
Adrián Aceitón Palomo. 2º ASIR
Arquitectura en la nube

# PRÁCTICA: KUBERNETES CON LOAD BALANCER Y AWS

## Introducción

- **Kubernetes** es un sistema que gestiona automáticamente aplicaciones en múltiples máquinas. Sirve para desplegar, escalar, supervisar y recuperar aplicaciones sin intervención manual.
- **k3d** es una versión ligera de Kubernetes (*50 MB vs 500 MB*). Funciona increíblemente bien en WSL2 sin Docker ni complicaciones.
- **POD** es la unidad mínima de Kubernetes, un contenedor ejecutándose. Tiene IP propia, es efímero (*desaparece cuando muere*), aislado.
- **Deployment** es un controlador que crea y mantiene múltiples pods idénticos, su responsabilidad es que, si un pod muere, crea uno nuevo. Mantiene la cantidad especificada.
- **Service** es un intermediario que da una IP estable para acceder a pods. Los pods tienen IPs que cambian. Service proporciona una IP que nunca cambia.
- **Load Balancer** es un tipo de Service que distribuye solicitudes entre múltiples pods. El LoadBalancer elige qué Pod atender (*round robin, menos conexiones, etc.*) va a ser atendido el cliente. Si un pod cae, otros atienden. La carga se distribuye.
- **Namespace** es un espacio aislado dentro de Kubernetes (*como carpetas*). Separar desarrollo/producción, equipos, versiones diferentes.
- **ConfigMap** almacena configuración (*URLs, puertos, etc.*). Cambias config sin recompilar ni redeploy.
- **Secret** es un ConfigMap pero para datos sensibles (*contraseñas, tokens*). Está cifrado, no es visible en logs.
- **Labels** son etiquetas para identificar objetos (*app: web-app, version: 1.0*).
- **Selectors** son formas de filtrar objetos por sus labels.
- **Escalado horizontal** es aumentar número de pods (*de 3 a 5*). La carga se distribuye entre más máquinas, Kubernetes lo hace automáticamente.
- **Escalado vertical** es aumentar recursos de una máquina (2GB → 8GB RAM), tiene límites.
- **Session Affinity** mantiene al cliente en el mismo pod si ya está conectado.
- **Port-forward** es un túnel que expone puertos de Kubernetes en tu máquina local. Su uso es solo para desarrollo/testing.
- **Túnel SSH** es una conexión SSH que redirige puertos desde máquina

remota a local. EC2 accede a Kubernetes sin ruta de red directa.

- **Ingress** es un objeto que gestiona acceso externo a servicios (*dominios, HTTPS, routing*). Su ventaja sobre port-forward es que tiene nombres reales (*miapp.com*), HTTPS, y su uso es profesional.
- **Persistent Volume** es almacenamiento que persiste más allá de la vida del pod. Si un pod muere, sus datos desaparecen. PV los guarda en almacenamiento externo.
- **StateFulset** es como Deployment pero para aplicaciones con estado (*BD, colas, etc.*). Los pods tienen identidad. El orden importa.
- **Job** ejecuta una tarea UNA VEZ y termina (*no indefinido como Deployment*). Se usa en migraciones, backups, procesamiento por lotes.
- **CronJob** es un Job que se ejecuta en un horario específico.
- **Yaml** es el formato para describir objetos de Kubernetes de forma legible.

**Arquitectura Simple**

1. CLIENTE
2. PORT-FORWARD / INGRESS
3. SERVICE (*IP estable*)
4. LOAD BALANCER (*elige pod*)
5. DEPLOYMENT (*3 replicas*)
6. PODS (*tu aplicación*)

**Flujo de una Petición**

1. Cliente solicita http://localhost:8080
2. Port-forward escucha, redirige a Service
3. Service elige qué pod (*LoadBalancer*)
4. Pod ejecuta tu código (*Flask, Node, Java, etc.*)
5. Respuesta vuelve al cliente

Si un pod muere, Deployment crea uno nuevo automáticamente.

## Objetivos

Al finalizar esta práctica, serás capaz de:

- Instalar Kubernetes local directamente en WSL2 (*sin Docker, sin dependencias*)
- Desplegar múltiples replicas de una aplicación en Kubernetes

- Configurar un Load Balancer interno para distribuir tráfico
- Conectar tu cluster local de Kubernetes con instancias EC2 en AWS
- Monitorear el tráfico y verificar el balanceo de carga
- Escalar aplicaciones dinámicamente en Kubernetes

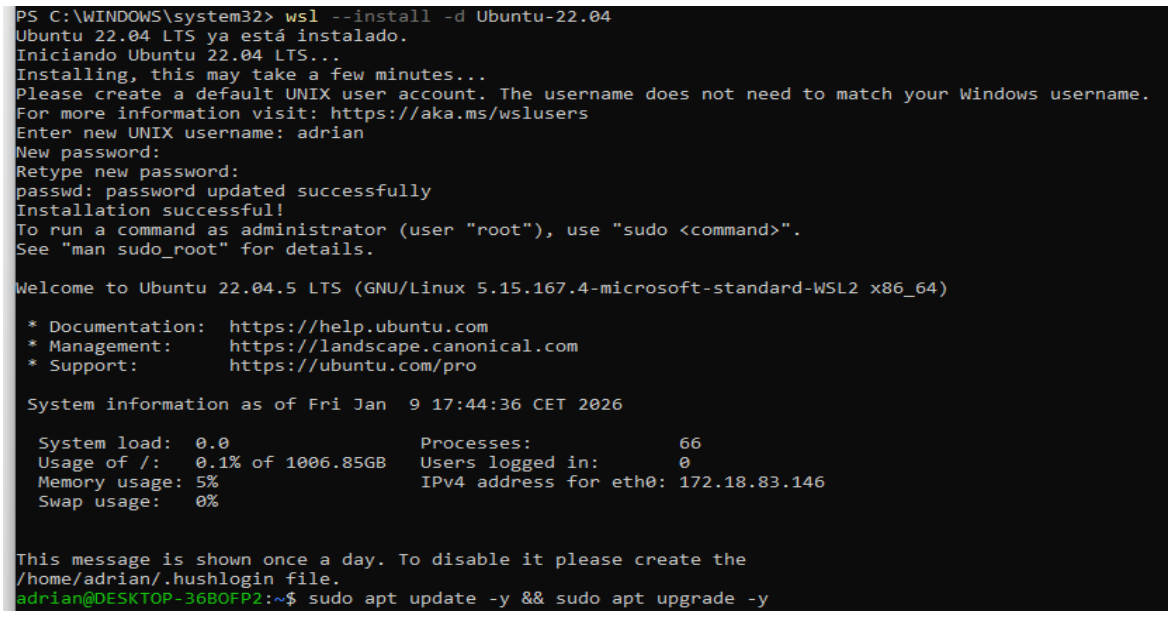
## Requisitos Previos

- Windows 11 con WSL2 habilitado
- Ubuntu 22.04 o superior en WSL2
- kubectl instalado
- k3s instalado (*Kubernetes ligero sin Docker*)
- Cuenta AWS Free Tier
- Acceso SSH a instancias EC2
- Terminal en Windows o WSL
- Al menos 2 GB RAM disponibles

## Requisitos de AWS (*100% alcanzable con Free Tier*)

- Acceso a EC2 (*crear/gestionar instancias*)
- Acceso a Security Groups
- Acceso a Key Pairs

Aquí he actualizado Ubuntu y me he instalado lo básico (curl/wget/git) para que luego no me falle nada al descargar cosas.



# PARTE 0: PREPARACIÓN DEL ENTORNO LOCAL EN WSL2

## 0.1 – Instalar k3s (Kubernetes ultraligero - SIN Docker)

k3s es Kubernetes completo, pero sin la complejidad. Es perfecto para desarrollo y testing. En WSL2 (*Ubuntu*):

*# Actualizar sistema*

*sudo apt update – y && sudo apt upgrade*

*-y # Instalar dependencias mínimas*

*sudo apt install -y curl wget git*

*# Instalar k3s SIN systemd (mejor para WSL2)*

*curl -sfL https://get.k3s.io | K3S_KUBECONFIG_MODE="644" sh*

*- # Verificar que k3s está instalado*

*sudo k3s --version*

*# Iniciar k3s y esperar 10 segundos a que inicie*

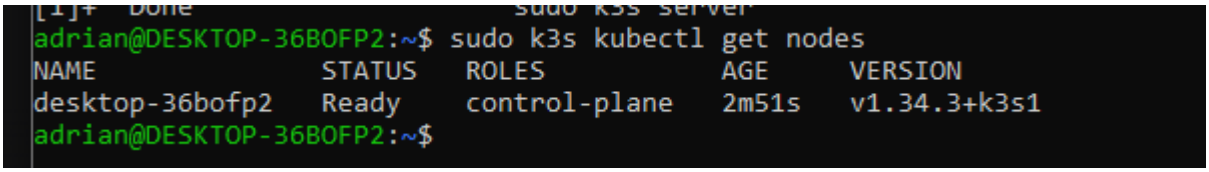
*sudo k3s server & sleep 10*

*# Verificar que está*

*corriendo sudo k3s kubectl*

*get nodes*

Aquí he montado k3s en WSL2. Básicamente ya tengo Kubernetes funcionando y he mirado que el nodo salga “Ready”.




## 0.2 – Instalar kubectl localmente (*para comodidad*)

*# Descargar kubectl*

*curl -LO "https://dl.k8s.io/release/:(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amdc4/kubectl"*

*sudo chmod +x kubectl*

*sudo mv kubectl /usr/local/bin/*

*# Verificar*

*kubectl version --client*

*# Configurar kubeconfig para acceder sin sudo*

*mkdir -p :HOME/.kube*

*sudo cp /etc/rancher/k3s/k3s.yaml :HOME/.kube/config*

*sudo chown :(id -u)::(id -g) :HOME/.kube/config*

*sudo chmod c00 :HOME/.kube/config*

*# Verificar que funciona sin sudo kubectl get nodes*

```
adrian@DESKTOP-36BOFP2:~$ sudo chmod +x kubectl
adrian@DESKTOP-36BOFP2:~$ sudo mv kubectl /usr/local/bin/
adrian@DESKTOP-36BOFP2:~$ kubectl version --client
Client Version: v1.35.0
Kustomize Version: v5.7.1
adrian@DESKTOP-36BOFP2:~$ mkdir -p $HOME/.kube
adrian@DESKTOP-36BOFP2:~$ sudo cp /etc/rancher/k3s/k3s.yaml $HOME/.kube/config
adrian@DESKTOP-36BOFP2:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
adrian@DESKTOP-36BOFP2:~$ sudo chmod 600 $HOME/.kube/config
adrian@DESKTOP-36BOFP2:~$ kubectl get nodes
NAME              STATUS   ROLES           AGE     VERSION
desktop-36bofp2   Ready    control-plane   8m40s   v1.34.3+k3s1
adrian@DESKTOP-36BOFP2:~$
```

## 0.3 – Crear directorio de trabajo

*mkdir -p ~/kubernetes-aws-practice*

*cd ~/kubernetes-aws-practice*

Simplemente he creado un directorio para trabajar con kubernetes

```
adrian@DESKTOP-36BOFP2:~$ mkdir -p ~/kubernetes-aws-practice
adrian@DESKTOP-36BOFP2:~$ cd ~/kubernetes-aws-practice
adrian@DESKTOP-36BOFP2:~/kubernetes-aws-practice$
```

# PARTE 1: CREAR APLICACIÓN SIMPLE PARA KUBERNETES

## 1.1 – Crear carpeta para la aplicación

```
mkdir -p ~/kubernetes-aws-practice/app

cd ~/kubernetes-aws-practice/app
```

## 1.2 – Crear aplicación Python con Flask

Crear app.py:

```
cat > app.py << 'EOF'

#!/usr/bin/env python3

from ffask import Flask, jsonify,

send_from_directory import os

import socket

from datetime import datetime

import sys

app = Flask(__name__)

# Variables de entorno inyectadas por Kubernetes

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

sudo chmod +x app.py
```

Aquí he creado el backend en Flask tal cual. He definido las rutas para la web y la información del pod, asegurándome de que lea las variables de entorno de Kubernetes.

```
  GNU nano 6.2                                                  app.py
#!/usr/bin/env python3
from flask import Flask, jsonify, send_from_directory
import os
import socket
from datetime import datetime
import sys

app = Flask(__name__)

# Variables de entorno inyectadas por Kubernetes
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)
```

