# 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

# 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*

```
root@santiago-VirtualBox:/home/santiago# sudo apt update && sudo apt upgrade -y
Des:1 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Obj:2 http://es.archive.ubuntu.com/ubuntu noble InRelease
Des:3 http://es.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Des:4 http://es.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Des:5 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [1.39
9 kB]
```

*# Instalar dependencias mínimas*

*sudo apt install -y curl wget git*

```
root@santiago-VirtualBox:/home/santiago# sudo apt install -y curl wget git
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
curl ya está en su versión más reciente (8.5.0-2ubuntu10.6).
wget ya está en su versión más reciente (1.21.4-1ubuntu4.1).
fijado wget como instalado manualmente.
El paquete indicado a continuación se instaló de forma automática y ya no es nec
esario.
```

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

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

```
root@santiago-VirtualBox:/home/santiago# curl -sfL https://get.k3s.io | K3S_KUBE
CONFIG_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
```

*# Verificar que k3s está instalado*

*sudo k3s --version*

```
root@santiago-VirtualBox:/home/santiago# sudo k3s --version
k3s version v1.34.3+k3s1 (48ffa7b6)
go version go1.24.11
```

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

*sudo k3s server & sleep 10*

```
root@santiago-VirtualBox:/home/santiago# sudo k3s server & sleep 10
[1] 16955
INFO[0000] Starting k3s v1.34.3+k3s1 (48ffa7b6)
INFO[0000] Configuring sqlite3 database connection pooling: maxIdleConns=2, maxO
penConns=0, connMaxLifetime=0s
INFO[0000] Configuring database table schema and indexes, this may take a moment
```

*# Verificar que está corriendo*

*sudo k3s kubectl get nodes*

```
root@santiago-VirtualBox:/home/santiago# sudo k3s kubectl get nodes
NAME                 STATUS   ROLES           AGE   VERSION
santiago-virtualbox   Ready    control-plane   97s   v1.34.3+k3s1
```

## 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/amd64/kubectl"*

*sudo chmod +x kubectl*

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

```
root@santiago-VirtualBox:/home/santiago# curl -LO "https://dl.k8s.io/release/$(c
url -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
sudo chmod +x kubectl
sudo mv kubectl /usr/local/bin/
curl: (2) no URL specified
curl: try 'curl --help' or 'curl --manual' for more information
bash: https://dl.k8s.io/release/stable.txt: No existe el archivo o el directorio
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   138  100   138    0     0    740      0 --:--:-- --:--:-- --:--:--   741
100   238  100   238    0     0    646      0 --:--:-- --:--:-- --:--:--   646
```

```
root@santiago-VirtualBox:/home/santiago# sudo chmod +x kubectl
root@santiago-VirtualBox:/home/santiago# sudo mv kubectl /usr/local/bin/
```

*# Verificar*

*kubectl version --client*

```
root@santiago-VirtualBox:/home/santiago# kubectl version --client
/usr/local/bin/kubectl: línea 1: error sintáctico cerca del elemento inesperado
`<'
/usr/local/bin/kubectl: línea 1: `<?xml version='1.0' encoding='UTF-8'?><Error><
Code>NoSuchKey</Code><Message>The specified key does not exist.</Message><Detail
s>No such object: 767373bbdcb8270361b96548387bf2a9ad0d48758c35/release/bin/linux
/amd64/kubectl</Details></Error>'
```

*# Configurar kubeconfig para acceder sin sudo*

*mkdir -p $HOME/.kube*

```
root@santiago-VirtualBox:/home/santiago# mkdir -p $HOME/.kube
```

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

```
root@santiago-VirtualBox:/home/santiago# sudo cp /etc/rancher/k3s/k3s.yaml $HOME
/.kube/config
```

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

```
root@santiago-VirtualBox:/home/santiago# sudo chown $(id -u):$(id -g) $HOME/.kub
e/config
```

*sudo chmod 600 $HOME/.kube/config*

```
root@santiago-VirtualBox:/home/santiago# sudo chmod 600 $HOME/.kube/config
```

*# Verificar que funciona sin sudo*

*kubectl get nodes*

```
root@santiago-VirtualBox:~/kubernetes-aws-practice/app# kubectl get nodes
NAME                  STATUS   ROLES           AGE   VERSION
santiago-virtualbox   Ready    control-plane   7d    v1.34.3+k3s1
```

## 0.3 – Crear directorio de trabajo

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

```
root@santiago-VirtualBox:/home/santiago# mkdir -p ~/kubernetes-aws-practice
```

