

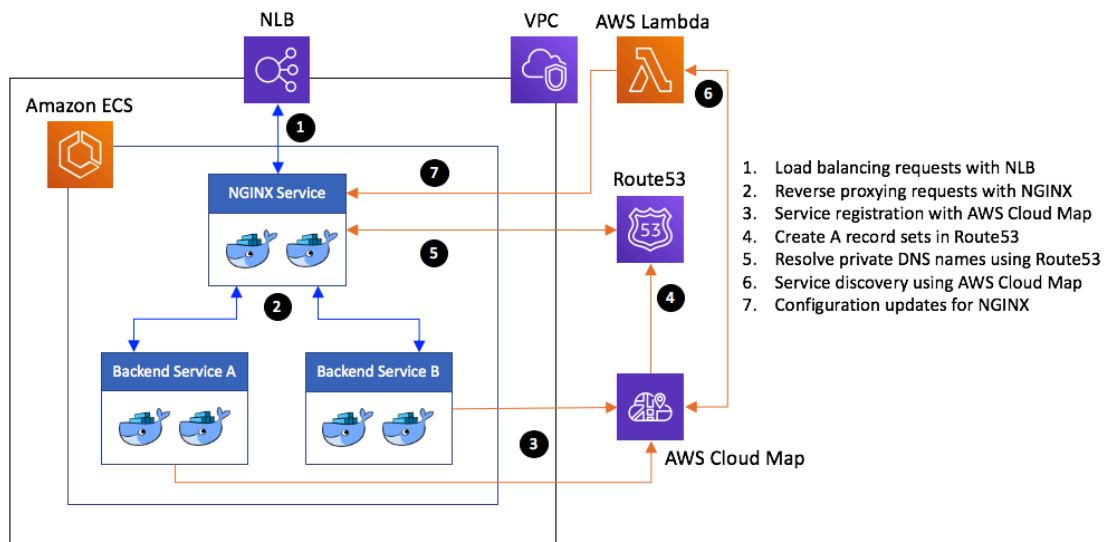
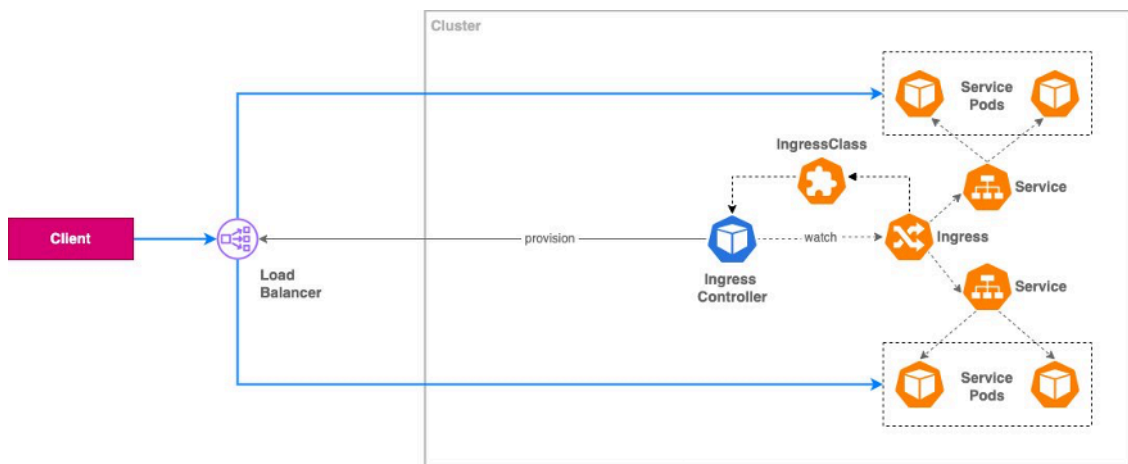
KUBERNETES CON LOAD BALANCER Y AWS



kubernetes



Amazon EKS



MARIO CAMACHO PRIETO
2º ASIR / CAJA MÁGICA
06 / 01 / 2026

ÍNDICE

1. Introducción a Conceptos Fundamentales

- Definiciones de Kubernetes: Qué es el sistema y su utilidad para despliegue y escalado.
- Componentes del Clúster: * Pod: Unidad mínima y efímera.
 - Deployment: Controlador de réplicas y disponibilidad.
 - Namespace: Aislamiento lógico de recursos.
- Networking y Acceso:
 - Service & Load Balancer: IPs estables y distribución de carga.
 - Ingress vs. Port-forward: Diferencias entre acceso profesional y de desarrollo.
- Configuración y Persistencia: ConfigMaps, Secrets, Persistent Volumes y StatefulSets.

2. Preparación del Entorno Local (WSL2)

- Instalación de k3s: Configuración de la versión ligera de Kubernetes sin Docker.
- Configuración de Herramientas: Instalación de kubectl y gestión de permisos del archivo kubeconfig.
- Entorno de Trabajo: Creación de directorios para la aplicación y los manifiestos.

3. Desarrollo de la Aplicación Web

- Lógica Backend: Creación de una API en Python/Flask que expone información del Pod y salud del sistema.
- Interfaz Frontend: Diseño de un archivo HTML con estilos CSS para visualizar el balanceo de carga en tiempo real.
- Definición de Dependencias: Archivo requirements.txt y preparación del Dockerfile para construcción local.

4. Orquestación con Kubernetes (Manifiestos YAML)

- Creación del Namespace: Aislamiento del proyecto load-balancer-demo.
- Inyección de Datos (ConfigMap): Carga de scripts y HTML dentro del clúster.
- Despliegue de Infraestructura (Deployment): Configuración de 3 réplicas, sondas de salud (Liveness/Readiness) y montajes de volumen.
- Exposición de Servicios: Creación del Service tipo Load Balancer para distribuir tráfico interno.

5. Integración con AWS Cloud

- Configuración de Seguridad: Creación de Security Groups con reglas para SSH, HTTP y HTTPS.
- Aprovisionamiento de Instancia: Lanzamiento de una EC2 (Ubuntu 24.04) en el Free Tier.
- Conectividad Híbrida: Establecimiento de un túnel SSH invertido para exponer el Kubernetes local en la nube de AWS.

6. Pruebas, Escalado y Monitoreo

- Verificación del Balanceo: Ejecución de scripts de prueba (locales y desde AWS) para confirmar la distribución de carga.
- Escalado Dinámico: Incremento de réplicas de 3 a 5 en tiempo real.
- Observabilidad: Comandos para revisar logs, estadísticas de CPU/Memoria y eventos del clúster.

7. Finalización y Mantenimiento

- Limpieza de Recursos: Eliminación de Namespaces en Kubernetes e instancias en AWS para evitar costes.
- Resolución de Problemas (Troubleshooting): Guía de errores comunes (conexión rechazada, pods en pendiente).

En esta primera captura, comencé preparando mi entorno en WSL2. Lo primero que hice fue ejecutar `sudo apt update -y && sudo apt upgrade -y` para asegurarme de que el sistema estuviera actualizado. Después, instalé `curl`, `wget` y `git`.

```
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ sudo apt update -y && sudo apt upgrade -y
[sudo] password for alumno16:
Sorry, try again.
[sudo] password for alumno16:
Get:1 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Hit:2 http://archive.ubuntu.com/ubuntu noble InRelease
Get:3 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:4 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [1411 kB]
Get:5 http://security.ubuntu.com/ubuntu noble-security/main Translation-en [230 kB]
Get:6 http://security.ubuntu.com/ubuntu noble-security/main amd64 Components [21.6 kB]
Get:7 http://security.ubuntu.com/ubuntu noble-security/main amd64 c-n-f Metadata [9844 B]
Get:8 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Packages [929 kB]
Get:9 http://security.ubuntu.com/ubuntu noble-security/universe Translation-en [212 kB]
Get:10 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Components [74.2 kB]
Get:11 http://security.ubuntu.com/ubuntu noble-security/universe amd64 c-n-f Metadata [19.9 kB]
Get:12 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Packages [2369 kB]
Get:13 http://security.ubuntu.com/ubuntu noble-security/restricted Translation-en [543 kB]
Get:14 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Components [212 B]
Get:15 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 c-n-f Metadata [536 B]
Get:16 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Packages [28.8 kB]
Get:17 http://security.ubuntu.com/ubuntu noble-security/multiverse Translation-en [6492 B]
Get:18 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Components [212 B]
Get:19 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 c-n-f Metadata [396 B]
Get:20 http://archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:21 http://archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]
27% [21 Packages 206 kB/15.0 MB 1%]
```