## 1.3 – Crear archivo HTML

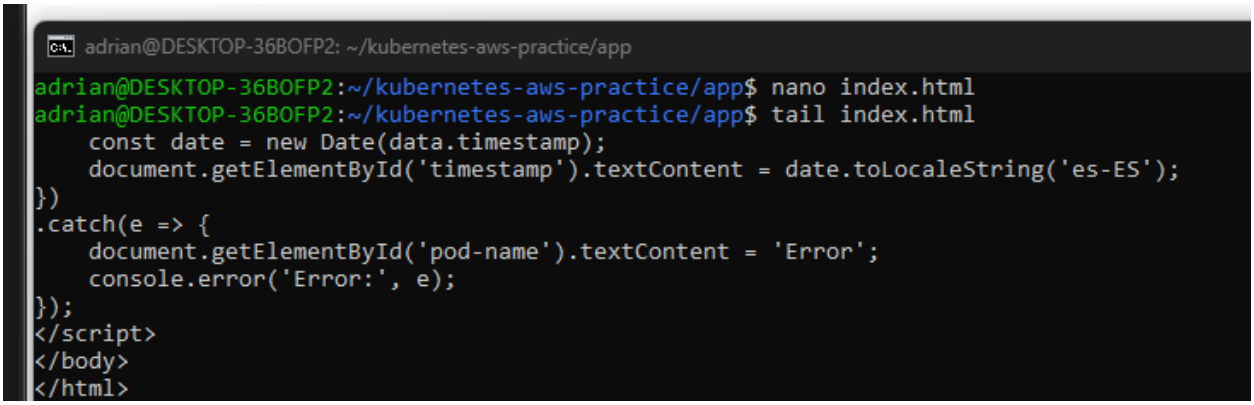
```
cat > index.html << 'EOF'
<!DOCTYPE html>
<html>
<head>
  <title>Kubernetes Load Balancer</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      display: ffex;
      justify-content: center;
      align-items: center;
      min-height: 100vh;
      margin: 0;
```

```
        background: linear-gradient(135deg, #cc7eea 0%, #7c4ba2 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: #cc7eea;
        margin: 0 0 30px 0;
    }
    .info {
        background: #f0f0f0;
        padding: 20px;
        border-radius: 5px;
        margin: 20px 0;
    }
    .pod-name {
        font-size: 28px;
        color: #7c4ba2;
        font-weight: bold;
        font-family: monospace;
    }
    .timestamp {
        color: #ccc;
        font-size: 13px;
        margin-top: 10px;
    }
    .instruction {
```

```
        background: #e0f2fe;
        padding: 15px;
        border-radius: 5px;
        color: #03cSa1;
        font-size: 14px;
        margin-top: 20px;
    }
    .refresh-button {
        margin-top: 20px;
        padding: 10px 20px;
        background: #cc7eea;
        color: white;
        border: none;
        border-radius: 5px;
        cursor: pointer;
        font-size: 14px;
    }
    .refresh-button:hover {
        background: #7c4ba2;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1> Kubernetes Load Balancer</h1>
    <div class="info">
        <p style="margin: 0 0 10px 0; color: #ccc;">Pod atendiendo solicitud:</p>
        <p class="pod-name" id="pod-name">Cargando...</p>
        <p class="timestamp" id="timestamp"></p>
    </div>
    <div class="instruction">
        Actualiza constantemente esta página para ver el <b>balanceo de carga en acción</b>
```

```
    </div>
    <button class="refresh-button" onclick="location.reload()">Actualizar Ahora</button>
  </div>
  <script>
    fetch('/pod-info')
      .then(r => r.json())
      .then(data => {
        document.getElementById('pod-name').textContent = data.pod_name;
        const date = new Date(data.timestamp);
        document.getElementById('timestamp').textContent =
          date.toLocaleString('es-ES');
      })
      .catch(e => {
        document.getElementById('pod-name').textContent = 'Error';
        console.error('Error:', e);
      });
  </script>
</body>
</html>
EOF
```

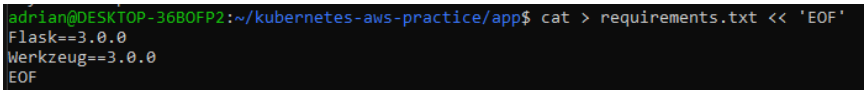
Aquí he creado el frontend de la aplicación. Es básicamente una página web sencilla que consulta al backend y muestra en grande el nombre del pod, lo cual es básico para comprobar visualmente que el balanceo de carga funciona bien al recargar la página.



## 1.4 – Crear requirements.txt

*cat > requirements.txt << 'EOF' Flask==3.0.0* Werkzeug==3.0.0 EOF

Aquí he definido las dependencias exactas (Flask y Werkzeug) básicamente para que la aplicación arranque sin errores de compatibilidad.



## 1.5 – Crear Dockerfile ultraligero

Nota, este Dockerfile es solo para construcción local. k3s lo ejecutará directamente:

*cat > Dockerfile << 'EOF'*

*FROM python:3.11-slim*

*WORKDIR /app*

*# Copiar archivos*

*COPY requirements.txt*

*. COPY app.py .*

*COPY index.html .*

*# Instalar dependencias*

*RUN pip install --no-cache-dir -r requirements.txt*

*# Ejecutar app*

*CMD ["python", "app.py"]*

*EOF*

He creado el Dockerfile para dejar claro como se empaquetaría la aplicación en una imagen de contenedor estándar.

```
adrian@DESKTOP-36BOFP2:~/kubernetes-aws-practice/app$ cat > Dockerfile << 'EOF'
FROM python:3.11-slim
WORKDIR /app
# Copiar archivos
COPY requirements.txt .
COPY app.py .
COPY index.html .
# Instalar dependencias
RUN pip install --no-cache-dir -r requirements.txt
# Ejecutar app
CMD ["python", "app.py"]
EOF
```

# PARTE 2: CONFIGURAR KUBERNETES (*MANIFESTS YAML*)

## 2.1 – Crear Namespace

*cd ~/kubernetes-aws-practice*

*cat > namespace.yaml <<*

*'EOF' apiVersion: v1*

*kind:*

*Namespace*

*metadata:*

*name: load-balancer-demo*

*labels:*

*name: load-balancer-demo*

*EOF*

*# Aplicar*

*kubectl apply -f namespace.yaml*

*# Verificar*

*kubectl get namespaces*

Aqui he creado un Namespace. Básicamente es como crear una carpeta virtual dentro de Kubernetes para meter ahí todas mis cosas (pods, servicios entre otras cosas) y que no se mezclen con lo demas.

```
adrian@DESKTOP-36BOFP2:~/kubernetes-aws-practice/app$ cd ~/kubernetes-aws-practice
cat > namespace.yaml << 'EOF'
apiVersion: v1
kind: Namespace
metadata:
 name: load-balancer-demo
 labels:
 name: load-balancer-demo
EOF
adrian@DESKTOP-36BOFP2:~/kubernetes-aws-practice$ kubectl apply -f namespace.yaml
namespace/load-balancer-demo created
adrian@DESKTOP-36BOFP2:~/kubernetes-aws-practice$ kubectl get namespaces
NAME                STATUS   AGE
default             Active   3d20h
kube-node-lease     Active   3d20h
kube-public         Active   3d20h
kube-system         Active   3d20h
load-balancer-demo  Active   4s
```

## 2.2 – Crear ConfigMap con archivos de la app

```
cat > configmap.yaml << 'EOF'
apiVersion: v1
kind: ConfigMap
metadata:
 name: app-files
 namespace: load-balancer-demo
data:
 requirements.txt: |
  Flask==3.0.0
  Werkzeug==3.0.0

 app.py: |
  #!/usr/bin/env python3
  from ffask 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)

  index.html: |
    <!DOCTYPE html>
    <html>
    <head>
        <title>Kubernetes Load Balancer</title>
        <style>
            body {
                font-family: Arial, sans-serif;
                display: ffex;
                justify-content: center;
                align-items: center;
                min-height: 100vh;
                margin: 0;
                background: linear-gradient(135deg, #cc7eea 0%, #7c4ba2 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: #cc7eea;
            margin: 0 0 30px 0;
        }
        .info {
            background: #f0f0f0;
            padding: 20px;
            border-radius: 5px;
            margin: 20px 0;
        }
        .pod-name {
            font-size: 28px;
            color: #7c4ba2;
            font-weight: bold;
            font-family: monospace;
        }
        .timestamp {
            color: #ccc;
            font-size: 13px;
            margin-top: 10px;
        }
        .instruction {
            background: #e0f2fe;
            padding: 15px;
            border-radius: 5px;
            color: #03cSa1;
            font-size: 14px;
```

```
            margin-top: 20px;
        }
        .refresh-button {
            margin-top: 20px;
            padding: 10px 20px;
            background: #cc7eea;
            color: white;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            font-size: 14px;
        }
        .refresh-button:hover {
            background: #7c4ba2;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Kubernetes Load Balancer</h1>
        <div class="info">
            <p style="margin: 0 0 10px 0; color: #ccc;">Pod atendiendo solicitud:</p>
            <p class="pod-name" id="pod-name">Cargando...</p>
            <p class="timestamp" id="timestamp"></p>
        </div>
        <div class="instruction">
            Actualiza constantemente esta página para ver el <b>balanceo de carga en acción</b>
        </div>
        <button class="refresh-button" onclick="location.reload()">Actualizar Ahora</button>
    </div>
    <script>
        fetch('/pod-info')
```

```
        .then(r => r.json())
        .then(data => {
          document.getElementById('pod-name').textContent = data.pod_name;
          const date = new Date(data.timestamp);
          document.getElementById('timestamp').textContent =
            date.toLocaleString('es-ES');
        })
        .catch(e => {
          document.getElementById('pod-name').textContent = ' Error';
          console.error('Error:', e);
        });
    </script>
  </body>
  </html>
EOF

# Aplicar
kubectl apply -f configmap.yaml
# Verificar
kubectl get configmap -n load-balancer-demo
```

Aquí he creado el configmap.yaml con nano y lo he aplicado al namespace. He comprobado que el ConfigMap app-files se ha actualizado y que contiene los 3 archivos de la aplicación.

```
adrian@DESKTOP-36BOFP2:~/kubernetes-aws-practice$ cd ~/kubernetes-aws-practice
adrian@DESKTOP-36BOFP2:~/kubernetes-aws-practice$ nano configmap.yaml
adrian@DESKTOP-36BOFP2:~/kubernetes-aws-practice$ kubectl apply -f configmap.yaml
configmap/app-files configured
adrian@DESKTOP-36BOFP2:~/kubernetes-aws-practice$ kubectl get configmap -n load-balancer-demo
NAME               DATA   AGE
app-files          3      30h
kube-root-ca.crt   1      31h
adrian@DESKTOP-36BOFP2:~/kubernetes-aws-practice$
```

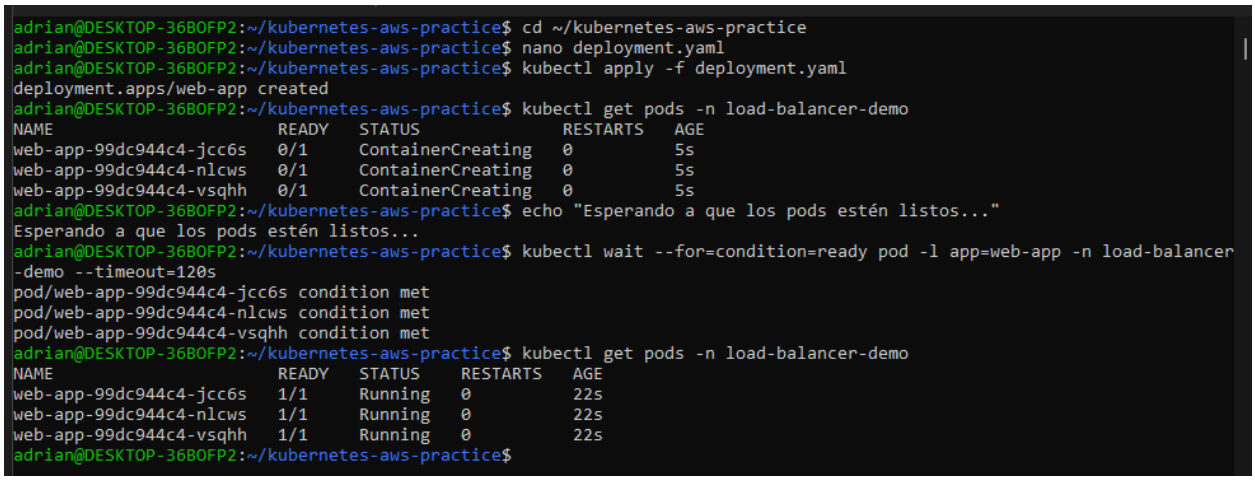
## 2.3 – Crear Deployment con 3 replicas

```
cat > deployment.yaml <<
'EOF' apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
```

```
  namespace: load-balancer-demo
  labels:
    app: web-app
spec:

  replicas: 3

  selector:
    matchLabels:
      app: web-app
  template:
    metadata:
      labels:
        app: web-app
    spec:
      containers:
      - name: web-app
        image: python:3.11-slim
        command: ["sh", "-c"]
        args:
          - |
            cd /app
            pip install --no-cache-dir -r requirements.txt > /dev/null 2>&1
            python app.py
        ports:
        - containerPort: 5000
          protocol: TCP
        env:
        - name: POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
```

```
          fieldPath: metadata.namespace
      volumeMounts:
- name: app-volume
      mountPath: /app
      livenessProbe:
        httpGet:
          path: /health
          port: 5000
        initialDelaySeconds: 15
        periodSeconds: 10
      readinessProbe:
        httpGet:
          path: /health
          port: 5000
        initialDelaySeconds: 5
        periodSeconds: 5
    volumes:
    - name: app-volume
      configMap:
        name: app-files
        defaultMode: 0755
EOF

# Aplicar deployment
kubectl apply -f deployment.yaml
# Verificar que los pods se están
creando kubectl get pods -n load-
balancer-demo
# Esperar a que estén ready (puede tardar 30-c0 segundos)
echo "Esperando a que los pods estén listos..."
kubectl wait --for=condition=ready pod -l app=web-app -n load-balancer-demo --timeout=120s
# Verificar
kubectl get pods -n load-balancer-demo
```