*cd ~/kubernetes-aws-practice*

```
root@santiago-VirtualBox:/home/santiago# cd ~/kubernetes-aws-practice
```

## PARTE 1: CREAR APLICACIÓN SIMPLE PARA

## KUBERNETES 1.1 – Crear carpeta para la aplicación

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

```
root@santiago-VirtualBox:~/kubernetes-aws-practice# mkdir -p ~/kubernetes-aws-pr
actice/app
```

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

```
root@santiago-VirtualBox:~/kubernetes-aws-practice# cd ~/kubernetes-aws-practice
/app
```

### 1.2 – Crear aplicación Python con Flask

Crear app.py:

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

#!/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') ssh*

*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*

```
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()
 })
st='0.0.0.0', port=5000, debug=False) EOF sudo chmod +x app.py derr) app.run(ho
```

## 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: 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 style="margin: 0 0 10px 0; color: #666;">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*

```
<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 {
EOFtml> t> etElementById('pod-name').textContent = ' Error';  console.error('Err
```

### 1.4 – Crear requirements.txt

*cat > requirements.txt << 'EOF'*

*Flask==3.0.0*

*Werkzeug==3.0.0*

*EOF*

```
root@santiago-VirtualBox:~/kubernetes-aws-practice/app# cat > requirements.txt <
< 'EOF'
Flask==3.0.0
Werkzeug==3.0.0
EOF
```

### 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*

```
root@santiago-VirtualBox:~/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*

```
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
/usr/local/bin/kubectl: línea 1: error sintáctico cerca del elemento inesperado
`<'
/usr/local/bin/kubectl: línea 1: `<?xml version='1.0' encoding='UTF-8'?><Error><
Code>NoSuchKey</Code><Message>The specified key does not exist.</Message><Detail
s>No such object: 767373bbdcb8270361b96548387bf2a9ad0d48758c35/release/bin/linux
/amd64/kubectl</Details></Error>'
/usr/local/bin/kubectl: línea 1: error sintáctico cerca del elemento inesperado
`<'
/usr/local/bin/kubectl: línea 1: `<?xml version='1.0' encoding='UTF-8'?><Error><
Code>NoSuchKey</Code><Message>The specified key does not exist.</Message><Detail
s>No such object: 767373bbdcb8270361b96548387bf2a9ad0d48758c35/release/bin/linux
/amd64/kubectl</Details></Error>'
```

## 2.2 – Crear ConfigMap con archivos de la app

*cat > configmap.yaml << 'EOF'*
*apiVersion: v1*

*d: CkinonfigMap*

```
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 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)


  index.html: |
    <!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 style="margin: 0 0 10px 0; color: #666;">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*

```
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 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():
kubectl get configmap -n load-balancer-demo tent = ' Error';  console.error('Err
```

## 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 30-60 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*

```
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:
 - |
kubectl get pods -n load-balancer-demo -l app=web-app -n load-balancer-demo --ti
```

### 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
```

```
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
```

## 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*

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

### 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"*