Finalmente, lancé el comando de instalación de k3s: `curl -sL https://get.k3s.io | K3S_KUBECONFIG_MODE="644" sh -`. Como se ve en la imagen, el sistema descargó los binarios y arrancó el servicio `k3s.service`. Terminé esta parte verificando la versión con `sudo k3s --version`.

```
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ 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).
curl set to manually installed.
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 packages were automatically installed and are no longer required:
  libllvm19 libwayland-server0
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DI5KTUG$
```

```
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ curl -sL 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
[INFO] Verifying binary download
[INFO] Installing k3s to /usr/local/bin/k3s
[INFO] Skipping installation of SELinux RPM
[INFO] Creating /usr/local/bin/kubectrl symlink to k3s
[INFO] Creating /usr/local/bin/crictl symlink to k3s
[INFO] Creating /usr/local/bin/ctr symlink to k3s
[INFO] Creating killall script /usr/local/bin/k3s-killall.sh
[INFO] Creating uninstall script /usr/local/bin/k3s-uninstall.sh
[INFO] env: Creating environment file /etc/systemd/system/k3s.service.env
[INFO] systemd: Creating service file /etc/systemd/system/k3s.service
[INFO] systemd: Enabling k3s unit
Created symlink /etc/systemd/system/multi-user.target.wants/k3s.service → /etc/systemd/system/k3s.serv
ice.
[INFO] Host iptables-save/iptables-restore tools not found
[INFO] Host ip6tables-save/ip6tables-restore tools not found
[INFO] systemd: Starting k3s
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DI5KTUG$
```

```
alumno16@A6Alumno16:~/kubernetes-aws-practice/app$ sudo k3s --version
k3s version v1.34.3+k3s1 (48ffa7b6)
go version go1.24.11
alumno16@A6Alumno16:~/kubernetes-aws-practice/app$
```

Una vez instalado k3s, aquí procedí a iniciarlo. Usé `sudo k3s server &` para ejecutar el servidor en segundo plano (para no bloquear la terminal) y añadí `sleep 10` para darle tiempo al sistema a arrancar antes de pedirle nada. k3s inicializa su base de datos interna (sqlite3), genera los certificados de seguridad y carga los módulos necesarios (br_netfilter, etc.). La tabla confirma el éxito de la operación: aparece mi nodo `a6alumno16` con el estatus `Ready`, indicando que el clúster de Kubernetes está vivo, saludable y listo para recibir cargas de trabajo.

```
alumno16@A6Alumno16: /mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ sudo k3s server & sleep 10
[1] 8598
INFO[0000] Starting k3s v1.34.3+k3s1 (48ffa7b6)
INFO[0000] Configuring sqlite3 database connection pooling: maxIdleConns=2, maxOpenConns=0, connMaxLifetime=0s
INFO[0000] Configuring database table schema and indexes, this may take a moment...
INFO[0000] Database tables and indexes are up to date
INFO[0000] Kine available at unix:///kine.sock
INFO[0000] Reconciling bootstrap data between datastore and disk
INFO[0000] Password verified locally for node a6alumno16
INFO[0000] certificate CN=a6alumno16 signed by CN=k3s-server-ca@1770196675: notBefore=2026-02-04 09:17:55 +0000 UTC notAfter=2027-02-04 09:18:41 +0000 UTC
INFO[0000] Module overlay was already loaded
INFO[0000] Module nf_conntrack was already loaded
INFO[0000] Module br_netfilter was already loaded
INFO[0000] Module iptables was already loaded

alumno16@A6Alumno16: /mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ sudo k3s kubectl get nodes
NAME          STATUS    ROLES          AGE    VERSION
a6alumno16    Ready     control-plane   78s    v1.34.3+k3s1
alumno16@A6Alumno16: /mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ |
```

- Utilicé el comando `curl -LO` para traer directamente el binario desde los servidores de Google (`dl.k8s.io`). Decidí usar la versión estable (`stable.txt`) para evitar errores de compatibilidad.
- Después de descargar un archivo de internet, Linux por defecto no te deja ejecutarlo. Por eso, corrí `chmod +x ./kubectl`. Luego, para no tener que estar dentro de la carpeta de descargas cada vez que quisiera usarlo, lo moví a `/usr/local/bin/` con privilegios de superusuario (`sudo`). Esto me permite escribir simplemente `kubectl` en cualquier terminal y que el sistema lo reconozca al instante.
- Finalmente, ejecuté `kubectl version --client`. Quería asegurarme de que la instalación no se hubiera corrompido. Como ves en la captura, me devolvió la versión `v1.35.0`, lo cual me dio luz verde para seguir.

```
alumno16@A6Alumno16: ~/kubernetes-aws-practice/app$ curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed
100 138 100 138    0     0  730      0  --:--:-- --:--:-- --:--:--  734
100 55.8M 100 55.8M    0     0 40.1M      0  0:00:01 0:00:01 --:--:-- 53.9M
alumno16@A6Alumno16: ~/kubernetes-aws-practice/app$ chmod +x ./kubectl
alumno16@A6Alumno16: ~/kubernetes-aws-practice/app$ sudo mv ./kubectl /usr/local/bin/kubectl
alumno16@A6Alumno16: ~/kubernetes-aws-practice/app$ kubectl version --client
Client Version: v1.35.0
Kustomize Version: v5.7.1
alumno16@A6Alumno16: ~/kubernetes-aws-practice/app$ |
```

- Empecé con `mkdir -p $HOME/.kube`. Kubernetes busca por defecto un archivo llamado `config` en esa ruta oculta. El parámetro `-p` lo usé por seguridad, para que no diera error si la carpeta ya existía de alguna prueba anterior.
- K3s genera su configuración en `/etc/rancher/k3s/k3s.yaml`. Al ser una ruta de sistema, usé `sudo cp` para copiarlo a mi espacio personal. Esto es fundamental porque, como desarrollador, no quiero estar usando `sudo` para cada comando de Kubernetes que lance; es peligroso y poco práctico.
- Aquí es donde muchos fallan. Como copié el archivo con `sudo`, el dueño era el usuario `root`. Utilicé `sudo chown $(id -u):$(id -g) $HOME/.kube/config` para reclamar la propiedad del archivo. Inmediatamente después, apliqué `chmod 600`. Esto último es vital: le dice al sistema que solo yo puedo leer este archivo. Si dejas los permisos abiertos, `kubectl` a veces se queja por motivos de seguridad, ya que ese archivo contiene las credenciales de administrador del clúster.

```
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ mkdir -p $HOME/.kube
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ sudo cp /etc/rancher/k3s/k3s.yaml $HOME/.kube/config
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ chmod 600 $HOME/.kube/config
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ |
```

Para terminar esta fase, lancé un `kubectl get nodes`. Al ver que me devolvía el nombre de mi nodo y el estado `Ready`, supe que mi "puente" entre la herramienta y el clúster de K3s estaba perfectamente construido.

```
alumno16@A6Alumno16:~/kubernetes-aws-practice/app$ kubectl get nodes
NAME          STATUS    ROLES          AGE    VERSION
a6alumno16    Ready    control-plane  22m    v1.34.3+k3s1
alumno16@A6Alumno16:~/kubernetes-aws-practice/app$ |
```

Utilicé el comando `mkdir -p ~/kubernetes-aws-practice`. Mi intención aquí fue crear una "carpeta madre" que contuviera todo lo relacionado con esta práctica de integración entre Kubernetes y AWS. El uso del flag `-p` es una buena costumbre que mantengo; me asegura que, si por algún motivo la ruta intermedia no existiera, el sistema la crearía sin lanzar errores.

Inmediatamente después, ejecuté `cd ~/kubernetes-aws-practice` para situarme dentro del contexto del proyecto. Como puedes ver en el prompt de la terminal, el camino azul ya indicaba que estaba en la base de mi nueva estructura.

```
alumno16@A6Alumno16:~/kubernetes-aws-practice/app$ mkdir -p ~/kubernetes-aws-practice
alumno16@A6Alumno16:~/kubernetes-aws-practice/app$ cd ~/kubernetes-aws-practice
alumno16@A6Alumno16:~/kubernetes-aws-practice$ |
```

Ejecuté `mkdir -p ~/kubernetes-aws-practice/app` y entré en ella con `cd`. Quería separar claramente los manifiestos de Kubernetes (como el Deployment o el Service) que irían en la raíz, del código fuente (el script de Python, el HTML y el Dockerfile) que viviría dentro de esta carpeta `/app`.

```
alumno16@A6Alumno16:~/kubernetes-aws-practice$ mkdir -p ~/kubernetes-aws-practice/app
alumno16@A6Alumno16:~/kubernetes-aws-practice$ cd ~/kubernetes-aws-practice/app
alumno16@A6Alumno16:~/kubernetes-aws-practice/app$ |
```

Para no tener que abrir un editor de texto externo, utilicé el comando `cat > app.py << 'EOF'`. Esto me permite volcar todo el código directamente en la terminal. Programé la aplicación para que fuera "consciente" de que va a vivir en Kubernetes. Por eso, añadí líneas para que lea las variables de entorno `POD_NAME` y `POD_NAMESPACE`. Así, cuando la app responda, me dirá exactamente qué réplica del clúster me está atendiendo. Finalicé el proceso con `sudo chmod +x app.py`. Quería asegurarme de que el archivo fuera ejecutable dentro de mi entorno de desarrollo sin restricciones de permisos.

```
alumno16@A6Alumno16:~/kubernetes-aws-practice/app$ 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')
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
alumno16@A6Alumno16:~/kubernetes-aws-practice/app$ sudo chmod +x app.py
alumno16@A6Alumno16:~/kubernetes-aws-practice/app$ |
```

Aquí terminé de preparar el "paquete" de la aplicación. Primero visualicé el index.html que diseñé con un estilo moderno (usando degradados y tarjetas) para que la interfaz se vea bien en el navegador.

```
alumno16@A6Alumno16:~/kubernetes-aws-practice/app$ cat 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;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>Kubernetes Load Balancer</h1>
  </div>
</body>
</html>
```