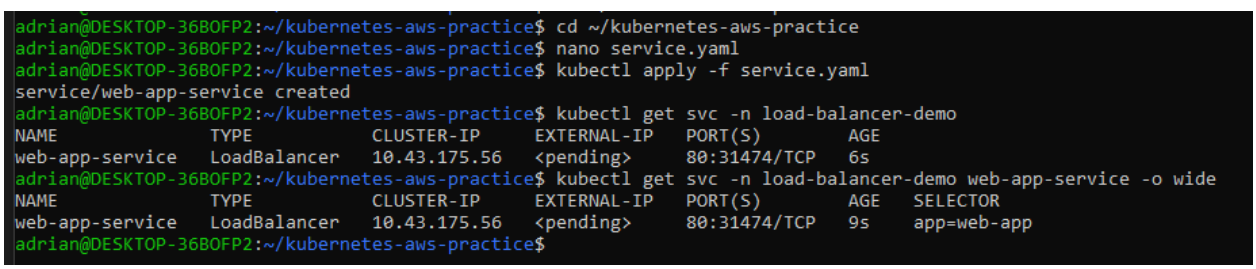
Aquí he creado el deployment.yaml con 3 réplicas, montando el ConfigMap como volumen en /app. Lo he aplicado y he esperado a que los pods salgan en Ready.



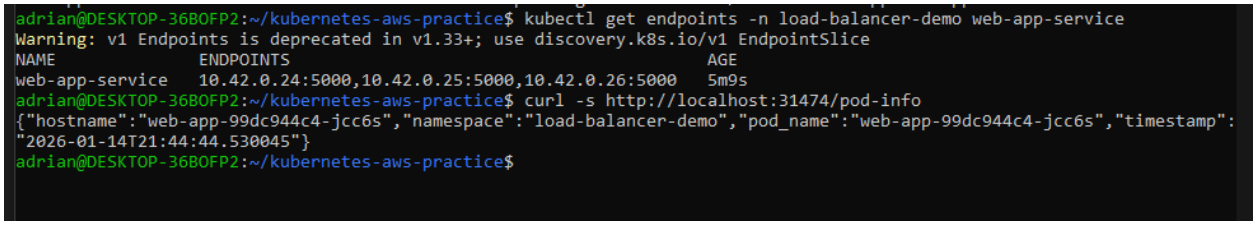
## 2.4 – Crear Service (*Load Balancer*)

```
cat > service.yaml << 'EOF'
apiVersion: v1
kind: Service
metadata:
  name: web-app-service
  namespace: load-balancer-demo
  labels:
    app: web-app
spec:
  type: LoadBalancer
  selector:
    app: web-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 5000
    name: http
  sessionAffinity: None
EOF


# Aplicar service
kubectl apply -f service.yaml
# Verificar el service
kubectl get svc -n load-balancer-demo
# Obtener IP del service
kubectl get svc -n load-balancer-demo web-app-service -o wide
```

Aquí he creado el service.yaml tipo LoadBalancer para exponer la app. Lo he aplicado en el namespace y he comprobado el servicio y su IP con el -o wide.



Aquí he comprobado que el Service realmente está repartiendo a los 3 pods viendo los endpoints. Luego he hecho un curl al NodePort y me ha devuelto el JSON con el nombre del pod, así que está funcionando.

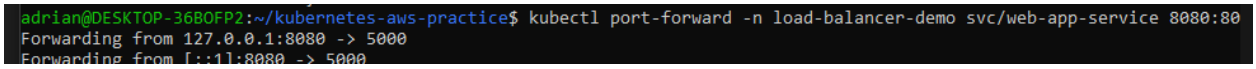



# PARTE 3: VERIFICAR BALANCEO DE CARGA LOCAL

## 3.1 – Levantar la aplicación

*# Opción 1: Port-forward (recomendado para WSL2)*

*kubectl port-forward -n load-balancer-demo svc/web-app-service 8080:80*
Aquí he levantado la aplicación con port-forward para exponer el service en el puerto 8080 y poder acceder desde el navegador.




## 3.2 – Probar balanceo con curl

*# Script para hacer peticiones y ver qué pod*

*responde for i in {1..10}; do*

*echo "Petición :i:"*

*curl -s http://localhost:8080/pod-info | python3 -m json.tool | grep pod_name*

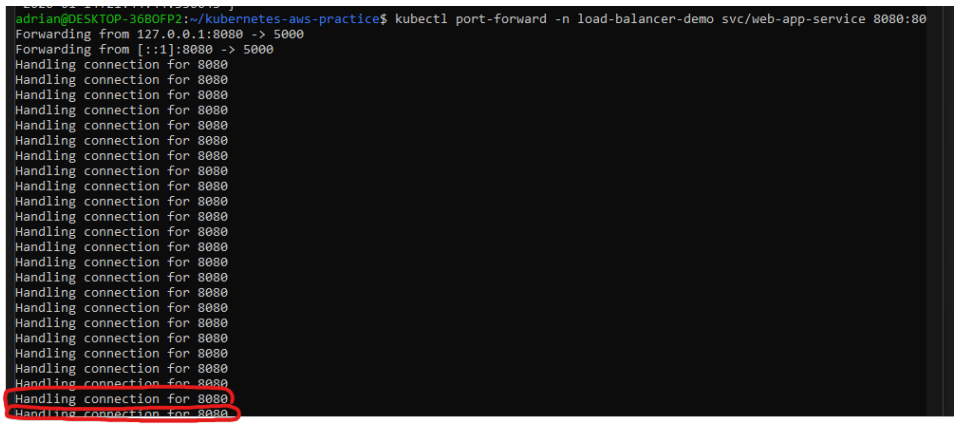
*sleep 1*

*done*

*# Deberías ver el mismo pod respondiendo varias veces lo*

*siguiente: # "pod_name": "web-app-xxxxx-11111"*

Aquí he hecho 10 peticiones seguidas con curl al /pod-info y he sacado el pod_name para ver qué pod está respondiendo.



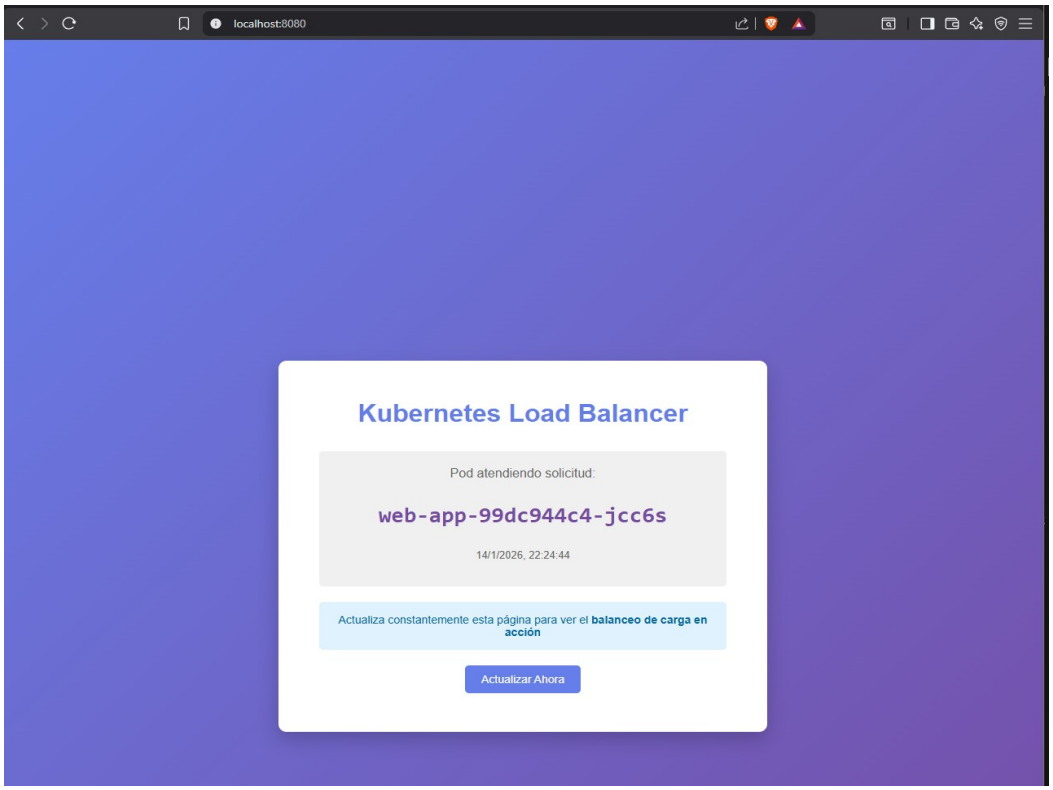
```
adrian@DESKTOP-36BOFP2:~/kubernetes-aws-practice$ for i in {1..20}; do
  curl -s -H "Connection: close" http://localhost:8080/pod-info | grep pod_name
  sleep 1
done
{"hostname":"web-app-99dc944c4-nlcws","namespace":"load-balancer-demo","pod_name":"web-app-99dc944c4-nlcws","timest
amp":"2026-01-15T21:58:59.172196"}
{"hostname":"web-app-99dc944c4-jcc6s","namespace":"load-balancer-demo","pod_name":"web-app-99dc944c4-jcc6s","timest
amp":"2026-01-15T21:59:00.178449"}
{"hostname":"web-app-99dc944c4-nlcws","namespace":"load-balancer-demo","pod_name":"web-app-99dc944c4-nlcws","timest
amp":"2026-01-15T21:59:01.184341"}
{"hostname":"web-app-99dc944c4-jcc6s","namespace":"load-balancer-demo","pod_name":"web-app-99dc944c4-jcc6s","timest
amp":"2026-01-15T21:59:02.189939"}
{"hostname":"web-app-99dc944c4-jcc6s","namespace":"load-balancer-demo","pod_name":"web-app-99dc944c4-jcc6s","timest
amp":"2026-01-15T21:59:03.194915"}
^C
```

## 3.3 – Verificar en navegador

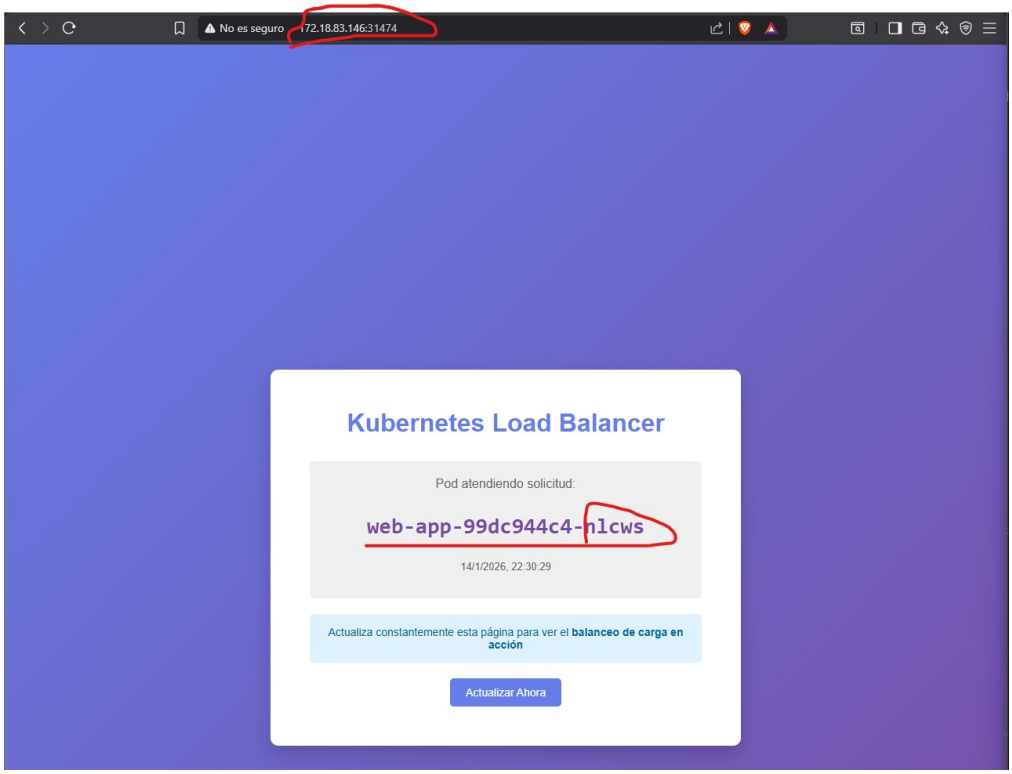
Abre http://localhost:8080 en tu navegador y actualiza varias veces. Verás que cambia el pod que atiende.

