

# PRUEBA TÉCNICA DEVOPS - DEVSU

**Nombre:** Cristian Fernando Idrobo Montalvo

## 1. Introducción

Se detalla el proceso seguido para dockerizar, desplegar en Kubernetes y configurar un pipeline CI/CD completo para una aplicación Node.js proporcionada en la prueba técnica.

## 2. Problema Inicial

El reto inicial consiste en tomar una aplicación Node.js sencilla, dockerizarla, automatizar su integración y despliegue continuo mediante GitHub Actions, y desplegarla en un clúster Kubernetes local (Minikube). Además, se recomienda añadir puntos extras creando infraestructura en un proveedor cloud (AWS o Google Cloud) usando Terraform.

## 3. Pruebas de funcionamiento inicial

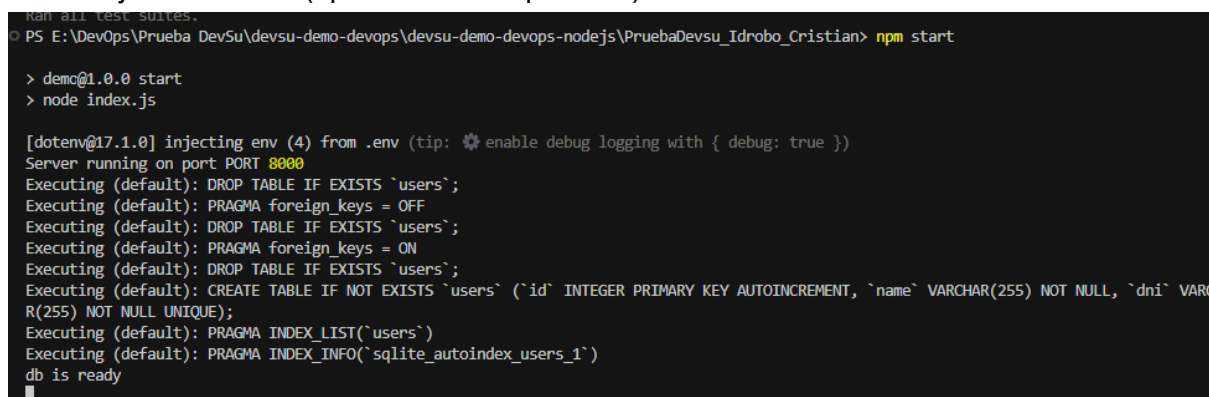
- Ejecución de pruebas (npm test)



File	Test Count	Pass Count	Fail Count	Percentage
All files	77.35	50	77.77	76.92
PruebaDevsu_Idrobo_Cristian	81.81	50	66.66	81.81
index.js	81.81	50	66.66	81.81
PruebaDevsu_Idrobo_Cristian/shared/database	100	50	100	100
database.js	100	50	100	100
PruebaDevsu_Idrobo_Cristian/shared/middleware	71.42	100	66.66	66.66
validateSchema.js	71.42	100	66.66	66.66
PruebaDevsu_Idrobo_Cristian/shared/schema	100	100	100	100
users.js	100	100	100	100
PruebaDevsu_Idrobo_Cristian/users	73.33	50	100	73.33
controller.js	68	50	100	68
model.js	100	100	100	100
router.js	100	100	100	100

Test Suites: 1 passed, 1 total  
Tests: 3 passed, 3 total  
Snapshots: 0 total  
Time: 3.978 s, estimated 5 s  
Ran all test suites.

- Ejecución local (npm install && npm start)



```
PS E:\DevOps\Prueba DevSu\devsu-demo-devops\devsu-demo-devops-nodejs\PruebaDevsu_Idrobo_Cristian> npm start

> demo@1.0.0 start
> node index.js

[dotenv@17.1.0] injecting env (4) from .env (tip: enable debug logging with { debug: true })
Server running on port PORT 8000
Executing (default): DROP TABLE IF EXISTS `users`;
Executing (default): PRAGMA foreign keys = OFF
Executing (default): DROP TABLE IF EXISTS `users`;
Executing (default): PRAGMA foreign keys = ON
Executing (default): DROP TABLE IF EXISTS `users`;
Executing (default): CREATE TABLE IF NOT EXISTS `users` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `name` VARCHAR(255) NOT NULL, `dni` VARCHAR(255) NOT NULL UNIQUE);
Executing (default): PRAGMA INDEX_LIST(`users`)
Executing (default): PRAGMA INDEX_INFO(`sqlite_autoindex_users_1`)
db is ready
```

## 4. Dockerizar la aplicación

### 4.1. Dockerfile

Se crea un Dockerfile para dockerizar la aplicación con el objetivo de crear un enfoque de seguridad, rendimiento y portabilidad. La imagen debe ser apta para entornos productivos en el pipeline y k8s, al igual que su ejecución para pruebas locales en Docker Compose.