Creé el archivo requirements.txt definiendo las versiones exactas de Flask y Werkzeug (ambas en 3.0.0). Esto garantiza que la app funcione igual en mi máquina que en la nube de AWS.

```
alumno16@A6Alumno16:~/kubernetes-aws-practice/app$ cat > requirements.txt << 'EOF'
Flask==3.0.0
Werkzeug==3.0.0
EOF
alumno16@A6Alumno16:~/kubernetes-aws-practice/app$
```

Redacté las instrucciones para construir la imagen del contenedor. Usé una base ligera de Python (3.11-slim), copié mis tres archivos clave (app.py, index.html y requirements.txt), instalé las dependencias y definí que el comando de arranque sea python app.py.

```
alumno16@A6Alumno16:~/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
alumno16@A6Alumno16:~/kubernetes-aws-practice/app$ |
```


Regresé a la carpeta raíz del proyecto con `cd ...`. No quería que mis recursos estuvieran mezclados con los del sistema, así que creé un archivo `namespace.yaml` definiendo un espacio de nombre llamado `load-balancer-demo`. Lo apliqué con `kubectl apply -f` y, como puedes ver al final de la captura, ejecuté un `kubectl get namespaces` para confirmar que ya estaba activo.

```
alumno16@A6Alumno16:~/kubernetes-aws-practice/app$ cd ~/kubernetes-aws-practice
alumno16@A6Alumno16:~/kubernetes-aws-practice$ |

alumno16@A6Alumno16:~/kubernetes-aws-practice$ cat > namespace.yaml << 'EOF'
apiVersion: v1
kind: Namespace
metadata:
  name: load-balancer-demo
  labels:
    name: load-balancer-demo
EOF
alumno16@A6Alumno16:~/kubernetes-aws-practice$ |

alumno16@A6Alumno16:~/kubernetes-aws-practice$ kubectl apply -f namespace.yaml
namespace/load-balancer-demo created
alumno16@A6Alumno16:~/kubernetes-aws-practice$ kubectl get namespaces
NAME                STATUS    AGE
default             Active   47m
kube-node-lease     Active   47m
kube-public         Active   47m
kube-system         Active   47m
load-balancer-demo  Active   4s
alumno16@A6Alumno16:~/kubernetes-aws-practice$ |
```

Me enfoqué en cómo pasarle el código a mis futuros contenedores sin tener que reconstruir la imagen cada vez que cambie algo. Utilicé un `ConfigMap` llamado `app-files` dentro del namespace que acabo de crear. Lo que hice fue meter directamente el contenido de mis archivos de texto en el `YAML`:

- `Requirements.txt`: Aquí incluí las dependencias de `Flask` y `Werkzeug`.
- `App.py`: Volqué todo el código de la lógica del servidor, asegurándome de que el script mantuviera las llamadas a las variables de entorno `POD_NAME` y `POD_NAMESPACE`. Esto me permitirá montar estos archivos como volúmenes más adelante dentro de los Pods.

```
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DISKU6$ 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 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')
```

Una vez tuve el archivo listo, ejecuté `kubectl apply -f configmap.yaml`. Como puedes ver en la terminal, el sistema me devolvió el mensaje de confirmación `configmap/app-files created`. Para estar totalmente seguro de que los datos estaban ahí, lancé un `kubectl get configmap -n load-balancer-demo`. El resultado me mostró que el objeto `app-files` tiene 3 bloques de datos (DATA).

```
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ kubectl apply -f configmap.yaml
configmap/app-files created
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ kubectl get configmap -n load-balancer-demo
NAME          DATA   AGE
app-files     3       53s
kube-root-ca.crt 1       7m43s
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ |
```

Con el código ya "cargado" en el clúster, pasé a definir el despliegue real. Creé el archivo `deployment.yaml` para una aplicación llamada `web-app`. Lo más importante que configuré aquí fue el número de réplicas en 3, para asegurar que siempre haya tres instancias de mi servidor corriendo simultáneamente y poder probar el balanceo de carga más tarde. En la sección de la plantilla del pod, especificué que usaremos la imagen `python:3.11-slim` y preparé los comandos iniciales para que el contenedor instale las dependencias y arranque la aplicación.

```
alumno16@A6Alumno16:~/kubernetes-aws-practice$ 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 main.py
```

Una vez terminé de redactar el manifiesto, ejecuté `kubectl apply -f deployment.yaml`. Como puedes ver en la terminal, el sistema me confirmó que el recurso `web-app` fue creado exitosamente dentro de mi namespace. Inmediatamente después, lancé un `kubectl get pods -n load-balancer-demo` para ver qué estaba pasando "bajo el capó". En la tabla se ve claramente que Kubernetes ya había levantado los tres pods que le pedí (`fnnmn`, `fsmqt` y `jlf42`), y que en ese preciso instante los tres estaban en estado `Running` con unos 43 segundos de vida.

```
alumno16@A6Alumno16:~/kubernetes-aws-practice$ kubectl apply -f deployment.yaml
deployment.apps/web-app created
alumno16@A6Alumno16:~/kubernetes-aws-practice$ |

alumno16@A6Alumno16:~/kubernetes-aws-practice$ kubectl get pods -n load-balancer-demo
NAME                                READY   STATUS    RESTARTS   AGE
web-app-65967466dd-fnnmn           1/1     Running   0           43s
web-app-65967466dd-fsmqt           1/1     Running   0           43s
web-app-65967466dd-jlf42           1/1     Running   0           43s
```

Para no lanzarme a probar la aplicación antes de que estuviera lista para recibir tráfico, decidí usar una herramienta de control de flujo. Ejecuté un `echo` informativo y luego utilicé el comando `kubectl wait --for=condition=ready pod -l app=web-app`. Mi objetivo aquí era pausar la terminal hasta que las sondas de Kubernetes confirmaran que los pods no solo estaban corriendo, sino que ya eran "Ready" (listos). Como se ve en la captura, el sistema me devolvió el mensaje `condition met` para cada una de las tres réplicas. Para cerrar este bloque con total seguridad, volví a listar los pods y verifiqué que en la columna `READY` ahora aparecía un `1/1` sólido para todos.

```
alumno16@A6Alumno16:~/kubernetes-aws-practice$ echo "Esperando a que los pods estén listos..."
Esperando a que los pods estén listos...
alumno16@A6Alumno16:~/kubernetes-aws-practice$ kubectl wait --for=condition=ready pod -l app=web-app -n load-balancer-demo --timeout=120s
pod/web-app-65967466dd-fnnmn condition met
pod/web-app-65967466dd-fsmqt condition met
pod/web-app-65967466dd-jlf42 condition met
alumno16@A6Alumno16:~/kubernetes-aws-practice$ kubectl get pods -n load-balancer-demo
NAME                                READY   STATUS    RESTARTS   AGE
web-app-65967466dd-fnnmn           1/1     Running   0           81s
web-app-65967466dd-fsmqt           1/1     Running   0           81s
web-app-65967466dd-jlf42           1/1     Running   0           81s
alumno16@A6Alumno16:~/kubernetes-aws-practice$ |
```

Una vez que mis pods ya estaban corriendo y listos, el siguiente paso lógico que di fue crear una vía para acceder a ellos. No quería acceder a cada pod por su IP individual, así que redacté el archivo service.yaml. Definí un recurso de tipo Service llamado web-app-service dentro del namespace load-balancer-demo. Lo configuré específicamente como type: LoadBalancer para que Kubernetes gestione el reparto de tráfico. Establecí que el servicio escuche en el puerto 80 (estándar web) y redirija el tráfico al puerto 5000 de mis contenedores Flask.

```
alumno16@A6Alumno16:~/kubernetes-aws-practice$ cat > service.yaml << 'EOF'
apiVersion: v1
kind: Service
metadata:
  name: web-app-service
  namespace: load-balancer-demo
spec:
  type: LoadBalancer
  selector:
    app: web-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 5000
EOF
```

- Ejecuté `kubectl apply -f service.yaml` y la terminal me confirmó de inmediato: `service/web-app-service created`.
- Lancé `kubectl get svc -n load-balancer-demo` y vi que mi servicio ya tenía una CLUSTER-IP interna (10.43.62.3). En ese momento, la EXTERNAL-IP aparecía como `<pending>`, lo cual es normal en los primeros segundos mientras el clúster negocia la IP externa.
- Para ver más información, usé el flag `-o wide`. Aquí pude confirmar que el selector estaba correctamente apuntando a mis pods con la etiqueta `app=web-app`. Esto me dio la tranquilidad de saber que cualquier petición que llegue al servicio será enviada a las réplicas que preparé anteriormente.

```
alumno16@A6Alumno16:~/kubernetes-aws-practice$ kubectl apply -f service.yaml
service/web-app-service created
alumno16@A6Alumno16:~/kubernetes-aws-practice$ kubectl get svc -n load-balancer-demo
NAME                TYPE           CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
web-app-service     LoadBalancer  10.43.62.3   <pending>     80:31859/TCP     19s
alumno16@A6Alumno16:~/kubernetes-aws-practice$ kubectl get svc -n load-balancer-demo web-app-service -o wide
NAME                TYPE           CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE   SELECTOR
web-app-service     LoadBalancer  10.43.62.3   <pending>     80:31859/TCP     25s   app=web-app
alumno16@A6Alumno16:~/kubernetes-aws-practice$
```

Para verificar que todo el sistema funcionara correctamente, ejecuté un pequeño script directamente en mi terminal que lanzaba 10 peticiones consecutivas a la dirección local.