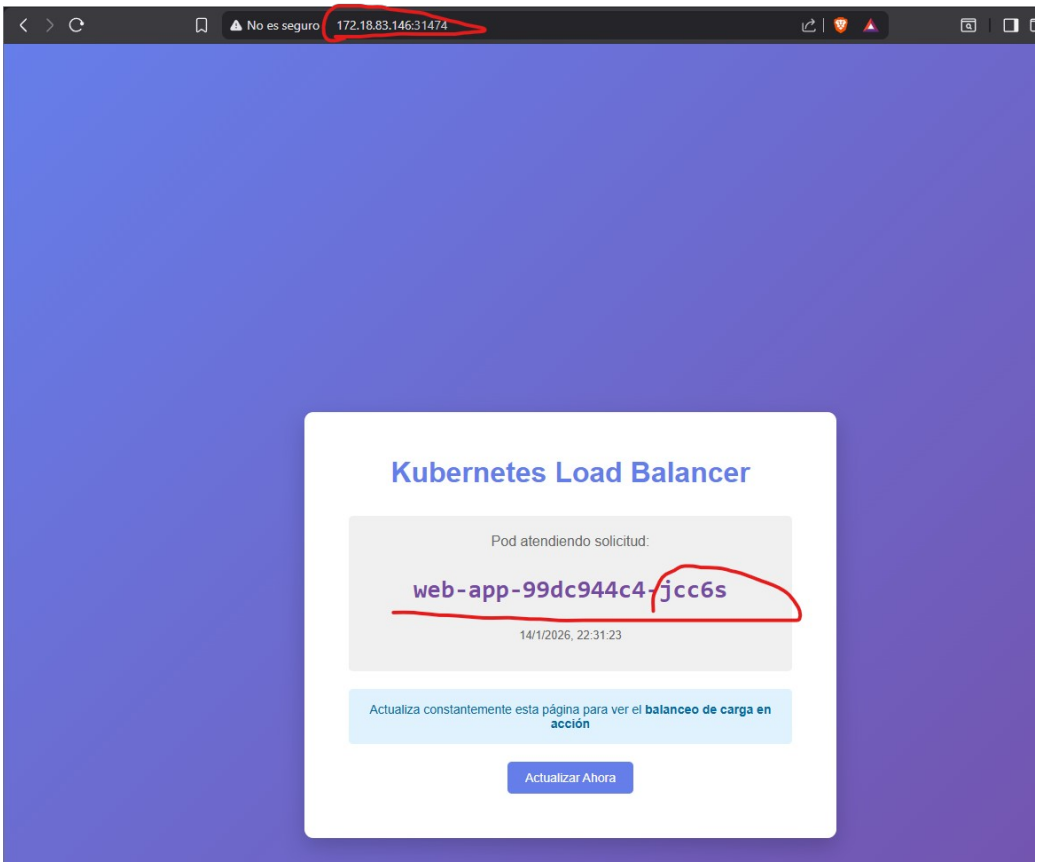
---

Aquí he abierto http://localhost:8080 en el navegador con el port-forward y he comprobado que la web carga y muestra el pod que está atendiendo la petición.



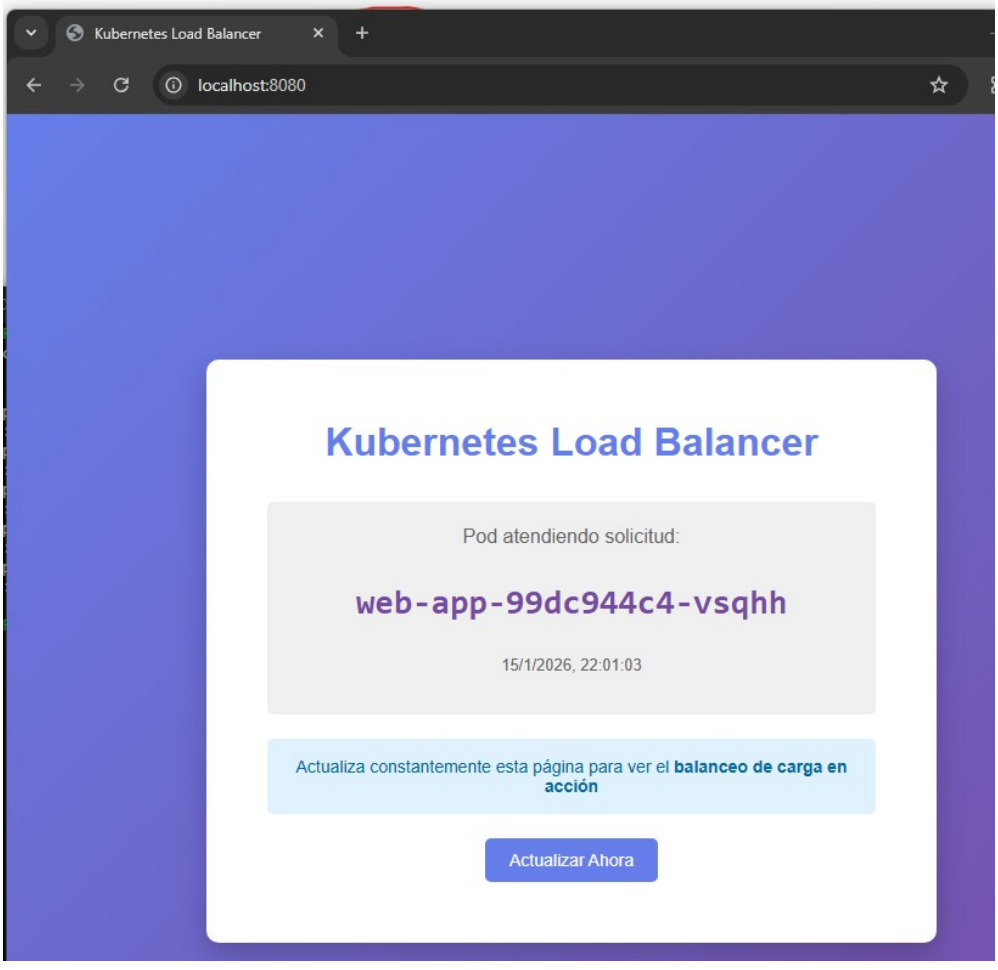

Aquí he entrado desde el navegador a http://172.18.83.146:31474 (NodePort) y al actualizar varias veces se ve cómo va cambiando el pod que responde (por ejemplo ...-nlcws y luego ...-jcc6s), comprobando el balanceo de carga.

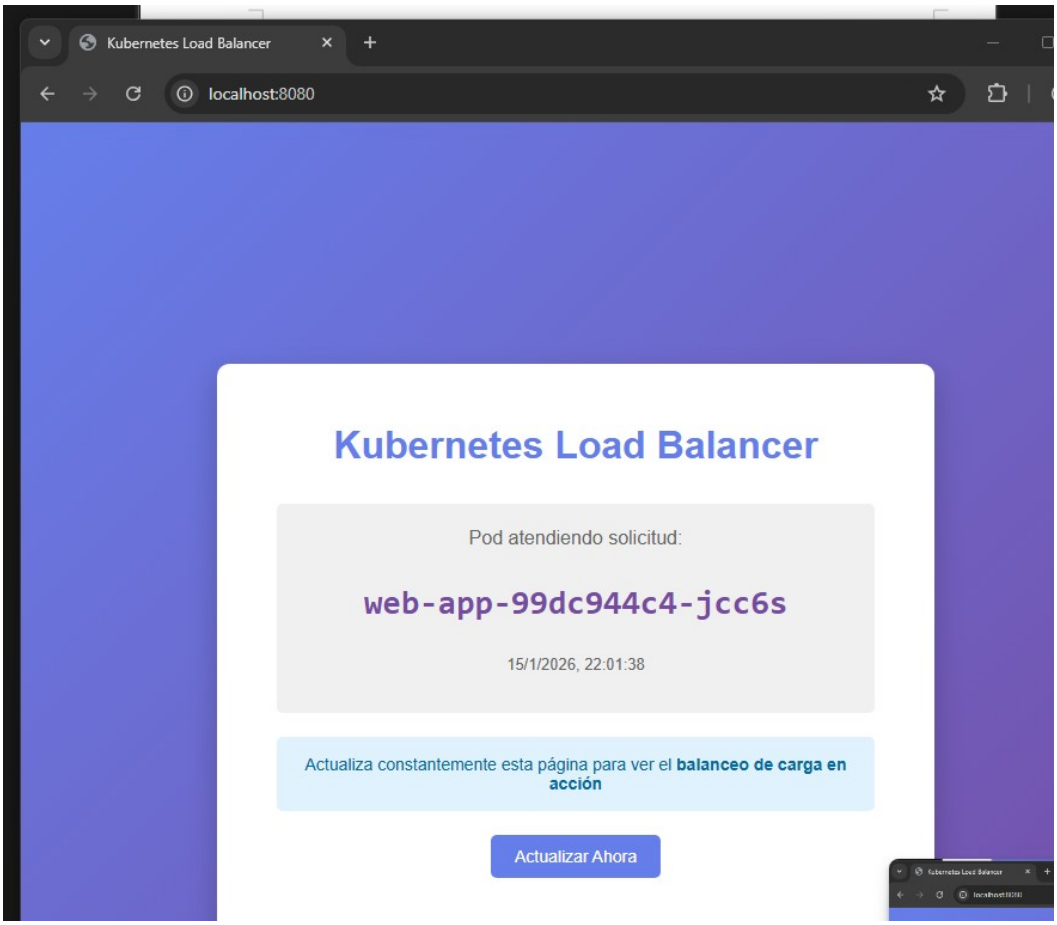

No es seguro
172.18.83.146:31474
Kubernetes Load Balancer
Pod atendiendo solicitud:
web-app-99dc944c4-n1cws
14/1/2026, 22:30:29
Actualiza constantemente esta página para ver el balanceo de carga en acción
Actualizar Ahora

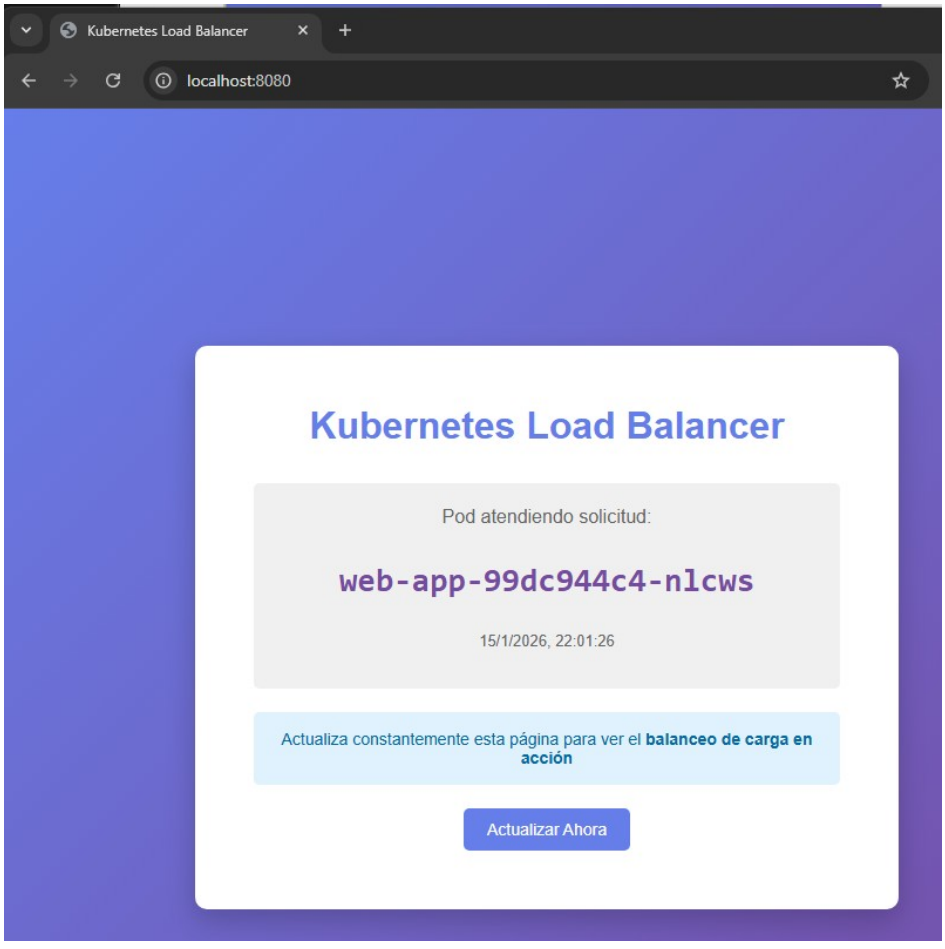

No es seguro
172.18.83.146:31474
Kubernetes Load Balancer
Pod atendiendo solicitud:
web-app-99dc944c4-jcc6s
14/1/2026, 22:31:23
Actualiza constantemente esta página para ver el balanceo de carga en acción
Actualizar Ahora

```
adrian@DESKTOP-36BOFP2:/mnt/c/Windows/System32$ for i in {1..10}; do
echo "Petición $i:"
curl -s http://172.18.83.146:31474/pod-info | python3 -m json.tool | grep pod_name
sleep 1
done
Petición 1:
    "pod_name": "web-app-99dc944c4-jcc6s",
Petición 2:
    "pod_name": "web-app-99dc944c4-jcc6s",
Petición 3:
    "pod_name": "web-app-99dc944c4-vsqhh",
Petición 4:
    "pod_name": "web-app-99dc944c4-nlcws",
Petición 5:
    "pod_name": "web-app-99dc944c4-vsqhh",
Petición 6:
    "pod_name": "web-app-99dc944c4-jcc6s",
Petición 7:
    "pod_name": "web-app-99dc944c4-vsqhh",
Petición 8:
    "pod_name": "web-app-99dc944c4-nlcws",
Petición 9:
    "pod_name": "web-app-99dc944c4-jcc6s",
Petición 10:
    "pod_name": "web-app-99dc944c4-jcc6s",
adrian@DESKTOP-36BOFP2:/mnt/c/Windows/System32$
```