En el dockerfile tenemos lo siguiente:

- **Imagen base node:18-alpine:** liviana, segura se utiliza para reducir el tamaño del contenedor y minimizar vulnerabilidades.
- **Usuario no root (appuser):** Se crea un usuario sin privilegios para ejecutar la app, evitando correr procesos como root en producción.
- **Instalación limpia (npm ci) y limpieza (npm prune):** En este paso se copia las dependencias, se instala las dependencias y eliminó dependencias de desarrollo para mantener la imagen liviana.
- **RUN chown -R appuser:appgroup /usr/src/app:** Se copia todo el código fuente y se otorgan los permisos adecuados al nuevo usuario.
- **Healthcheck:** Válida la salud del contenedor realizando una petición HTTP a un endpoint básico.

Dockerfile
<pre># ——— BASE DE PRODUCCIÓN FROM node:18-alpine  # Crear usuario sin privilegios RUN addgroup -S appgroup &amp;&amp; adduser -S appuser -G appgroup  WORKDIR /usr/src/app  # Sólo deps de prod COPY package*.json ./ RUN npm ci RUN npm prune --production # Copiar código COPY . . RUN chown -R appuser:appgroup /usr/src/app  # Cambiar a usuario sin privilegios USER appuser  # Puerto por defecto ENV NODE_ENV=production ENV PORT=8000 EXPOSE \${PORT}  # Healthcheck HEALTHCHECK --interval=30s --timeout=5s --start-period=10s \   CMD wget --quiet --tries=1 --spider http://localhost:\${PORT}/api/users    exit 1  CMD ["node", "index.js"]</pre>

**Nota:** Como se utiliza el RUN npm prune --production, para mantener la imagen liviana utilizamos el siguiente comando para que tener las dependencias de producción listas en especial “dotenv”

```
npm install dotenv --save-prod
```

## 4.2. Docker Compose

Construir el proyecto, facilitando pruebas locales al levantar la aplicación con una configuración completa usando docker-compose, tenemos lo siguiente:

- **build:** construye la imagen a partir del Dockerfile.
- **env\_file:** carga las variables de entorno desde .env.
- **volumes:** persiste la base de datos SQLite localmente.
- **restart:** reinicia el contenedor automáticamente si falla.
- **healthcheck:** valida la disponibilidad del endpoint.

### Docker Compose

```
version: '3.8'

services:
  app:
    build: .
    image: devsu_test:latest
    container_name: devsu_app

    # Mapea el puerto
    ports:
      - "${PORT:-8000}:8000"

    # Carga las vars de tu .env
    env_file:
      - .env

    # Volumen para persistir SQLite
    volumes:
      - ./dev.sqlite:/usr/src/app/dev.sqlite
    user: root
    # Reinicia hasta 3 veces si falla el contenedor
    restart: "on-failure:3"

    # Healthcheck
    healthcheck:
      test: ["CMD-SHELL", "curl --fail http://localhost:${PORT:-8000}/api/users || exit 1"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 10s
```