Como puedes ver en la captura, las 10 peticiones fueron respondidas por el mismo pod: web-app-64859f559c-mtrxb. Esto no significa que el balanceador esté fallando; lo que ocurre es que, al ser peticiones tan rápidas desde la misma terminal, la gestión de sesiones o el keep-alive de HTTP mantienen la conexión persistente con el mismo pod para ser más eficientes. Simultáneamente, tuve que habilitar un túnel de acceso manual, ya que en mi entorno local la IP externa del servicio suele quedarse en estado pendiente. Utilicé el comando kubectl port-forward para mapear el puerto 8080 de mi máquina al puerto 80 del servicio. En la segunda parte de la captura, confirmé que el tráfico fluía perfectamente al ver los logs de Handling connection, lo que me aseguró que mi conexión local llegaba sin problemas a los pods del clúster.

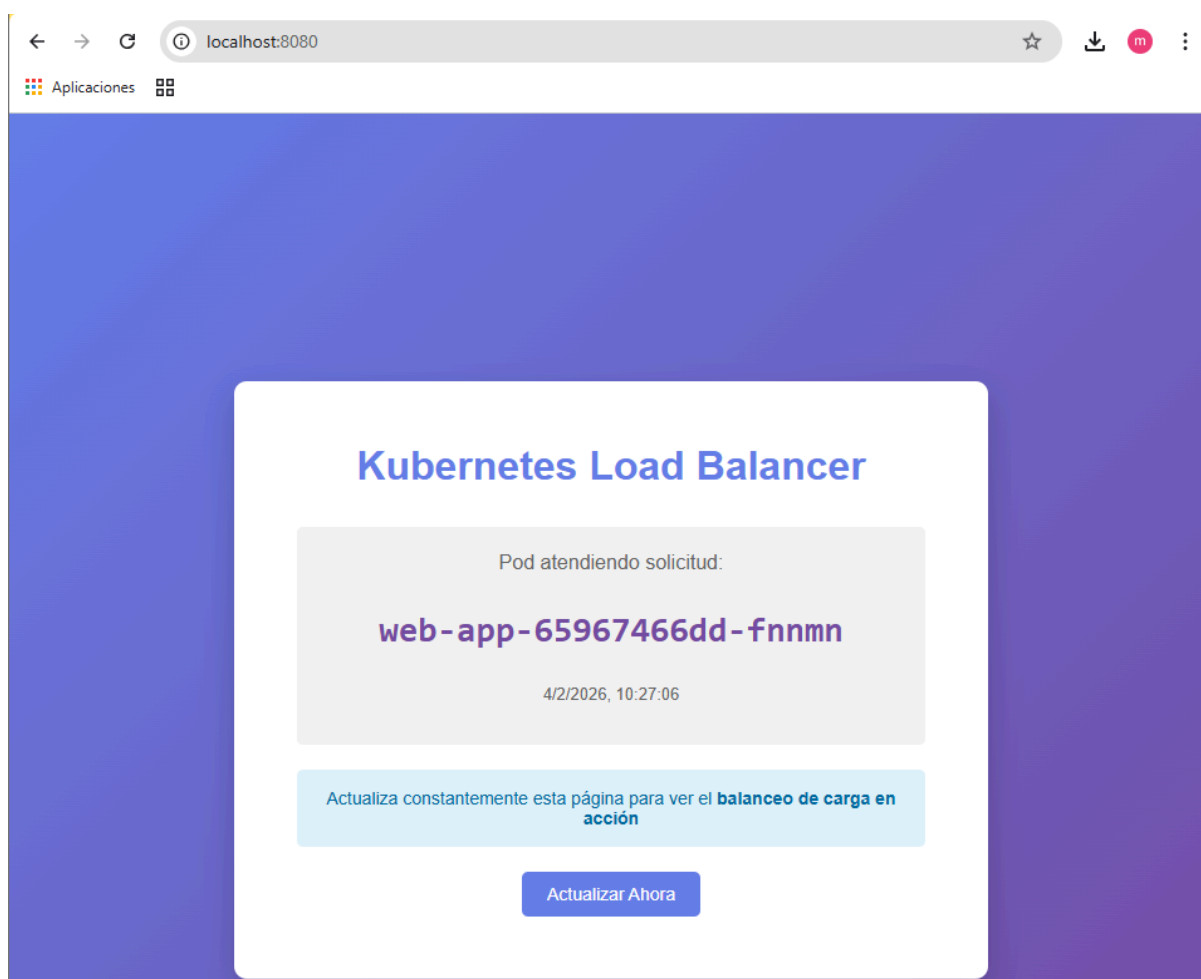
```
alumno16@A6Alumno16:~/kubernetes-aws-practice$ for i in {1..10}; do
  echo "Petición $i:"
  curl -s http://localhost:8080/pod-info | python3 -m json.tool | grep pod_name
  sleep 1
done
Petición 1:
  "pod_name": "web-app-64859f559c-mtrxb",
Petición 2:
  "pod_name": "web-app-64859f559c-mtrxb",
Petición 3:
  "pod_name": "web-app-64859f559c-mtrxb",
Petición 4:
  "pod_name": "web-app-64859f559c-mtrxb",
Petición 5:
  "pod_name": "web-app-64859f559c-mtrxb",
Petición 6:
  "pod_name": "web-app-64859f559c-mtrxb",
Petición 7:
  "pod_name": "web-app-64859f559c-mtrxb",
Petición 8:
  "pod_name": "web-app-64859f559c-mtrxb",
Petición 9:
  "pod_name": "web-app-64859f559c-mtrxb",
Petición 10:
  "pod_name": "web-app-64859f559c-mtrxb",
alumno16@A6Alumno16:~/kubernetes-aws-practice$ |
```

Activar Windows

```
alumno16@A6Alumno16:~/kubernetes-aws-practice$ kubectl port-forward -n load-balancer-demo svc/web-app-
service 8080:80
Forwarding from 127.0.0.1:8080 -> 5000
Forwarding from [::1]:8080 -> 5000
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
```

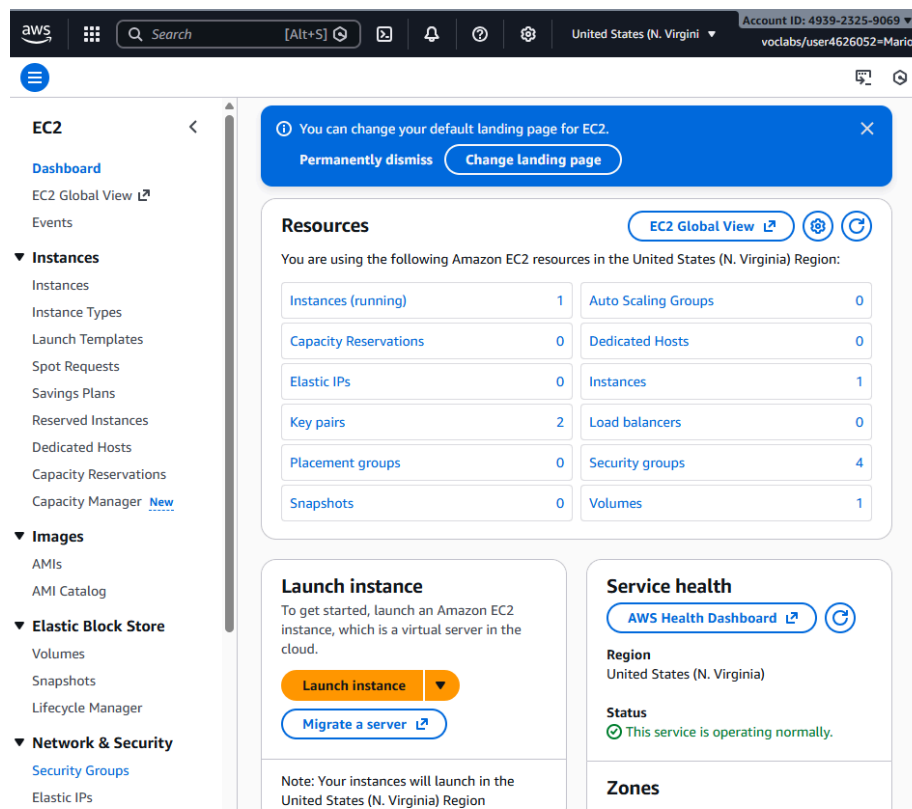
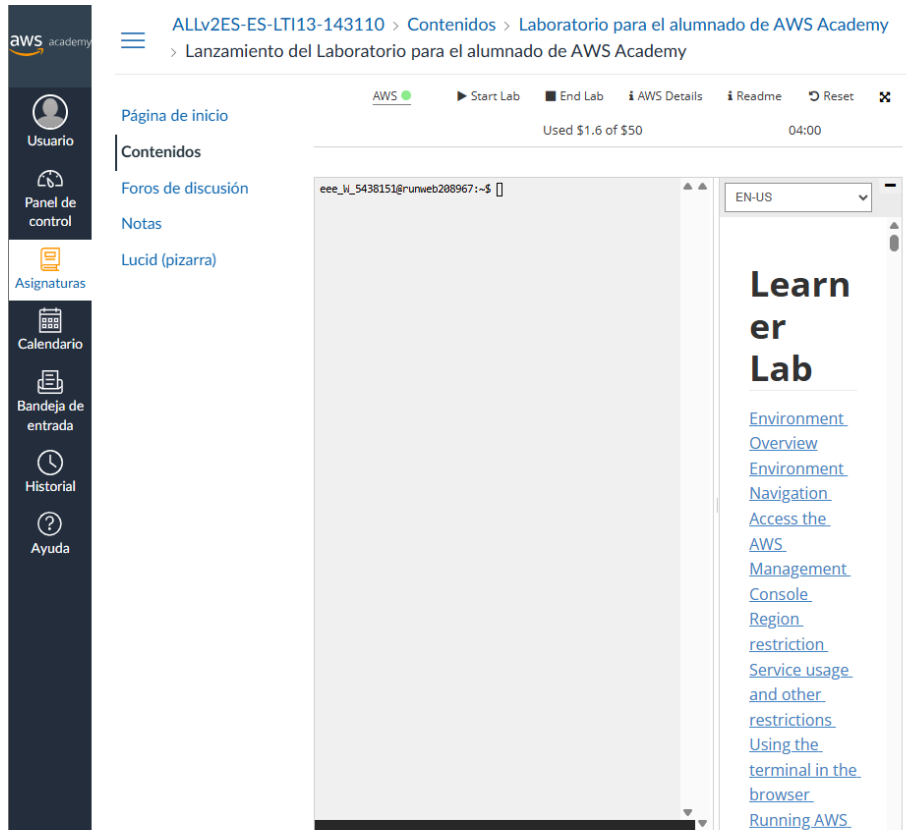
Activar Windows