## ESTOS 3 DE ABAJO SON LOS BUENOS (PUERTO 8080 CON BALANCEO)




Kubernetes Load Balancer
localhost:8080
Kubernetes Load Balancer
Pod atendiendo solicitud:
web-app-99dc944c4-jcc6s
15/1/2026, 22:01:38
Actualiza constantemente esta página para ver el balanceo de carga en acción
Actualizar Ahora


Kubernetes Load Balancer
localhost:8080
Kubernetes Load Balancer
Pod atendiendo solicitud:
web-app-99dc944c4-nlcws
15/1/2026, 22:01:26
Actualiza constantemente esta página para ver el balanceo de carga en acción
Actualizar Ahora

# PARTE 4: CONFIGURACIÓN EN AWS

## 4.1 – Crear Security Group

En AWS Console:

1. **Accede a EC2:** Security Groups
2. Haz clic en "Create security group"
3. **Nombre:** kubernetes-aws-sg
4. **Descripción:** Security group para Kubernetes con AWS
5. Agregar reglas de entrada:

| Tipo | Protocolo | Puerto | Origen |
|---|---|---|---|
| **SSH** | **TCP** | 22 | 0.0.0.0/0 |
| **HTTP** | **TCP** | 80 | 0.0.0.0/0 |
| **HTTPS** | **TCP** | 443 | 0.0.0.0/0 |

## 4.2 – Crear instancia EC2

En AWS Console:

1. **EC2** → Instances → Launch instances
2. **Nombre:** kubernetes-test-server
3. **AMI:** Ubuntu 24.04 LTS
4. **Tipo:** t3.micro (*Free Tier*)
5. **Key pair:** Tu clave SSH
6. **VPC:** Default
7. **Security group:** kubernetes-aws-sg
8. **Storage:** 8 GiB, gp3
9. Launch instance

## 4.3 – Conectar a EC2

*# Obtener IP pública desde AWS Console (ej:*

*54.123.45.c7) # Conectar via SSH*

*ssh -i ~/.ssh/tu-clave-aws.pem ubuntu@54.123.45.c7*

*# En EC2, instalar herramientas*

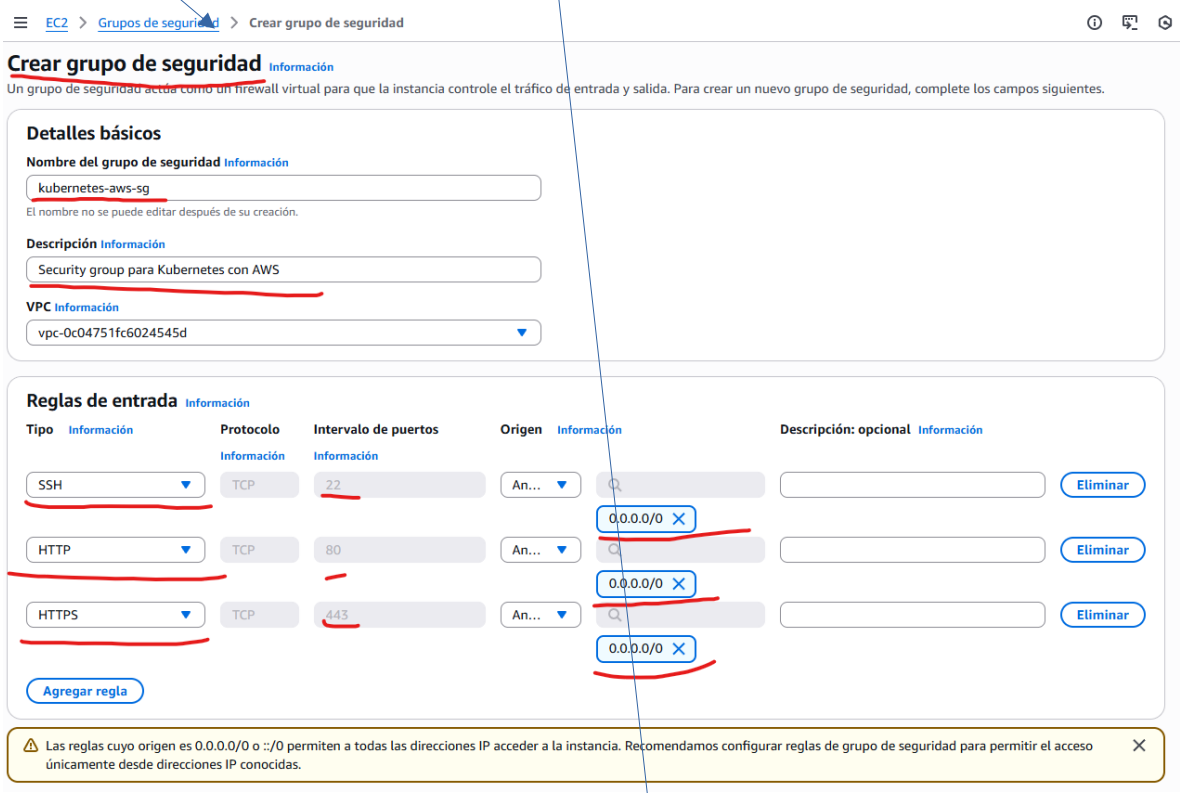
*sudo apt update*

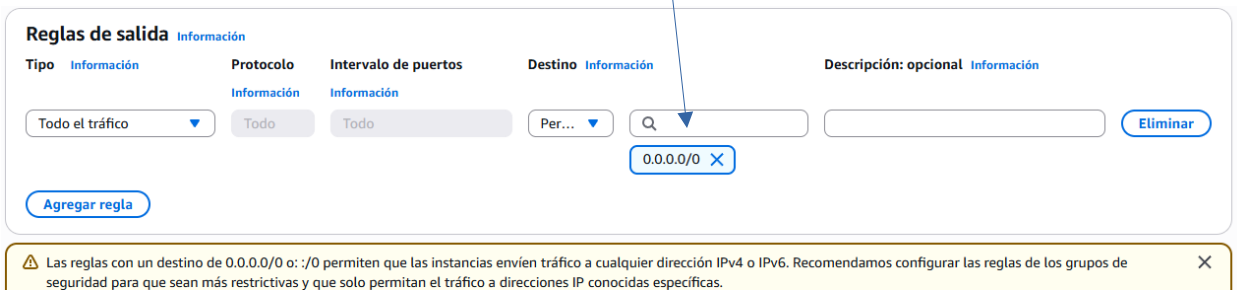
*sudo apt install -y curl wget python3 python3-pip*

*git # Crear directorio de trabajo*

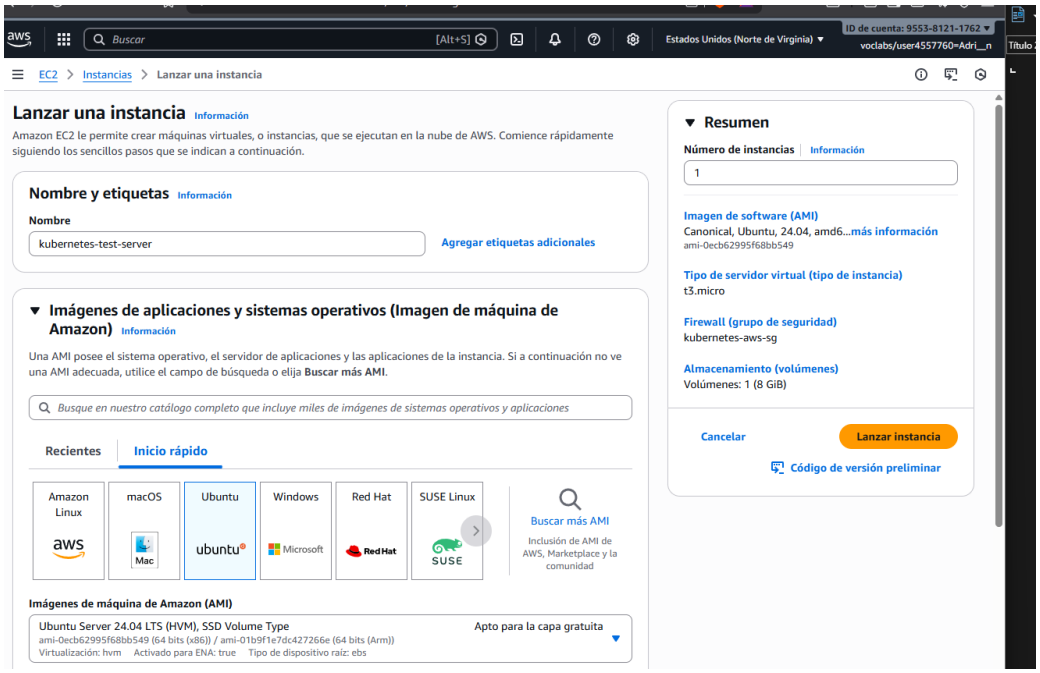
*mkdir -p ~/kubernetes-test*

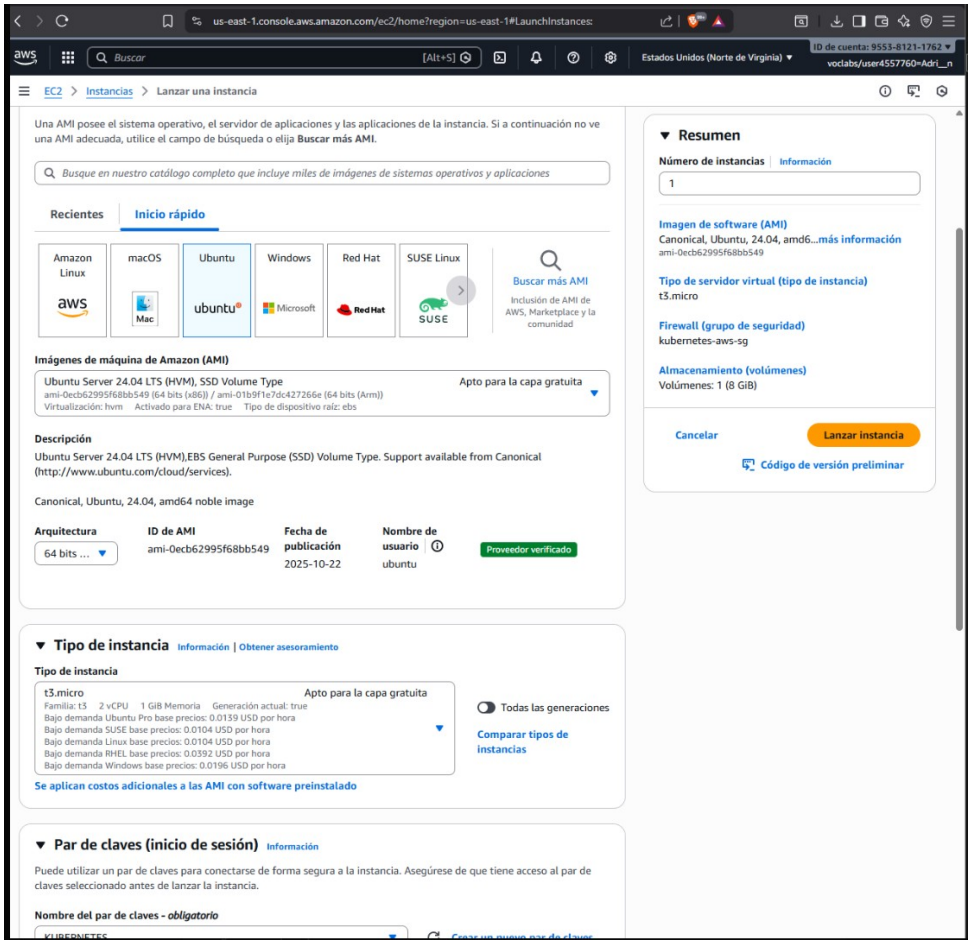
Aquí he creado el Security Group kubernetes-aws-sg y he configurado las reglas de entrada SSH (22), HTTP (80) y HTTPS (443) permitidas desde 0.0.0.0/0.