```
> # 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"
```

### 3.3 – Verificar en navegador

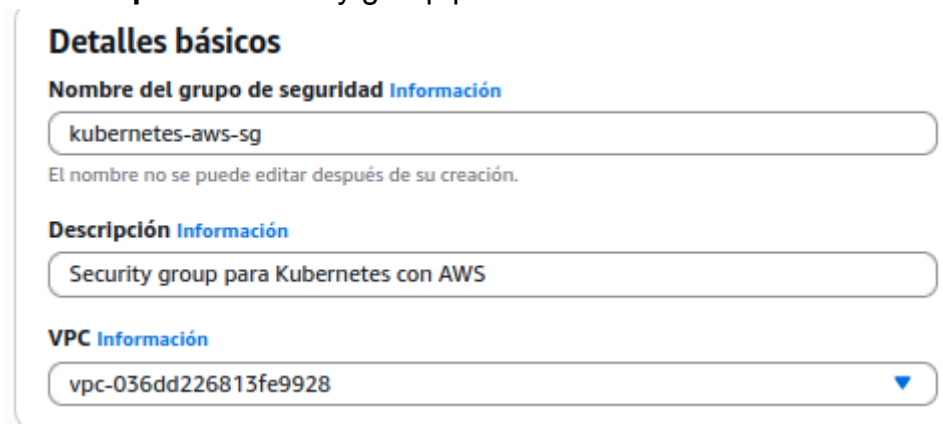
Abre http://localhost:8080 en tu navegador y actualiza varias veces. Verás que  cambia el pod que atiende.

# PARTE 4: CONFIGURACIÓN EN AWS

### 4.1 – Crear Security Group

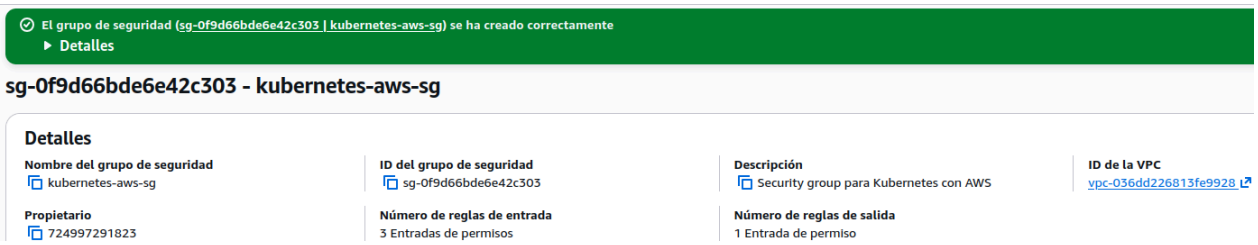
En AWS Console:

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




Agregar reglas de entrada:

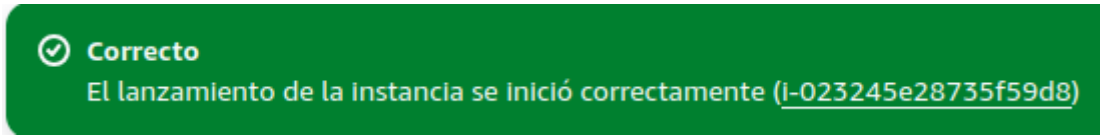
| Tipo | Protocolo | Puerto | Origen |
|---|---|---|---|
| **SSH** | **TCP** | 22 | 0.0.0.0/0 |
| **HTTP** | **TCP** | 80 | 0.0.0.0/0 |
| **HTTPS** | **TCP** | 443 | 0.0.0.0/0 |




### 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: 54.123.45.67)*

*# Conectar via SSH*

*ssh -i ~/.ssh/tu-clave-aws.pem ubuntu@54.123.45.67*

```
root@santiago-VirtualBox:~/kubernetes-aws-practice/app# ssh -i /home/santiago/De
scargas/clave_kubernetes.pem ubuntu@34.236.152.52
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.14.0-1015-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Fri Jan 16 08:27:34 UTC 2026

  System load:  0.0               Temperature:           -273.1 C
  Usage of /:   26.5% of 6.71GB   Processes:             111
  Memory usage: 24%               Users logged in:       0
  Swap usage:   0%                IPv4 address for ens5: 172.31.20.252
```

*# En EC2, instalar herramientas*

*sudo apt update*

```
ubuntu@ip-172-31-20-252:~$ sudo apt update
```

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

```
ubuntu@ip-172-31-20-252:~$ sudo apt install -y curl wget python3 python3-pip git
Reading package lists... Done
```

*# Crear directorio de trabajo*

*mkdir -p ~/kubernetes-test*

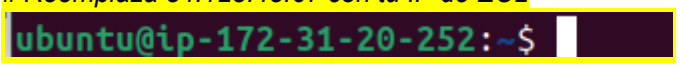
```
ubuntu@ip-172-31-20-252:~$ mkdir -p ~/kubernetes-test
```

# 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 54.123.45.67 con tu IP de EC2*




*ssh -i ~/.ssh/tu-clave-aws.pem *

*-N -R 8888:localhost:8080 *

*ubuntu@54.123.45.67*

*# -N: No ejecutar comandos remotos*

*# -R 8888:localhost:8080: Redirige puerto 8888 en EC2 a puerto 8080 local*

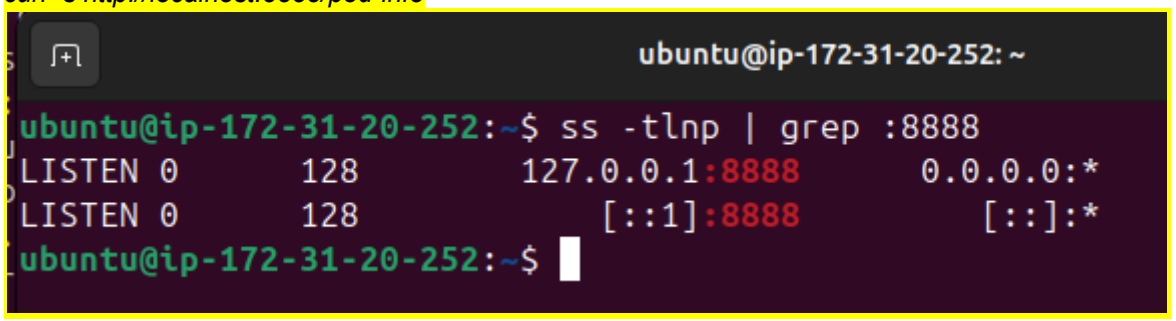
En AWS EC2 (*otra terminal local*):

*# Conecta a la instancia en otra terminal*

*ssh -i ~/.ssh/tu-clave-aws.pem ubuntu@54.123.45.67*

*# 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", ...}*

### 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)

 if [ -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
```