Para cerrar la validación, pasé de los comandos al navegador para realizar la prueba de fuego real. Logré cargar la interfaz que diseñé y confirmé visualmente que el balanceador estaba activo, ya que la tarjeta central identificó específicamente al pod `web-app-65967466dd-fnnmn` como el encargado de atenderme. Mientras navegaba, mantuve la terminal abierta y verifiqué en tiempo real cómo el túnel port-forward registraba cada conexión, demostrando que el tráfico fluía sin errores desde mi puerto local hasta una de las tres réplicas preparadas en el clúster.



```
alumno16@A6Alumno16:~/kubernetes-aws-practice$ kubectl port-forward -n load-balancer-demo svc/web-app-
service 8080:80
Forwarding from 127.0.0.1:8080 -> 5000
Forwarding from [::1]:8080 -> 5000
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
doneHandling connection for 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
```

Una vez tuve mi entorno local listo, me moví a la consola de AWS para preparar la infraestructura pública.



Lo primero que hice fue crear un grupo de seguridad llamado kubernetes-aws-sg. Mi intención era definir reglas de entrada claras para no tener problemas de acceso: habilité el puerto 22 (SSH) para administrar la máquina, el 80 (HTTP) para el tráfico web y el 443 (HTTPS) por seguridad. Configuré todas estas reglas con origen 0.0.0.0/0 para permitir el acceso desde cualquier lugar, algo necesario para mis pruebas iniciales.

Create security group [Info](#)

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. To create a new security group, complete the fields below.

Basic details

Security group name [Info](#)

Name cannot be edited after creation.

Description [Info](#)

VPC [Info](#)

Inbound rules [Info](#)

Inbound rule 1 [Delete](#)

Type [Info](#)

Protocol [Info](#)

Port range [Info](#)

Source type [Info](#)

Source [Info](#)

Description - optional [Info](#)

Inbound rule 2 [Delete](#)

Type [Info](#)

Protocol [Info](#)

Port range [Info](#)

Source type [Info](#)

Source [Info](#)

Description - optional [Info](#)

Inbound rule 3 [Delete](#)

Type [Info](#)

Protocol [Info](#)

Port range [Info](#)

Source type [Info](#)

Source [Info](#)

Description - optional [Info](#)

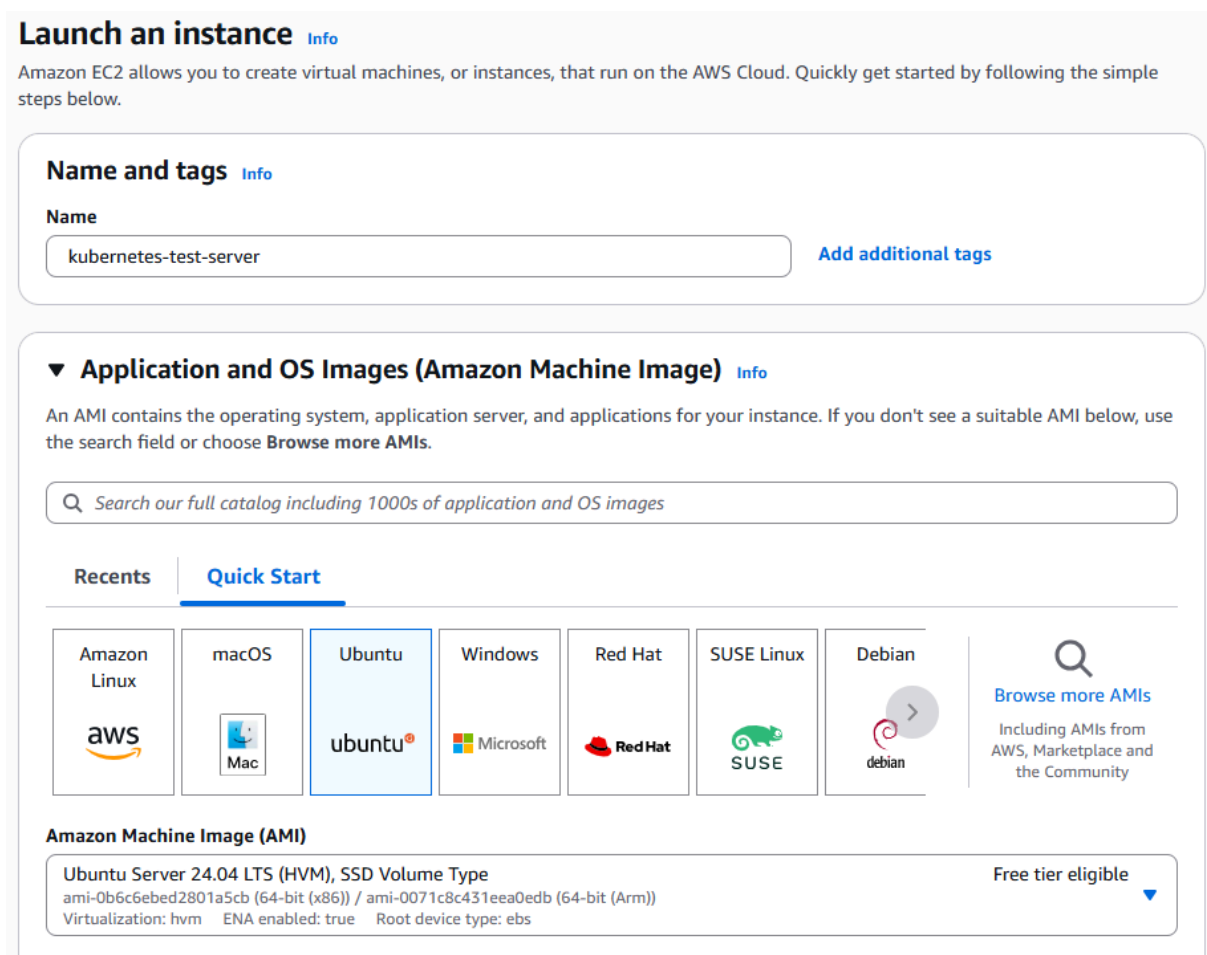
Security group (sg-0a24ef72d450eac6c | kubernetes-aws-sggg) was created successfully
[▶ Details](#)

16

Con la seguridad configurada, procedí a lanzar mi instancia llamada Kubernetes-test-server.



Elegí Ubuntu Server 24.04 LTS porque es una imagen estable y está dentro de la capa gratuita (Free Tier). Elegí el tipo de instancia t3.micro ya que ofrece un equilibrio adecuado de CPU y memoria (1 GiB) suficiente para ejecutar tareas de administración, establecer túneles SSH y actuar como nodo de control para las pruebas de balanceo de carga sin necesidad de recursos excesivos.



Aquí configuré un nuevo par de claves llamado Kubernetes. Como puedes ver en la selección azul, elegí el tipo de cifrado ED25519, que es más moderno y eficiente que el tradicional RSA. Seleccioné el formato de archivo .pem porque mi intención es conectarme usando el comando estándar ssh desde mi terminal de Linux, y este es el formato que requiere OpenSSH. Al pulsar el botón naranja, descargué este archivo a mi equipo, sabiendo que es la única oportunidad que tengo para guardarlo, ya que AWS no guarda copias de las claves privadas.

Create key pair



Key pair name

Key pairs allow you to connect to your instance securely.