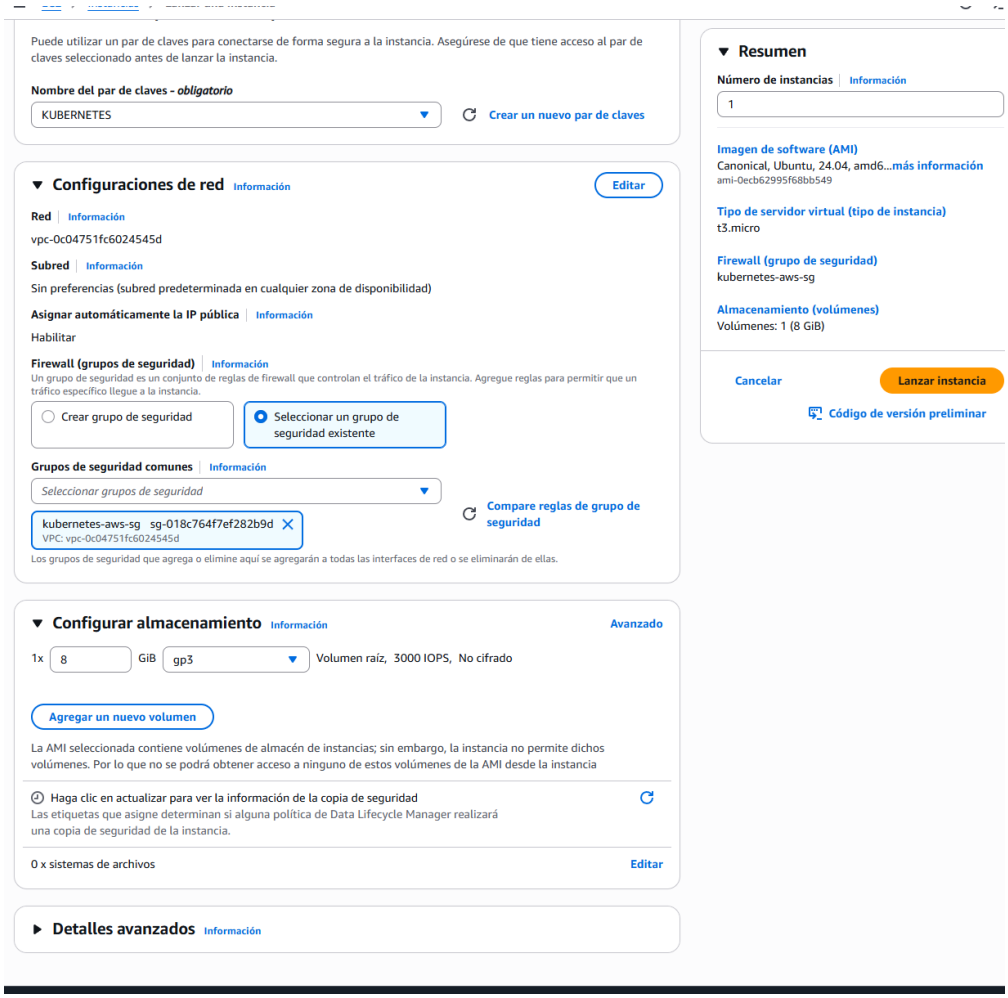


Aquí en las 3 capturas de abajo he preparado la instancia EC2 con nombre kubernetes-test-server, usando Ubuntu Server 24.04 LTS, tipo t3.micro (Free Tier), seleccionando mi key pair KUBERNETES y asignando el Security Group kubernetes-aws-sg. También he dejado el almacenamiento en 8 GiB gp3 y la IP pública habilitada para poder conectarme desde fuera.






# PARTE 5: CONECTAR KUBERNETES LOCAL CON AWS EC2

## 5.1 – Crear túnel SSH que expone el LoadBalancer

En una nueva máquina local (*WSL2*), mantén esta terminal abierta, el túnel estará activo mientras esté abierta:

```
# Reemplaza 54.123.45.c7 con tu IP de
EC2 ssh -i ~/.ssh/tu-clave-aws.pem \
  -N -R 8888:localhost:8080 \
  ubuntu@54.123.45.c7
# -N: No ejecutar comandos remotos
# -R 8888:localhost:8080: Redirige puerto 8888 en EC2 a puerto 8080 local
```

En AWS EC2 (*otra terminal local*):

*# Conecta a la instancia en otra terminal*

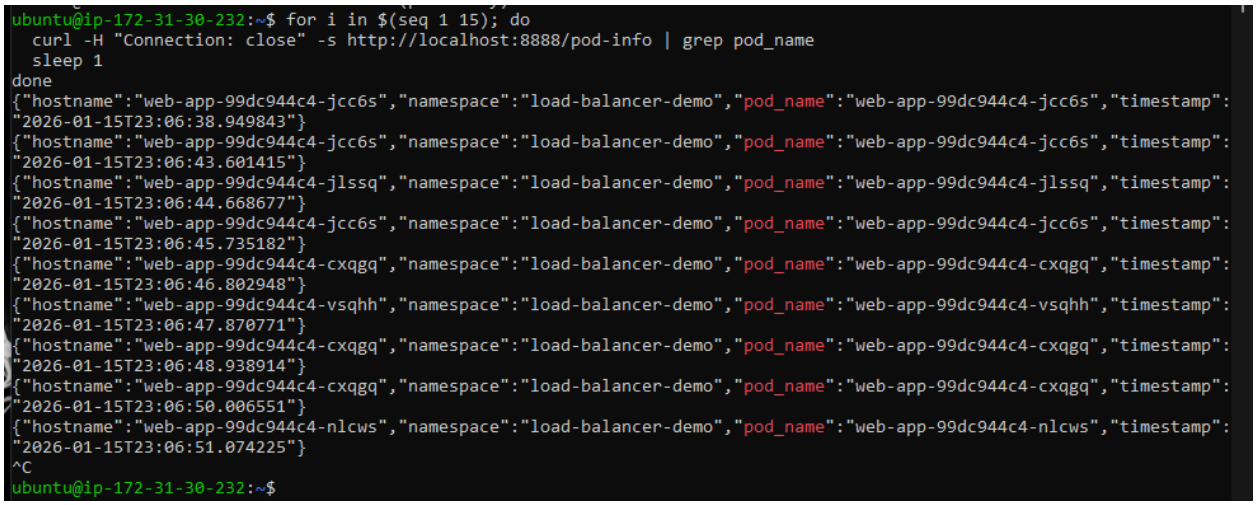
*ssh -i ~/.ssh/tu-clave-aws.pem ubuntu@54.123.45.c7*

*# Verificar que el túnel funciona*

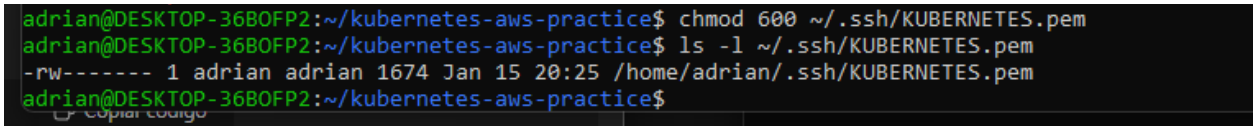
*curl -s http://localhost:8888/pod-info*

*# Deberías recibir JSON:*

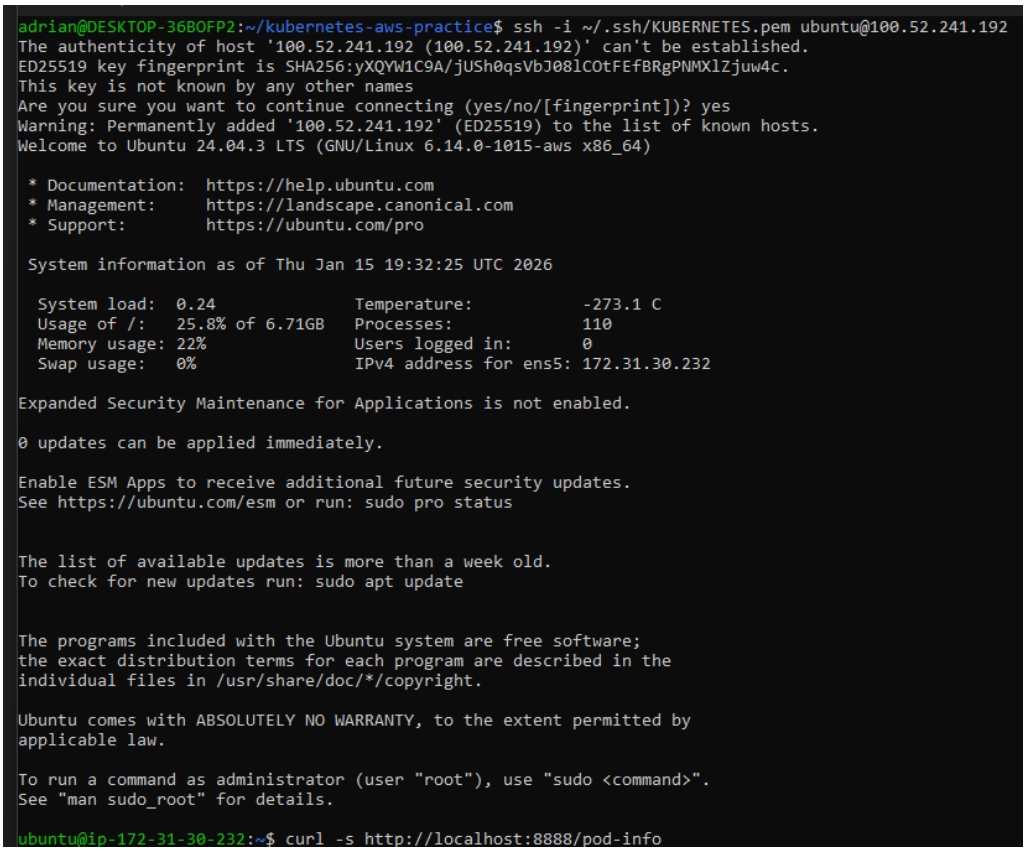
*# {"pod_name":"web-app-xxxxx-11111", "namespace":"load-balancer-demo", ...}*



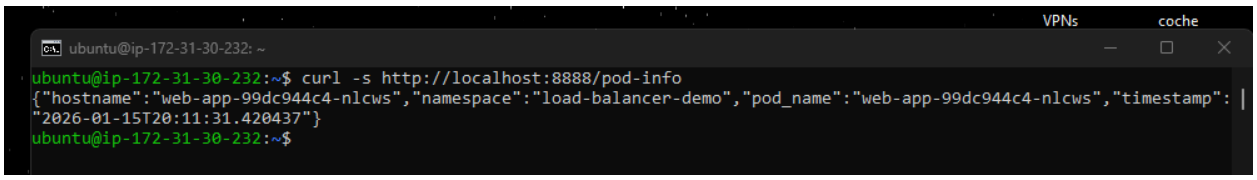

Lo primero que he hecho ha sido lo siguiente: He copiado el .pem de kubernetes que he creado antes y le he dado permisos.



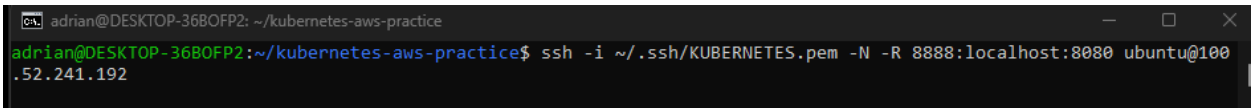
Aquí he abierto el túnel SSH hacia la EC2 con -N -R 8888:localhost:8080, dejando esta terminal abierta para mantener el túnel activo.



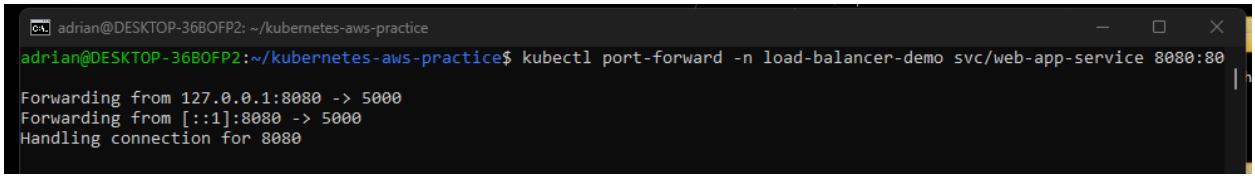

Aquí he comprobado desde la propia EC2 que el túnel funciona: hago curl a http://localhost:8888/pod-info y me devuelve el JSON con el pod_name del pod que está atendiendo.