## 4.3. Comandos utiles para dockerizar:

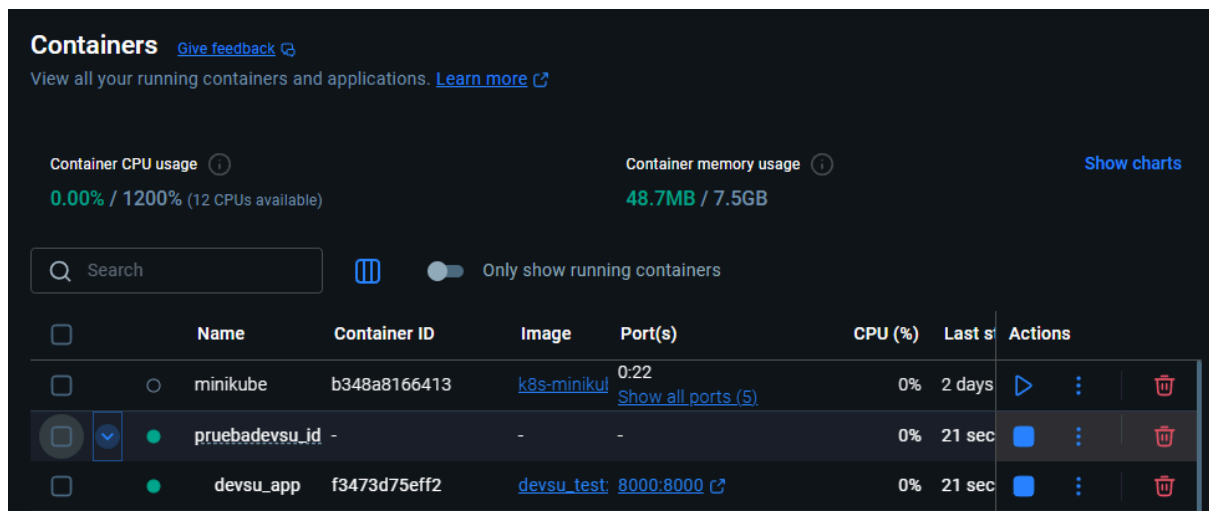
```
#Construir la imagen
docker build -t devsu_test .
```

```
#Levantar la imagen con compose
docker-compose up -d --build

#Revisar los contenedores corriendo
docker ps

#Revisar los logs de la app
docker logs devsu_app
```

#### 4.4. Pruebas de dockerización:



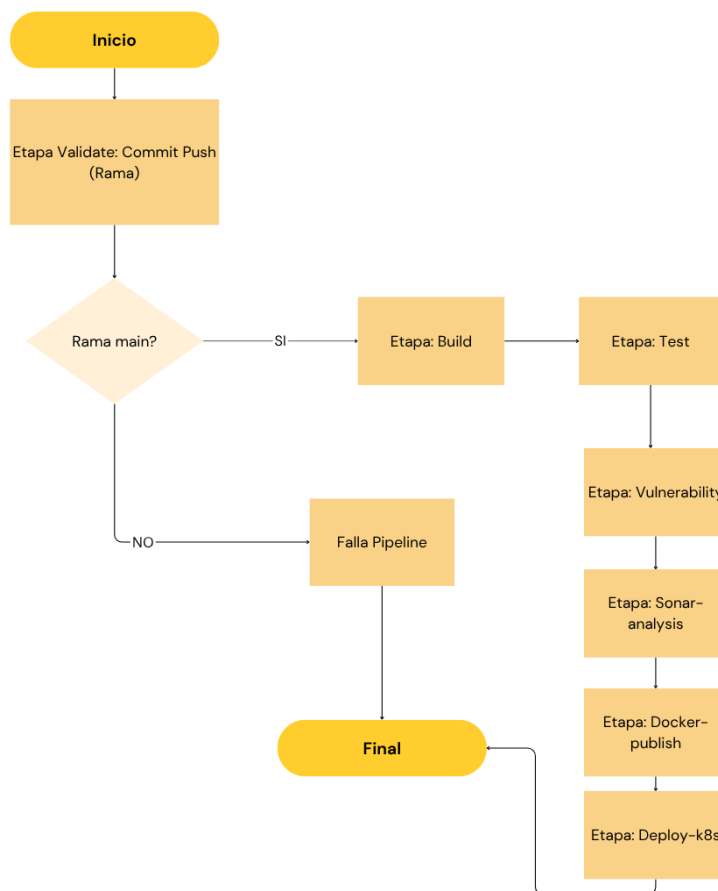
#### 5. Pipeline CI/CD con GitHub Actions

El pipeline de este proyecto está estructurado en 7 etapas “stages” principales que se ejecutan de forma secuencial para garantizar un flujo de trabajo robusto y controlado. Estas etapas son:

Etapa	Descripción
validate	Verifica que el push provenga de la rama principal (main). Solo si la rama coincide, el pipeline continúa; de lo contrario, se detiene.
build	Instala de forma limpia las dependencias (npm ci) y ejecuta el paso de build si existe (npm run build). Prepara el artefacto de la aplicación para las siguientes fases.
test	Ejecuta las pruebas unitarias con Jest bajo un entorno de test (NODE_ENV=test), asegurando que el código compila y que los casos de prueba pasan antes de avanzar.
vulnerability	Realiza un escaneo automático de vulnerabilidades en las dependencias (npm

	audit --audit-level=moderate) y guarda el reporte en JSON como artefacto descargable para revisión.
sonar-analysis	Integra SonarCloud para análisis estático de código y medición de cobertura. Usa sonar-scanner con los parámetros de organización, proyecto y credenciales definidas en secretos.
docker-publish	Construye la imagen Docker (multitag: :sha8 y :latest) y la publica en GitHub Container Registry. Garantiza que cada build genere un tag único (SHA) y un tag mutable (latest).
deploy-k8s	Despliega la aplicación en el clúster Kubernetes (Minikube) aplicando los manifests (Deployment, ConfigMap, Secret, Service, HPA) y forzando un rollout restart para que siempre use la última imagen.

### 5.1. Diagrama de flujo del Pipeline CI/CD



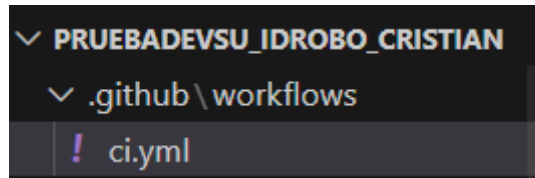
### 5.2. Etapas del Pipeline:

Primeramente se deben incluir las variables de entorno globales como REGISTRY, IMAGE\_OWNER, IMAGE\_REPO, que son la url del contenedor y variables de Github como el propietario y nombre del repositorio.

- **Etapla VALIDATE**

En esta etapa se comprueba que el push provenga de la rama main, y su propósito es evitar que otras ramas disparen el pipeline y por lo tanto los despliegues.

Como requisito previo necesitamos incluir en la raíz del proyecto el workflow del pipeline.



Si la etapa tiene éxito continúa al siguiente job caso contrario se detiene el pipeline.

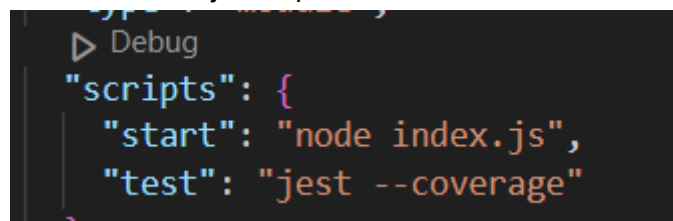
- **Etapla BUILD**

En esta etapa se instala dependencias exactamente según package-lock.json y ejecuta el script de build si existe, su propósito es preparar artefactos (transpiled code, bundles) para pruebas y análisis, y como requisito previo se necesita el package-lock.json, una vez pasado el job el código compilado se queda en el workspace de actions.

- **Etapla TEST**

En esta etapa se ejecutan las pruebas unitarias con Jest en modo “test”, para asegurar que el código cumple los casos de prueba antes de continuar.

Como requisito previo necesitamos la configuración de Jest en package.json, como resultado de tests si es exitoso reflejara reportes de cobertura en la consola.g



24	----- ----- ----- ----- ----- -----
25	File   % Stmts   % Branch   % Funcs   % Lines   Uncovered Line #s
26	----- ----- ----- ----- ----- -----
27	All files   77.35   50   77.77   76.92
28	PruebaDevsu_Idrobo_Cristian   81.81   50   66.66   81.81
29	index.js   81.81   50   66.66   81.81   10,16
30	PruebaDevsu_Idrobo_Cristian/shared/database   100   50   100   100
31	database.js   100   50   100   100   11
32	PruebaDevsu_Idrobo_Cristian/shared/middleware   71.42   100   66.66   66.66
33	validateSchema.js   71.42   100   66.66   66.66   13-15
34	PruebaDevsu_Idrobo_Cristian/shared/schema   100   100   100   100
35	users.js   100   100   100   100
36	PruebaDevsu_Idrobo_Cristian/users   73.33   50   100   73.33
37	controller.js   68   50   100   68   9-10,20,25-26,36,43-44
38	model.js   100   100   100   100
39	router.js   100   100   100   100
40	----- ----- ----- ----- ----- -----
41	Test Suites: 1 passed, 1 total
42	Tests: 3 passed, 3 total
43	Snapshots: 0 total
44	Time: 1.74 s
45	Ran all test suites.