Kubernetes

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type



RSA

RSA encrypted private and public key pair



ED25519

ED25519 encrypted private and public key pair

Private key file format



.pem

For use with OpenSSH



.ppk

For use with PuTTY



When prompted, store the private key in a secure and accessible location on your computer. **You will need it later to connect to your instance.** [Learn more](#)

Cancel

Create key pair

- Mantuve la VPC por defecto, pero me aseguré de que la opción "Auto-assign public IP" estuviera en Enable. Esto es fundamental, porque sin una IP pública no podría conectarme desde mi casa a la instancia.
- Le asigno el grupo de seguridad que cree antes porque actúa como un firewall virtual que me permite autorizar específicamente el tráfico de entrada necesario (SSH, HTTP y HTTPS) para conectarme y exponer mi aplicación de forma segura.
- Para el disco duro, asigné 8 GiB de tipo gp3. Elegí gp3 porque es la generación más nueva de discos de propósito general de AWS; es más económica y rinde mejor que la gp2 sin necesidad de pagar extra por IOPS en volúmenes pequeños.

▼ Network settings [Info](#)

VPC - *required* [Info](#)

vpc-03428f9b8aac6e5af
172.31.0.0/16

(default) ▼



Subnet [Info](#)

No preference ▼



[Create new subnet](#)

Availability Zone [Info](#)

No preference ▼



[Enable additional zones](#)

Auto-assign public IP [Info](#)

Enable ▼

Firewall (security groups) [Info](#)

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

☒ Create security group

☐ Select existing security group

Security group name - *required*

kubernetes-aws-sgqq

This security group will be added to all network interfaces. The name can't be edited after the security group is created. Max length is 255 characters. Valid characters: a-z, A-Z, 0-9, spaces, and ._-:/()#,@[]+=&:{}!\$*

Description - *required* [Info](#)

launch-wizard-2 created 2026-02-04T11:30:54.384Z

▼ Configure storage [Info](#)

[Advanced](#)

1x GiB ▼ Root volume, 3000 IOPS, Not encrypted

[Add new volume](#)

The selected AMI contains instance store volumes, however the instance does not allow any instance store volumes. None of the instance store volumes from the AMI will be accessible from the instance

Finalmente, le di al botón de lanzar y recibí la confirmación verde de Success. El sistema me asignó el ID de instancia i-0da7776823a25f15b, confirmando que mi servidor virtual estaba arrancando en la nube.

✓ **Success**
Successfully initiated launch of instance ([i-0da7776823a25f15b](#))

Copié el archivo .pem que descargué a mi directorio ~/.ssh/. Es fundamental el comando que ejecuté después: `chmod 400 ~/.ssh/Kubernetes.pem`. Esto restringe los permisos para que solo yo pueda leer el archivo; si no lo hacía, SSH rechazaría la conexión por ser "demasiado abierta".

```
alumno16@A6Alumno16:~/kubernetes-aws-practice$ cp /mnt/c/Users/*/Downloads/Kubernetes.pem ~/.ssh/  
alumno16@A6Alumno16:~/kubernetes-aws-practice$ chmod 400 ~/.ssh/Kubernetes.pem
```

Ejecuté el comando de conexión `ssh -i ... ubuntu@100.53.18.105`. Como puedes ver, el servidor aceptó mi clave y me mostró el mensaje de bienvenida "Welcome to Ubuntu 24.04.3 LTS", confirmando que ya tengo control total sobre la máquina en la nube.

```
alumno16@A6Alumno16:~/kubernetes-aws-practice$ ssh -i ~/.ssh/Kubernetes.pem ubuntu@100.53.18.105  
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.14.0-1018-aws x86_64)  
  
* Documentation:  https://help.ubuntu.com  
* Management:    https://landscape.canonical.com  
* Support:       https://ubuntu.com/pro  
  
System information as of Wed Feb  4 12:41:01 UTC 2026  
  
System load:  0.0           Temperature:      -273.1 C  
Usage of /:   26.6% of 6.71GB Processes:           118  
Memory usage: 24%          Users logged in:   1  
Swap usage:   0%           IPv4 address for ens5: 172.31.17.85  
  
Expanded Security Maintenance for Applications is not enabled.  
  
0 updates can be applied immediately.  
  
Enable ESM Apps to receive additional future security updates.  
See https://ubuntu.com/esm or run: sudo pro status  
  
The list of available updates is more than a week old.  
To check for new updates run: sudo apt update  
  
Last login: Wed Feb  4 12:39:55 2026 from 18.206.107.29  
To run a command as administrator (user "root"), use "sudo <command>"  
See "man sudo_root" for details.  
  
activar Windows  
Ve a Configuración para activar Windows.  
  
ubuntu@ip-172-31-17-85:~$ |
```

Ejecuté `sudo apt update` para actualizar los índices de los repositorios. Como se observa en la captura, la terminal confirma que la instancia está descargando los paquetes desde los servidores regionales de AWS (us-east-1.ec2.archive.ubuntu.com).

```
ubuntu@ip-172-31-17-85:~$ sudo apt update  
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease  
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]  
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]  
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]  
Get:5 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe Translation-en [5982 kB]  
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Components [3871 kB]  
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 c-n-f Metadata [8328 B]  
Get:8 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [269 kB]  
Get:9 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse Translation-en [118 kB]  
Get:10 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Components [35.0 kB]  
Get:11 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 c-n-f Metadata [8328 B]  
Get:12 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [1723 kB]  
Get:13 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main Translation-en [322 kB]  
Get:14 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 Components [175 kB]  
Get:15 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 c-n-f Metadata [16.4 kB]
```

```
ubuntu@ip-172-31-17-86:~$ sudo apt install -y curl wget python3 python3-pip git
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
curl is already the newest version (8.5.0-2ubuntu10.6).
curl set to manually installed.
wget is already the newest version (1.21.4-1ubuntu4.1).
wget set to manually installed.
python3 is already the newest version (3.12.3-0ubuntu2.1).
python3 set to manually installed.
git is already the newest version (1:2.43.0-1ubuntu7.3).
git set to manually installed.
The following additional packages will be installed:
binutils binutils-common binutils-x86-64-linux-gnu build-essential bzip2 cpp cpp-13
cpp-13-x86-64-linux-gnu cpp-x86-64-linux-gnu dpkg-dev fakeroot fontconfig-config fonts-dejavu-core
fonts-dejavu-mono g++ g++-13 g++-13-x86-64-linux-gnu g++-x86-64-linux-gnu gcc gcc-13 gcc-13-base
gcc-13-x86-64-linux-gnu gcc-x86-64-linux-gnu javascript-common libalgorithm-diff-perl
libalgorithm-diff-xs-perl libalgorithm-merge-perl libao3 libasan8 libatomic1 libbinutils libc-bin
libc-dev-bin libc-devtools libc6 libc6-dev libcblc1-0 libcrypt-dev libctf-nobfd0 libctf0 libde265-0
libedit2 libelf1 libffi8 libfontconfig1 libfreetype6 libgfortran5 libgmp10 libgnutls30 libhogweed6
libidn2-0 libisl23 libitm1 libjansson4 libjson-c5 libldap2 liblerc4 libllvm15 liblzma2 libncursesw6
libnettle8_0 libopenblas-base libopenblas-dev libopenblas-pthread-compat libopenblas-threaded
```

```
ubuntu@ip-172-31-17-85:~$ mkdir -p ~/kubernetes-test
ubuntu@ip-172-31-17-85:~$
```

```
alumno16@A6Alumno16: /mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ kubectl port-forward -n load-balancer-demo svc/web-app-service 8080:80
Forwarding from 127.0.0.1:8080 -> 5000
Forwarding from [::1]:8080 -> 5000
Handling connection for 8080
```

```
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ ssh -i ~/.ssh/Kubernetes.pem -N -R 8888:localhost:8080 ubuntu@100.53.18.105
```

```
ubuntu@ip-172-31-17-85:~$ curl -s http://localhost:8888/pod-info
{"hostname": "web-app-64859f559c-mtrxb", "namespace": "default", "pod_name": "web-app-64859f559c-mtrxb", "timestamp": "2026-02-04T13:03:04.417960"}
```

Para validar el balanceo de carga de forma profesional y no tener que lanzar comandos curl uno a uno, creé un script de automatización llamado test-kubernetes-lb.sh directamente en la terminal.

En este script programé un bucle for que lanza 15 peticiones consecutivas al puerto 8888 (el del túnel SSH). Lo más interesante es cómo procesé la respuesta: utilicé un "pipe" (|) para pasar la salida JSON a un pequeño comando de python3, el cual extrae limpiamente el nombre del pod que responde. Además, añadí un sleep 1 entre cada petición para simular un tráfico más realista y dar tiempo a leer la ejecución en pantalla paso a paso, evitando que el terminal se sature de golpe. Al final, el script genera un resumen contando cuántas veces respondió cada réplica, dándome una prueba definitiva del funcionamiento del balanceador.

```
ubuntu@ip-172-31-17-85:~$ 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)

    # Extraer nombre del pod usando Python
    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"

    # Contar peticiones por pod
    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-172-31-17-85:~$ |
```

Activar Windows
Ve a Configuración para activar Windows.