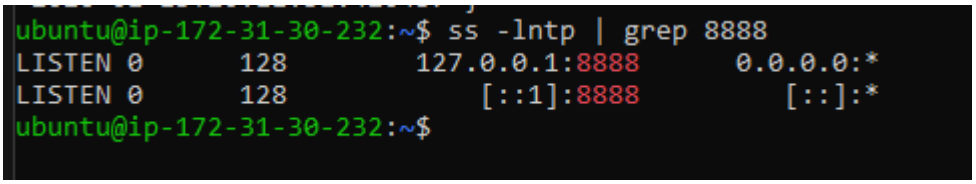

Aquí he creado el túnel SSH hacia la EC2 con -N -R 8888:localhost:8080, y dejo esta terminal abierta para que el túnel se mantenga activo.



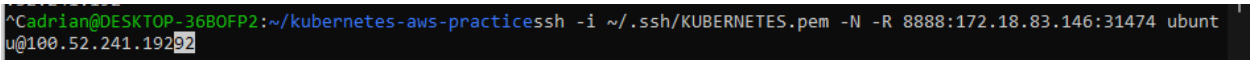
Aquí he levantado el kubectl port-forward del servicio web-app-service en el namespace load-balancer-demo, exponiéndolo en mi máquina local por el puerto 8080.

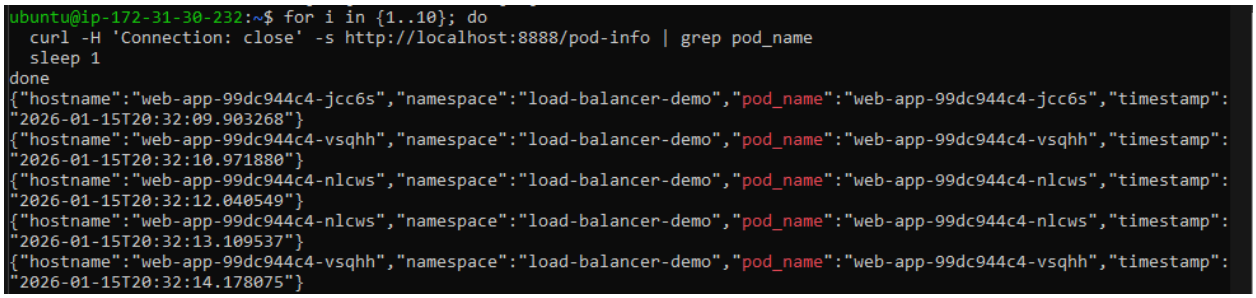



Aquí se ve que en la EC2 el puerto 8888 está en escucha, confirmando que el puerto remoto se ha abierto correctamente gracias al túnel.




Aquí abajo se puede ver que he creado el túnel SSH desde mi WSL hacia la instancia EC2 con -N -R 8888:172.18.83.146:31474, dejando la terminal abierta para que el túnel siga activo. Después, desde la propia EC2 he lanzado varias peticiones a http://localhost:8888/pod-info y se ve cómo cambia el pod_name, confirmando que el acceso funciona y que el servicio está balanceando entre pods. (ESTO ES COMO APORTACION ADICIONAL)




## 5.2 – Crear script de prueba en EC2

En la instancia EC2:

*cat > ~/test-kubernetes-lb.sh << 'EOF'*

*#!/bin/bash*

*echo "=== Prueba Kubernetes Load Balancer desde AWS ==="*

*echo ""*

*declare -A*

*pods_count for i in*

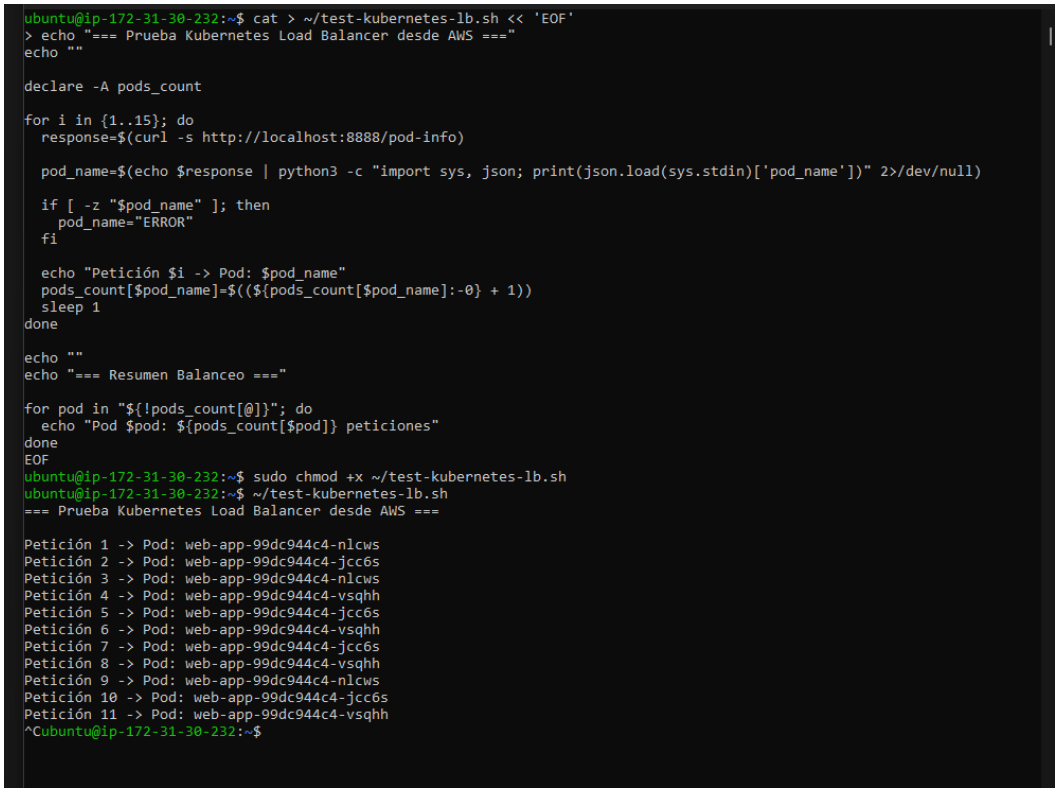
*{1..15}; do*

*response=:(curl -s http://localhost:8888/pod-info)*

```
  pod_name=:(echo :response | python3 -c "import sys, json; print(json.load(sys.stdin)
['pod_name'])" 2>/dev/null)

  it [ -z ":pod_name" ]; then
```

```
    pod_name="ERROR"
  fi
  echo "Petición :i → Pod: :pod_name"
  pods_count[:pod_name]=:((:{pods_count[:pod_name]:-0} + 1))
  sleep 1
done
echo ""
echo "=== Resumen Balanceo ==="
for pod in ":{!pods_count[@]}"; do
  echo "Pod :pod: :{pods_count[:pod]} peticiones"
done
EOF

sudo chmod +x ~/test-kubernetes-lb.sh
# Ejecutar prueba
~/test-kubernetes-lb.sh
```

Aquí he creado y ejecutado en la EC2 el script test-kubernetes-lb.sh, que hace varias peticiones a http://localhost:8888/pod-info. En la salida se ve cómo va cambiando el pod_name, confirmando que el balanceo funciona.



# PARTE 6: ESCALADO DINÁMICO

## 6.1 – Escalar a 5 replicas

En tu máquina local:

*# Aumentar a 5 replicas*

*kubectl scale deployment web-app -n load-balancer-demo --replicas=5*

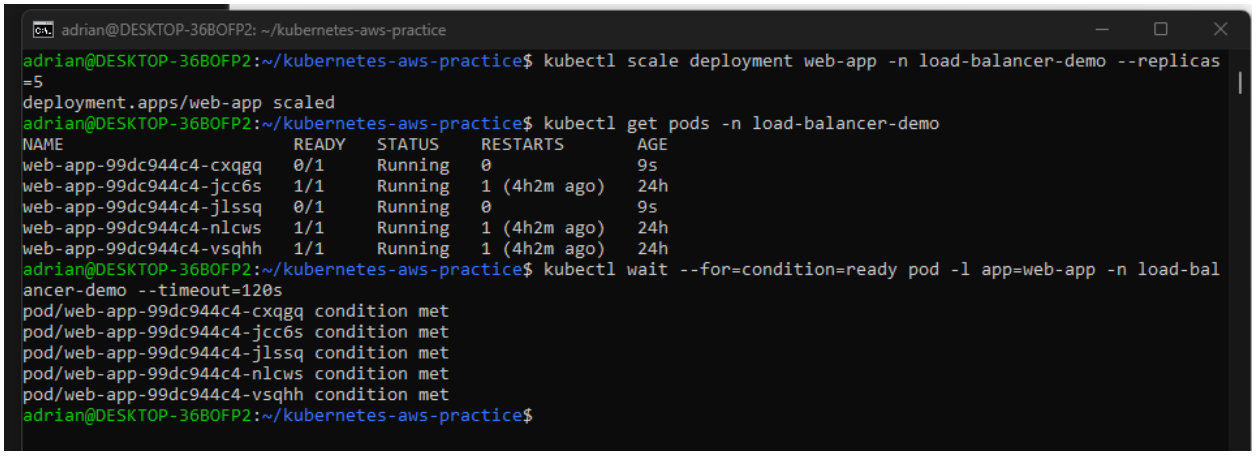
*# Verificar*

*kubectl get pods -n load-balancer-demo*

*# Esperar a que estén ready*

*kubectl wait --for=condition=ready pod -l app=web-app -n load-balancer-demo –timeout=120s*

Aquí he escalado el deployment web-app del namespace load-balancer-demo a 5 réplicas con kubectl scale. Después he listado los pods con kubectl get pods para comprobar que ya aparecen los 5, y por último he usado kubectl wait --for=condition=ready para asegurarme de que todos han quedado en estado Ready dentro del tiempo indicado.



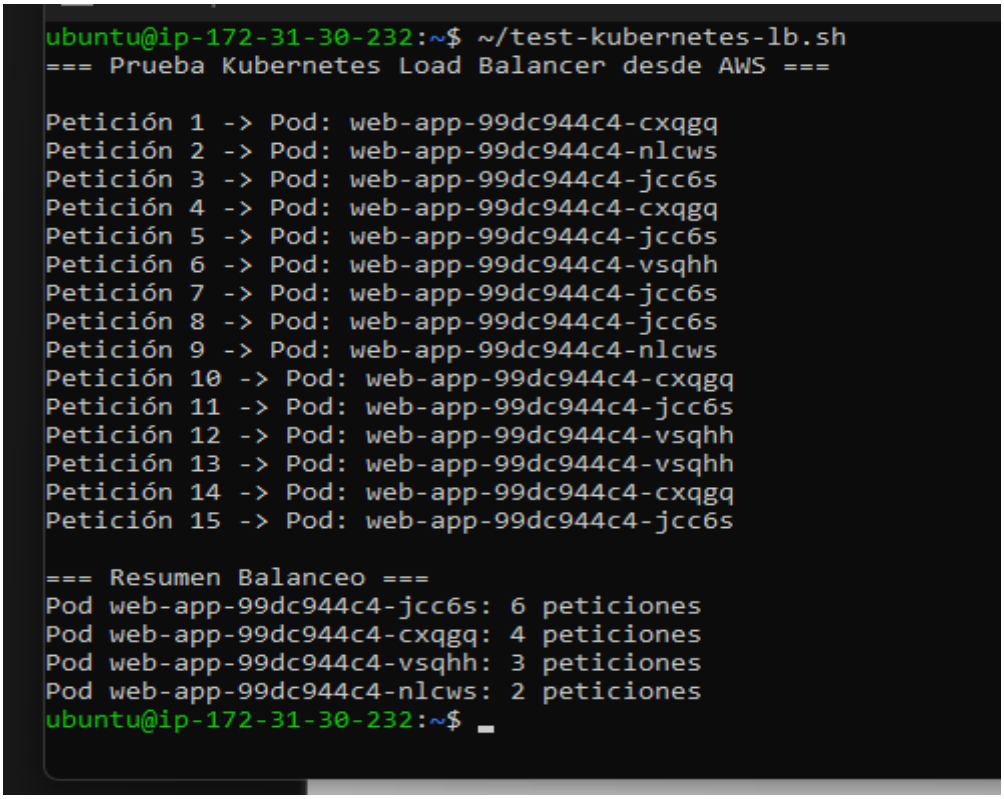
## 6.2 – Probar balanceo desde AWS

En EC2:

*~/test-kubernetes-lb.sh*

*# Hay más variedad de pods, pero solo va aparecer uno*

Aquí he ejecutado el script test-kubernetes-lb.sh desde la EC2 para lanzar varias peticiones a http://localhost:8888/pod-info. Se ve cómo va cambiando el pod que responde y al final me saca un resumen con el número de peticiones atendidas por cada pod, confirmando el balanceo.