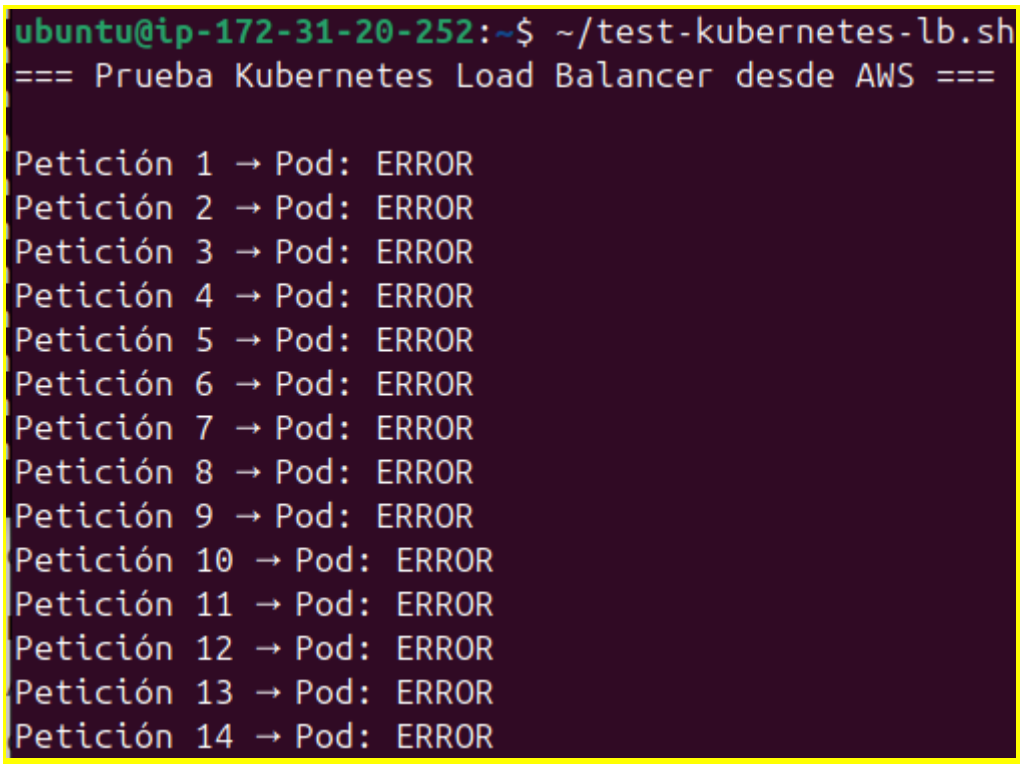
```
ubuntu@ip-172-31-20-252:~$ cat > ~/test-kubernetes-lb.sh << 'EOF'
> #!/bin/bash
echo "=== Prueba Kubernetes Load Balancer desde AWS ==="
echo ""
declare -A pods_count
for i in {1..15}; do
 response=$(curl -s http://localhost:8888/pod-info)
 pod_name=$(echo $response | python3 -c "import sys, json;
print(json.load(sys.stdin)['pod_name'])" 2>/dev/null)
 if [ -z "$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*

```
ubuntu@ip-172-31-20-252:~$ sudo chmod +x ~/test-kubernetes-lb.sh
```

*# Ejecutar prueba*

*~/test-kubernetes-lb.sh*


ubuntu@ip-172-31-20-252:~$ ~/test-kubernetes-lb.sh
=== Prueba Kubernetes Load Balancer desde AWS ===
Petición 1 → Pod: ERROR
Petición 2 → Pod: ERROR
Petición 3 → Pod: ERROR
Petición 4 → Pod: ERROR
Petición 5 → Pod: ERROR
Petición 6 → Pod: ERROR
Petición 7 → Pod: ERROR
Petición 8 → Pod: ERROR
Petición 9 → Pod: ERROR
Petición 10 → Pod: ERROR
Petición 11 → Pod: ERROR
Petición 12 → Pod: ERROR
Petición 13 → Pod: ERROR
Petición 14 → Pod: ERROR

# 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*