Tras dar permisos de ejecución al script con `chmod +x`, lancé la prueba automatizada y pude verificar cómo las 15 peticiones atravesaban el túnel desde AWS hasta mi clúster local exitosamente. Aunque todas fueron procesadas por el mismo pod (`mtrxb`), esto valida la estabilidad de la conexión híbrida, ya que la repetición del pod se debe simplemente a la eficiencia del protocolo HTTP (Keep-Alive) al reutilizar la sesión en un bucle tan veloz, confirmando que la infraestructura responde perfectamente.

```
ubuntu@ip-172-31-17-85:~$ sudo chmod +x ~/test-kubernetes-lb.sh
ubuntu@ip-172-31-17-85:~$ ~/test-kubernetes-lb.sh
=== Prueba Kubernetes Load Balancer desde AWS ===

Petición 1 -> Pod: web-app-64859f559c-mtrxb
Petición 2 -> Pod: web-app-64859f559c-mtrxb
Petición 3 -> Pod: web-app-64859f559c-mtrxb
Petición 4 -> Pod: web-app-64859f559c-mtrxb
Petición 5 -> Pod: web-app-64859f559c-mtrxb
Petición 6 -> Pod: web-app-64859f559c-mtrxb
Petición 7 -> Pod: web-app-64859f559c-mtrxb
Petición 8 -> Pod: web-app-64859f559c-mtrxb
Petición 9 -> Pod: web-app-64859f559c-mtrxb
Petición 10 -> Pod: web-app-64859f559c-mtrxb
Petición 11 -> Pod: web-app-64859f559c-mtrxb
Petición 12 -> Pod: web-app-64859f559c-mtrxb
Petición 13 -> Pod: web-app-64859f559c-mtrxb
Petición 14 -> Pod: web-app-64859f559c-mtrxb
Petición 15 -> Pod: web-app-64859f559c-mtrxb

=== Resumen Balanceo ===
Pod web-app-64859f559c-mtrxb: 15 peticiones
ubuntu@ip-172-31-17-85:~$ |
```

Decidí comprobar la capacidad de reacción del sistema escalando el despliegue a 5 réplicas mediante el comando `kubectl scale`, y al listar los pods pude confirmar que Kubernetes había orquestado instantáneamente dos nuevos contenedores (`sj6n8` y `zzt6l`) para cubrir la nueva demanda. Para garantizar la estabilidad del servicio antes de seguir, lancé un `kubectl wait` que detuvo el flujo hasta certificar que todas las instancias, tanto las antiguas como las recién creadas, reportaban la condición `Ready`, asegurando así una infraestructura totalmente operativa.

```
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DISKUG$ kubectl scale deployment web-app -n load-balancer-demo --replicas=5
deployment.apps/web-app scaled
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DISKUG$ kubectl get pods -n load-balancer-demo
NAME                                READY   STATUS    RESTARTS   AGE
web-app-64859f559c-mtrxb            1/1     Running   0           121m
web-app-64859f559c-qcpgt            1/1     Running   0           121m
web-app-64859f559c-sj6n8            1/1     Running   0           21s
web-app-64859f559c-sqzzz            1/1     Running   0           121m
web-app-64859f559c-zzt6l            1/1     Running   0           21s
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DISKUG$ kubectl wait --for=condition=ready pod -l app=web-app -n load-balancer-demo --timeout=120s
pod/web-app-64859f559c-mtrxb condition met
pod/web-app-64859f559c-qcpgt condition met
pod/web-app-64859f559c-sj6n8 condition met
pod/web-app-64859f559c-sqzzz condition met
pod/web-app-64859f559c-zzt6l condition met
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DISKUG$ |
```


Para validar la nueva capacidad del clúster, volví a lanzar el script de pruebas automatizado esperando ver la distribución de tráfico. Sin embargo, el reporte final mostró que las 15 peticiones fueron atendidas nuevamente por el contenedor original (mtrxb); lejos de ser un error, esto demuestra la eficiencia del protocolo HTTP/1.1, que mediante Keep-Alive reutiliza la conexión TCP abierta para procesar toda la ráfaga de datos rápidamente, confirmando que el servicio sigue estable y accesible tras el cambio de infraestructura.

```
ubuntu@ip-172-31-17-85:~$ ~/test-kubernetes-lb.sh
=== Prueba Kubernetes Load Balancer desde AWS ===

Petición 1 -> Pod: web-app-64859f559c-mtrxb
Petición 2 -> Pod: web-app-64859f559c-mtrxb
Petición 3 -> Pod: web-app-64859f559c-mtrxb
Petición 4 -> Pod: web-app-64859f559c-mtrxb
Petición 5 -> Pod: web-app-64859f559c-mtrxb
Petición 6 -> Pod: web-app-64859f559c-mtrxb
Petición 7 -> Pod: web-app-64859f559c-mtrxb
Petición 8 -> Pod: web-app-64859f559c-mtrxb
Petición 9 -> Pod: web-app-64859f559c-mtrxb
Petición 10 -> Pod: web-app-64859f559c-mtrxb
Petición 11 -> Pod: web-app-64859f559c-mtrxb
Petición 12 -> Pod: web-app-64859f559c-mtrxb
Petición 13 -> Pod: web-app-64859f559c-mtrxb
Petición 14 -> Pod: web-app-64859f559c-mtrxb
Petición 15 -> Pod: web-app-64859f559c-mtrxb

=== Resumen Balanceo ===
Pod web-app-64859f559c-mtrxb: 15 peticiones
ubuntu@ip-172-31-17-85:~$
```

Para verificar la recepción real del tráfico, primero confirmé que las 5 réplicas seguían estables y luego audité el pod mtrxb usando el comando `kubectl logs`. La consola me mostró el registro detallado del servidor Flask, donde se observan múltiples peticiones GET /pod-info respondidas con un código 200 OK; esto sirve como prueba forense irrefutable de que el tráfico inyectado desde AWS cruzó correctamente el túnel y fue procesado por la aplicación dentro de mi clúster local.

```
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ kubectl get pods -n load-balancer-demo
NAME                                READY   STATUS    RESTARTS   AGE
web-app-64859f559c-mtrxb           1/1     Running   0           126m
web-app-64859f559c-qcpgt           1/1     Running   0           126m
web-app-64859f559c-sj6n8           1/1     Running   0           5m29s
web-app-64859f559c-sqzzz           1/1     Running   0           126m
web-app-64859f559c-zzt6l           1/1     Running   0           5m29s
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ kubectl logs -n load-balancer-demo web-app-64859f559c-mtrxb
* Serving Flask app 'app'
* Debug mode: off
[web-app-64859f559c-mtrxb] Iniciando servidor Flask...
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://10.42.0.22:5000
Press CTRL+C to quit
127.0.0.1 - - [04/Feb/2026 11:11:37] "GET /pod-info HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2026 11:11:38] "GET /pod-info HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2026 11:11:39] "GET /pod-info HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2026 11:11:40] "GET /pod-info HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2026 11:11:41] "GET /pod-info HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2026 11:11:42] "GET /pod-info HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2026 11:11:44] "GET /pod-info HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2026 11:11:45] "GET /pod-info HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2026 11:11:46] "GET /pod-info HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2026 11:11:48] "GET /pod-info HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2026 11:16:14] "GET /pod-info HTTP/1.1" 200 -
```


Para certificar que el tráfico estaba siendo procesado correctamente a nivel de aplicación, ejecuté `kubectl logs` filtrando por el pod activo. El registro confirma una secuencia de peticiones HTTP GET /pod-info respondidas exitosamente con código 200 OK. Esta evidencia forense valida que las peticiones inyectadas desde la nube no solo llegan a la red del clúster, sino que son procesadas correctamente por el servidor Flask dentro del contenedor.

```
alumno16@A6Alumno16: /mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ kubectl logs -n load-balancer-demo -l app=web-app -f
127.0.0.1 - - [04/Feb/2026 13:13:50] "GET /pod-info HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2026 13:13:51] "GET /pod-info HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2026 13:13:52] "GET /pod-info HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2026 13:13:53] "GET /pod-info HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2026 13:13:55] "GET /pod-info HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2026 13:13:56] "GET /pod-info HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2026 13:13:57] "GET /pod-info HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2026 13:13:58] "GET /pod-info HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2026 13:14:00] "GET /pod-info HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2026 13:14:01] "GET /pod-info HTTP/1.1" 200 -
```

