser.pem -N -R 8888:localhost:8080 ubuntu@54.209.201.33
juan@A6Alumno21:/mnt/c/Windows/System32$ # Ejecuta el túnel. Si se queda "pensando" SIN dar error de timeout, ¡DÉJALO ASÍ!
ssh -i ~/labsuser.pem" -v -N -R 8888:localhost:8080 ubuntu@54.209.201.33
[5]+ Killed kubectl port-forward -n load-balancer-demo svc/web-app-service 8080:80
OpenSSH_9.6p1 Ubuntu-3ubuntu13.14, OpenSSL 3.0.13 30 Jan 2024
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: /etc/ssh/ssh_config line 19: include /etc/ssh/ssh_config.d/*.conf matched no files
debug1: /etc/ssh/ssh_config line 21: Applying options for *
```

Se confirma la conectividad híbrida mediante la ejecución de un `curl` desde la instancia de AWS. El tráfico viaja desde la nube por el puerto 8888, cruza el túnel SSH inverso hasta nuestro puerto local 8080 y llega finalmente a los Pods de Kubernetes. La respuesta JSON recibida en la terminal de AWS es la prueba técnica de que el despliegue es exitoso.

```
juan@A6Alumno21:~/kubernetes-aws-practice$ curl -s http://localhost:8888/pod-info
juan@A6Alumno21:~/kubernetes-aws-practice$ curl -v http://localhost:8888/pod-info
Host localhost:8888 was resolved.
IPv6: ::1
IPv4: 127.0.0.1
Trying [::1]:8888...
connect to ::1 port 8888 from ::1 port 37124 failed: Connection refused
Trying 127.0.0.1:8888...
connect to 127.0.0.1 port 8888 from 127.0.0.1 port 42318 failed: Connection refused
Failed to connect to localhost port 8888 after 0 ms: Couldn't connect to server
Closing connection
url: (7) Failed to connect to localhost port 8888 after 0 ms: Couldn't connect to server
juan@A6Alumno21:~/kubernetes-aws-practice$
```

Paso 6: Escalado Dinámico

Para concluir la práctica, se realizó un **escalado horizontal** del despliegue. Pasamos de 3 a 5 réplicas de forma instantánea. Kubernetes gestionó automáticamente la creación de los nuevos contenedores y el Load Balancer comenzó a distribuir el tráfico entre todos ellos. Esto garantiza que la aplicación pueda manejar un mayor volumen de usuarios sin perder rendimiento.

```
C:\Windows\System32>wsl  
juan@A6Alumno21:/mnt/c/Windows/System32$ kubectl scale deployment web-app -n load-balancer-demo --replicas=5  
kubectl get pods -n load-balancer-demo  
deployment.apps/web-app scaled  
NAME          READY   STATUS        RESTARTS   AGE  
web-app-76b5864fcb-569xb  0/1   ContainerCreating  0          0s  
web-app-76b5864fcb-5ndk8  1/1   Running       0          96m  
web-app-76b5864fcb-dvd4v  1/1   Running       0          96m  
web-app-76b5864fcb-168dm  1/1   Running       0          96m  
web-app-76b5864fcb-rvm97  0/1   ContainerCreating  0          0s  
juan@A6Alumno21:/mnt/c/Windows/System32$
```