```
santiago@santiago-VirtualBox:~$ kubectl scale deployment web-app -n load-balancer-demo
error: required flag(s) "replicas" not set
santiago@santiago-VirtualBox:~$ kubectl get pods -n load-balancer-demo
No resources found in load-balancer-demo namespace.
```

*# Esperar a que estén ready*

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

```
santiago@santiago-VirtualBox:~$ kubectl wait --for=condition=ready pod -l app=web-app -
n load-balancer-demo --timeout=120s
error: no matching resources found
```

## 6.2 – Probar balanceo desde AWS

En EC2:

*~/test-kubernetes-lb.sh*

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

```
santiago@santiago-VirtualBox:~$ ~/test-kubernetes-lb.sh
bash: /home/santiago/test-kubernetes-lb.sh: No existe el archivo o el directorio
```

## PARTE 7: MONITOREO Y OBSERVABILIDAD 7.1 – Ver estado de pods

*# Ver todos los pods*

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

```
santiago@santiago-VirtualBox:~$ kubectl get pods -n load-balancer-demo
No resources found in load-balancer-demo namespace.
```

*# 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*

### 7.2 – Ver estadísticas

*# CPU y memoria de pods*

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

*# Nodos del cluster*

*kubectl top nodes*

```
santiago@santiago-VirtualBox:~$ kubectl logs -n load-balancer-demo -l app=web-app -f
No resources found in load-balancer-demo namespace.
santiago@santiago-VirtualBox:~$ ~kubectl top pods -n load-balancer-demo
Orden «~kubectl» no encontrada. Quizá quiso decir:
  la orden «kubectl» del paquete snap «kubectl (1.34.3)»
Consulte «snap info <nombre del snap>» para ver más versiones.
santiago@santiago-VirtualBox:~$ kubectl top nodes
NAME                 CPU(cores)   CPU(%)   MEMORY(bytes)   MEMORY(%)
santiago-virtualbox   160m         4%       1676Mi          42%
```

*# Eventos del cluster*

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

```
santiago@santiago-VirtualBox:~$ kubectl get events -n load-balancer-demo
No resources found in load-balancer-demo namespace.
```

### 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*

```
santiago@santiago-VirtualBox:~$ kubectl describe deployment web-app -n load-balancer-de
mo # Historial de cambios
kubectl rollout history deployment web-app -n load-balancer-demo
Error from server (NotFound): namespaces "load-balancer-demo" not found
Error from server (NotFound): namespaces "load-balancer-demo" not found
```

## PARTE 8: ELIMINAR RECURSOS

### 8.1 – Limpiar Kubernetes

*# Eliminar todo en el namespace*

*kubectl delete namespace load-balancer-demo*

```
santiago@santiago-VirtualBox:~$ kubectl delete namespace load-balancer-demo
Error from server (NotFound): namespaces "load-balancer-demo" not found
```

*# Verificar*

*kubectl get namespaces*

```
santiago@santiago-VirtualBox:~$ kubectl get namespaces
NAME              STATUS   AGE
default           Active   14d
kube-node-lease   Active   14d
kube-public       Active   14d
kube-system       Active   14d
practica          Active   6d23h
```

### 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*

```
santiago@santiago-VirtualBox:~$ sudo systemctl stop k3s
```

*# Para eliminar completamente*

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

```
santiago@santiago-VirtualBox:~$ sudo /usr/local/bin/k3s-uninstall.sh
+ id -u
+ [ 0 -eq 0 ]
+ K3S_DATA_DIR=/var/lib/rancher/k3s
+ /usr/local/bin/k3s-killall.sh
+ [ -s /etc/systemd/system/k3s.service ]
+ basename /etc/systemd/system/k3s.service
+ systemctl stop k3s.service
+ [ -x /etc/init.d/k3s* ]
+ killtree 15325 15380 15420 16576 16665 35255
+ kill -9 15325 15386 15725 15380 15467 15717 15420 15492 15967 16576 16607 16846 16891
 16665 16691 16994 35255 35278 35342 35377 35378 35379 35380
+ do_unmount_and_remove /run/k3s
+ set +x
sh -c 'umount -f "$0" && rm -rf "$0"' /run/k3s/containerd/io.containerd.runtime.v2.task
/k8s.io/bbd5642ce5b53d8cfec88a2b42db86766ea23b0f2b16ff6af4c561574b735eb7/rootfs
```

## 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="644" sh -*