# PARTE 7: MONITOREO Y OBSERVABILIDAD

## 7.1 – Ver estado de pods

*# Ver todos los pods*

*kubectl get pods -n load-balancer-demo*

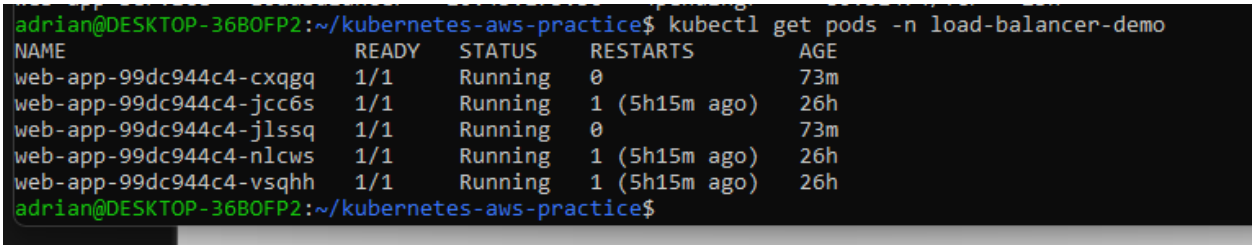
*# Ver logs de un pod específico*

*kubectl logs -n load-balancer-demo web-app-xxxxx-11111*

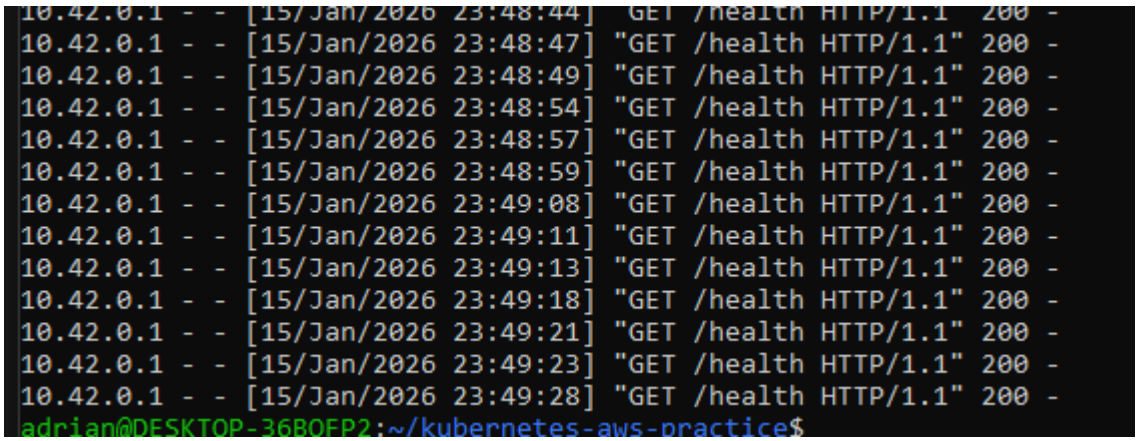
*# Ver logs en tiempo real*

*kubectl logs -n load-balancer-demo -l app=web-app -f*

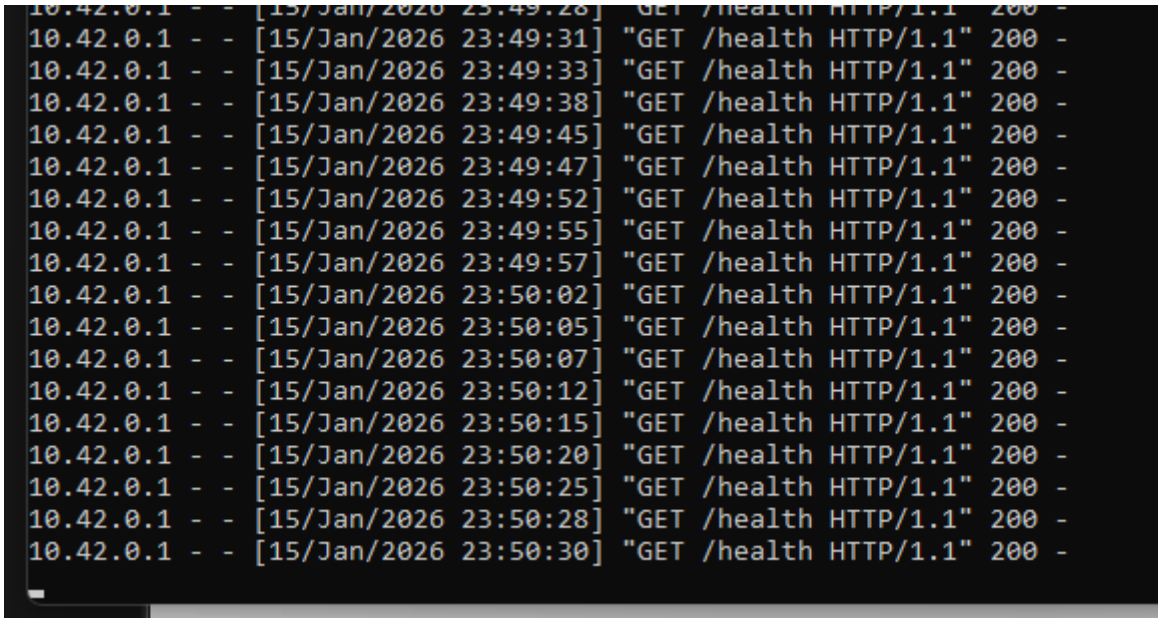
Aquí he listado todos los pods del namespace load-balancer-demo para comprobar que están en estado Running y Ready.




Aquí he consultado los logs del pod web-app-99dc944c4-cxqgq, para ver las peticiones que está recibiendo y comprobar que la app responde bien (códigos 200 en /health)




Aquí he dejado los logs en tiempo real con kubectl logs -f del mismo pod web-app-99dc944c4-cxqgq, y se ve cómo van apareciendo las nuevas peticiones conforme se hacen pruebas, confirmando que el servicio está funcionando.



## 7.2 – Ver estadísticas

*# CPU y memoria de pods*

*kubectl top pods -n load-balancer-*

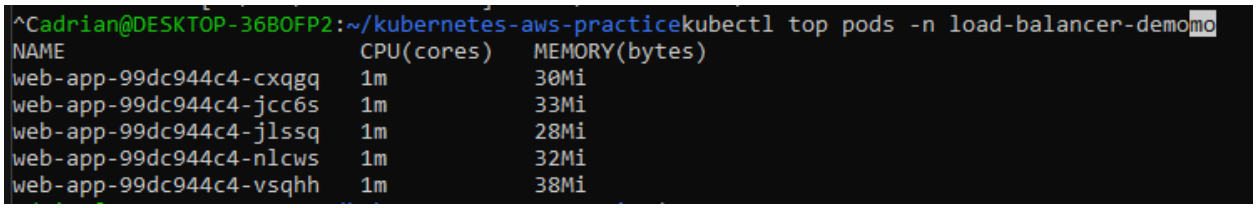
*demo # Nodos del cluster*

*kubectl top nodes*

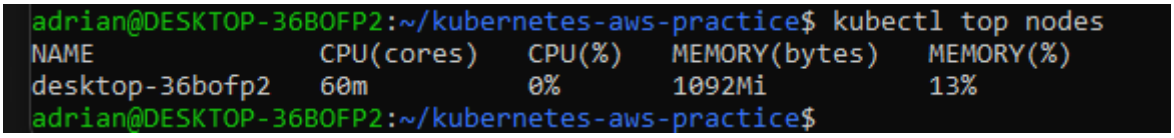
*# Eventos del cluster*

*kubectl get events -n load-balancer-demo*

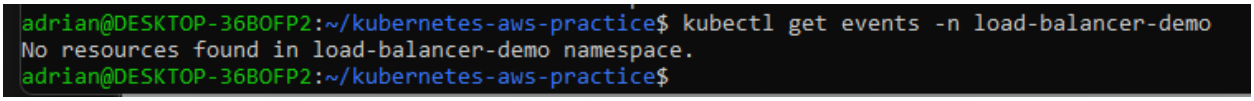
Aquí he sacado el consumo de CPU y memoria de los pods del namespace load-balancer-demo con kubectl top pods para ver que el uso es normal.




Aquí he comprobado el uso de recursos de los nodos del clúster con kubectl top nodes para ver CPU y memoria a nivel de nodo.




Aquí he comprobado los eventos del namespace load-balancer-demo con kubectl get events y no aparece nada, así que no hay avisos ni errores registrados en este momento.




## 7.3 – Ver detalles del deployment

*# Información completa del deployment*

*kubectl describe deployment web-app -n load-balancer-demo*

*# Historial de cambios*

*kubectl rollout history deployment web-app -n load-balancer-demo*

Aquí he usado kubectl describe deployment web-app -n load-balancer-demo para ver toda la información del deployment. Se ve que está en el namespace correcto, que ya tengo 5 réplicas disponibles, la imagen que se está usando y las probes de readiness/liveness apuntando a /health, y con esto he confirmado que el despliegue está bien y estable.

```
adrian@DESKTOP-36BOFP2:~/kubernetes-aws-practice$ kubectl describe deployment web-app -n load-balancer-demo
Name:                   web-app
Namespace:              load-balancer-demo
CreationTimestamp:      Wed, 14 Jan 2026 22:31:42 +0100
Labels:                 app=web-app
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=web-app
Replicas:               5 desired | 5 updated | 5 total | 5 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=web-app
  Containers:
   web-app:
    Image:      python:3.11-slim
    Port:       5000/TCP
    Host Port:  0/TCP
    Command:
      sh
      -c
    Args:

      cd /app

      pip install --no-cache-dir -r requirements.txt >/dev/null 2>&1

      python app.py

    Liveness:   http-get http://:5000/health delay=15s timeout=1s period=10s #success=1 #failure=3
    Readiness:  http-get http://:5000/health delay=5s timeout=1s period=5s #success=1 #failure=3
    Environment:
      POD_NAME:        (v1:metadata.name)
      POD_NAMESPACE:   (v1:metadata.namespace)
    Mounts:
      /app from app-volume (rw)
  Volumes:
   app-volume:
    Type:          ConfigMap (a volume populated by a ConfigMap)
    Name:          app-files
    Optional:      false
  Node-Selectors:  <none>
  Tolerations:     <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Progressing    True    NewReplicaSetAvailable
  Available      True    MinimumReplicasAvailable
OldReplicaSets:  <none>
NewReplicaSet:   web-app-99dc944c4 (5/5 replicas created)
Events:          <none>
```

Aquí he comprobado el historial de cambios del deployment con kubectl rollout history deployment web-app -n load-balancer-demo. Me muestra la revisión 1, y como no aparece “change-cause”, significa que no se han registrado cambios adicionales (solo la versión actual del deployment).

```
adrian@DESKTOP-36BOFP2:~/kubernetes-aws-practice$ kubectl rollout history deployment web-app -n load-balancer-demo
deployment.apps/web-app
REVISION  CHANGE-CAUSE
1         <none>
```

## PARTE 8: ELIMINAR RECURSOS

### 8.1 – Limpiar Kubernetes

*# Eliminar todo en el namespace*

*kubectl delete namespace load-balancer-demo*

*# Verificar*

*kubectl get namespaces*

### 8.2 – Eliminar instancia EC2

En AWS Console:

1. **EC2** → Instances
2. Selecciona kubernetes-test-server
3. **Instance State** → Terminate
4. Confirma

### 8.3 – Detener k3s

*# Si quieres pausar k3s*

*sudo systemctl stop k3s*

*# Para eliminar completamente*

*sudo /usr/local/bin/k3s-uninstall.sh*

*ESTO NO LO HE HECHO POR SI ACASO COMPROBARLO BIEN*

---

## NOTAS IMPORTANTES

- **k3s vs Minikube:** k3s es más ligero y no necesita Docker ni drivers. Perfecto para WSL2.
- **Sin Docker:** Esta práctica NO usa Docker en ningún momento. Solo Python puro.
- **Túnel SSH:** Es la forma más simple conectar local con AWS sin complicaciones de networking.
- **Balanceo real:** Kubernetes distribuye tráfico entre pods. Ver cambios en el navegador.

- **Costos:** t3.micro está en Free Tier. Elimina cuando termines.
- **Datos:** Los datos no persisten entre reinicios de pods (*sin BD*).

---

## TROUBLESHOOTING

**Error: "Unable to pick a default driver"**

- k3s está instalado. Si aparece este error, ignóralo. k3s no necesita driver.

**Error: "Connection refused" desde EC2**

- El túnel SSH no está activo
- **Solución:** Verificar sesión SSH con -R en terminal local

**Pods en "Pending"**

- k3s no tiene suficientes recursos o está

iniciando *kubectl describe pod <pod-name> -n load-*

*balancer-demo* **Port-forward no responde**

*# Matar procesos previos*

*pkill -f "port-forward"*

*# Reiniciar*

*kubectl port-forward -n load-balancer-demo svc/web-app-service 8080:80*

**ConfigMap no se actualiza**

- Los pods en ejecución mantienen versión anterior

*kubectl rollout restart deployment web-app -n load-balancer-demo*

- k3s no inicia después de actualizar

*sudo /usr/local/bin/k3s-uninstall.sh*

*curl -sfL https://get.k3s.io | K3S_KUBECONFIG_MODE="c44" sh -*