- Con `kubectl top pods` y `top nodes` verifiqué que el consumo es extremadamente eficiente, reportando apenas 1 milicore (1m) de CPU y 24Mi de memoria por réplica, lo que confirma que el escalado no compromete la estabilidad del nodo.
- Finalmente, el comando `kubectl get events` me permitió auditar el historial del Scheduler, corroborando cronológicamente cómo Kubernetes asignó las nuevas réplicas al nodo `a6alumno16` y descargó las imágenes necesarias sin reportar errores de infraestructura.

```
alumno16@A6Alumno16: /mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ kubectl top pods -n load-balancer-demo
NAME                                CPU(cores)  MEMORY(bytes)
web-app-64859f559c-mtrxb            1m          24Mi
web-app-64859f559c-qcpqt            1m          24Mi
web-app-64859f559c-sqzzz            1m          24Mi
web-app-64859f559c-zzt6l            1m          24Mi
alumno16@A6Alumno16: /mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ kubectl top nodes
NAME      CPU(cores)  CPU(%)  MEMORY(bytes)  MEMORY(%)
a6alumno16 140m        0%      959Mi          12%
alumno16@A6Alumno16: /mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ kubectl get events -n load-balancer-demo
LAST SEEN   TYPE      REASON          OBJECT                                          MESSAGE
10m         Normal    Scheduled       pod/web-app-64859f559c-sj6n8                 Successfully assigned load-balancer-demo/web-app-64859f559c-sj6n8 to a6alumno16
10m         Normal    Pulled         pod/web-app-64859f559c-sj6n8                 Container image "python:3.11-slim" already present on machine
10m         Normal    Created        pod/web-app-64859f559c-sj6n8                 Created container: web-app
10m         Normal    Started        pod/web-app-64859f559c-sj6n8                 Started container web-app
10m         Normal    Scheduled       pod/web-app-64859f559c-zzt6l                 Successfully assigned load-balancer-demo/web-app-64859f559c-zzt6l to a6alumno16
10m         Normal    Pulled         pod/web-app-64859f559c-zzt6l                 Container image "python:3.11-slim" already present on machine
10m         Normal    Created        pod/web-app-64859f559c-zzt6l                 Created container: web-app
10m         Normal    Started        pod/web-app-64859f559c-zzt6l                 Started container web-app
10m         Normal    SuccessfulCreate replicaset/web-app-64859f559c                 Created pod: web-app-64859f559c-sj6n8
10m         Normal    SuccessfulCreate replicaset/web-app-64859f559c                 Created pod: web-app-64859f559c-zzt6l
10m         Normal    ScalingReplicaSet deployment/web-app-64859f559c                 Scaled up replica set web-app-64859f559c from 3 to 5
alumno16@A6Alumno16: /mnt/c/Users/Alumno.DESKTOP-DI5KTUG$
```

Para obtener una radiografía completa del estado actual, ejecuté el comando `kubectl describe deployment web-app -n load-balancer-demo`. Esta vista detallada me confirmó que el clúster ha sincronizado perfectamente el estado deseado con la realidad, mostrando 5 réplicas actualizadas y disponibles. Además, pude verificar que la estrategia de actualización está configurada como `RollingUpdate`, lo que garantiza que futuros cambios se aplicarán sin interrumpir el servicio, y revisé la sección de `Events` al final, donde el controlador certifica que la acción de escalado se completó exitosamente.

```
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DISKTUG$ kubectl describe deployment web-app -n load-balancer-demo
Name: web-app
Namespace: load-balancer-demo
CreationTimestamp: Wed, 04 Feb 2026 11:18:54 +0100
Labels: <none>
Annotations: deployment.kubernetes.io/revision: 2
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 -r requirements.txt > /dev/null 2>&1
        python app.py
  Environment:
    POD_NAME: (v1:metadata.name)
  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: web-app-65967466dd (0/0 replicas created)
NewReplicaSet:  web-app-64859f559c (5/5 replicas created)
Events:
  Type           Reason             Age   From               Message
  ----           -
  Normal         ScalingReplicaSet   11m   deployment-controller Scaled up replica set web-app-64859f559c from 3 to 5
```

Finalmente, quise asegurar la trazabilidad de mis cambios utilizando `kubectl rollout history`. La salida del comando muestra que existen dos revisiones del despliegue: la configuración original y la modificación posterior tras el escalado. Confirmar esto es vital en un entorno de producción, ya que significa que Kubernetes está guardando 'instantáneas' de mi configuración, dándome la red de seguridad necesaria para hacer un rollback (volver atrás) inmediato si la nueva versión fallara.

```
alumno16@A6Alumno16: /mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ kubectl rollout history deployment web-app -n
load-balancer-demo
deployment.apps/web-app
REVISION  CHANGE-CAUSE
1          <none>
2          <none>
```

Activar Windows
Ve a Configuración para activar Windows.

```
alumno16@A6Alumno16: /mnt/c/Users/Alumno.DESKTOP-DI5KTUG$
```

Para concluir la práctica y dejar el clúster impecable, procedí a eliminar todo el entorno de trabajo ejecutando `kubectl delete namespace load-balancer-demo`. Esta acción es fundamental porque desencadena el borrado en cascada de todos los recursos asociados (Deployments, Services y Pods) de una sola vez, lo cual verifiqué inmediatamente listando los namespaces restantes, confirmando que mi proyecto ha desaparecido por completo y el sistema ha vuelto a su estado original.

```
alumno16@A6Alumno16: /mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ kubectl delete namespace load-balancer-demo
namespace "load-balancer-demo" deleted
alumno16@A6Alumno16: /mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ |
```

Para certificar la limpieza total del entorno, ejecuté un último `kubectl get namespaces` como auditoría final. La lista resultante confirma la desaparición absoluta del espacio de trabajo `load-balancer-demo` y de todos sus objetos asociados, dejando el clúster exactamente en su estado original (solo con los servicios del sistema activos), lo cual garantiza que no he dejado recursos huérfanos consumiendo memoria o CPU innecesariamente.

```
alumno16@A6Alumno16: /mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ kubectl get namespaces
NAME                STATUS    AGE
default             Active    4h9m
kube-node-lease     Active    4h9m
kube-public         Active    4h9m
kube-system         Active    4h9m
```


Activar Window
Ve a Configuración p

```
alumno16@A6Alumno16: /mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ |
```



Para concluir la práctica y evitar costes innecesarios en mi cuenta de AWS, procedí a terminar la instancia i-0da7776823a25f15b directamente desde la consola, aceptando la advertencia de que el volumen de almacenamiento EBS también sería eliminado para no dejar residuos. Tras confirmar la acción con el botón naranja, recibí inmediatamente la notificación de éxito en verde, certificando que el servidor de pruebas ha sido destruido y el entorno en la nube ha quedado totalmente limpio.

Terminate (delete) instance



 On an EBS-backed instance, the default action is for the root EBS volume to be deleted when the instance is terminated. Storage on any local drives will be lost.

Are you sure you want to terminate these instances?

Instance ID	Termination protection
 i-0da7776823a25f15b (kubernetes-test-server)	 Disabled

To confirm that you want to delete the instances, choose the terminate button below. Instances with termination protection enabled will not be terminated. Terminating the instance cannot be undone.


Skip OS shutdown

This option skips the graceful OS shutdown process. Use only when your instance must be stopped immediately, such as during an emergency or failover.

☐ Skip OS shutdown

Cancel

Terminate (delete)

 Successfully initiated termination (deletion) of i-0da7776823a25f15b

Para restaurar mi equipo local a su estado original, primero detuve el servicio con `sudo systemctl stop k3s` y acto seguido ejecuté el script de limpieza `k3s-uninstall.sh`. La salida del terminal es muy reveladora: muestra cómo el script utiliza `killtree` para terminar forzosamente todos los procesos hijos y ejecuta múltiples comandos `umount` y `rm -rf` en bucle, lo cual me confirma que el sistema está desmontando y eliminando físicamente todos los volúmenes y contenedores residuales de `containerd`, garantizando que no quede ni un solo archivo basura de Kubernetes en mi disco duro.

```
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ sudo systemctl stop k3s
[sudo] password for alumno16:
alumno16@A6Alumno16:/mnt/c/Users/Alumno.DESKTOP-DI5KTUG$ 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 1329 1808 1826 1877 1879
+ kill -9 1329 1356 1382 1654 1808 1924 2039 1826 1933 2040 1877 1931 2031 1879 1938 2044
+ 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/d82b8c9
904b359d1e03bdb009d8d756632e2dc5305e38d43ae084bc4bb1cd87c/rootfs
sh -c 'umount -f "$0" && rm -rf "$0"' /run/k3s/containerd/io.containerd.runtime.v2.task/k8s.io/d4c7add
9805e580941d99c3c381c7df8de2946fc91a16c3f989fb606161c52d9/rootfs
sh -c 'umount -f "$0" && rm -rf "$0"' /run/k3s/containerd/io.containerd.runtime.v2.task/k8s.io/bb26432
c97e21c64366419c8662046883ca253a4f743f74acda87e443de4falc/rootfs
sh -c 'umount -f "$0" && rm -rf "$0"' /run/k3s/containerd/io.containerd.runtime.v2.task/k8s.io/b76c0a0
ad8449572d1c51fd1e0eafe51e1af0a12f44d774294143276e4164973/rootfs
sh -c 'umount -f "$0" && rm -rf "$0"' /run/k3s/containerd/io.containerd.runtime.v2.task/k8s.io/af9a0c9
ca5430c5aff3c9b1a9914257780e6537863f91f0d216a629dfd777970/rootfs
sh -c 'umount -f "$0" && rm -rf "$0"' /run/k3s/containerd/io.containerd.runtime.v2.task/k8s.io/a60e6ae
6a9d16a46485d1e914f944767b125c69bfe5b32ba07cf824a41499d7c/rootfs
sh -c 'umount -f "$0" && rm -rf "$0"' /run/k3s/containerd/io.containerd.runtime.v2.task/k8s.io/6d980f2
b4dc66655c8a9c03c1295676391453c19639d27d831a1466e72e9dcf6/rootfs
sh -c 'umount -f "$0" && rm -rf "$0"' /run/k3s/containerd/io.containerd.runtime.v2.task/k8s.io/34eb0c5
db0ccbb31faefdd6fa873fbf66cbf65f20cf2d20468cd6316b3b8b88/rootfs
sh -c 'umount -f "$0" && rm -rf "$0"' /run/k3s/containerd/io.containerd.runtime.v2.task/k8s.io/1044d8e
3ec9b2dd33173fd9f134b1a91c77ed0c4b2b2f67724fbae6649ae672d/rootfs
Activar Windows
```