- **Etapla VULNERABILITY SCAN**

En esta etapa se escanea vulnerabilidades de dependencias con npm audit y guarda un JSON, con el propósito de detectar posibles riesgos de seguridad en librerías antes de la publicación de la imagen, como requisito tener npm instalado, y su salida será un artefacto audit-report.json descargable desde la UI de Actions.

Triggered via push 2 days ago  
cfidrobo pushed · 725fbc8 · main  
Status: Success  
Total duration: 4m 2s  
Artifacts: 1

ci.yml  
on: push

validate → build → test → vulnerability → sonar analysis → docker publish → deploy k8s

Artifacts  
Produced during runtime

Name	Size	Digest
audit-report	2.79 KB	sha256:8cea98b6ac30771f16aa5da4085056499575fb50622e8ce3230632a2938b2f

- **Etapas SONAR ANALYSIS**

En esta etapa se ejecuta un análisis estático de código y cobertura en SonarCloud, con el propósito de verificar calidad del código, “code smells”, “bugs” y cobertura mínima, según el Quality Gate configurado, como requisitos previos necesitamos:

### Configurar en GitHub Secrets:

- SONAR\_TOKEN (token de SonarCloud)

Cristian Fernando Idrobo Montalvo

Profile Security Notifications Organizations Appearance

### Security

If you want to enforce security by not providing credentials of a real SonarCloud user to run your code scan or to invoke web services, you can provide a User Token as a replacement of the user login. This will increase the security of your installation by not letting your analysis user's password going through your network.

**Generate Tokens**

Enter Token Name Generate Token








**Existing Tokens**

Name	Last use	Created
SONAR_TOKEN	Never	7 July 2025

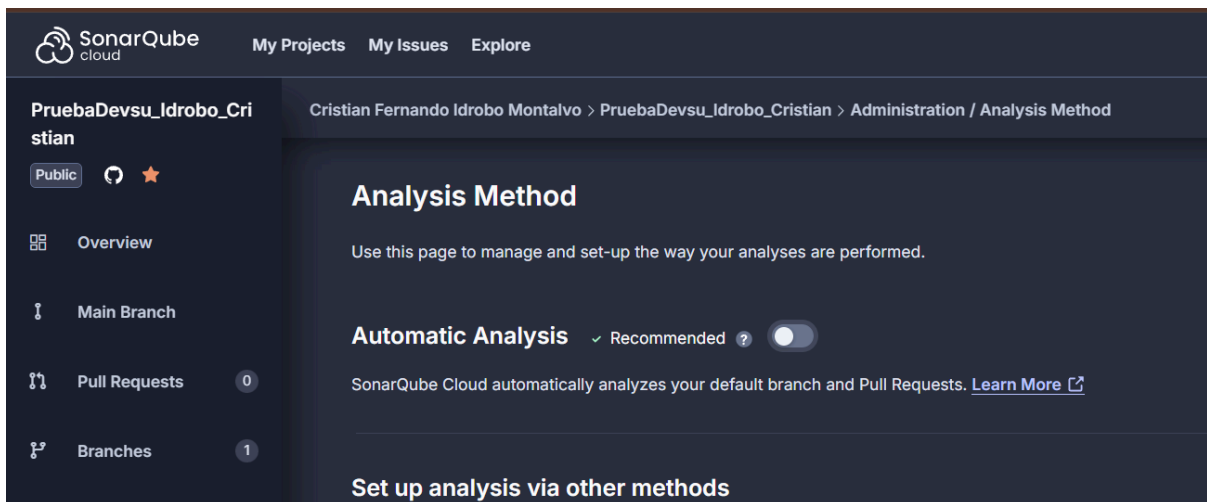
Revoke

- SONAR\_ORG (organización en SonarCloud)
- SONAR\_PROJECT\_KEY (clave de proyecto)

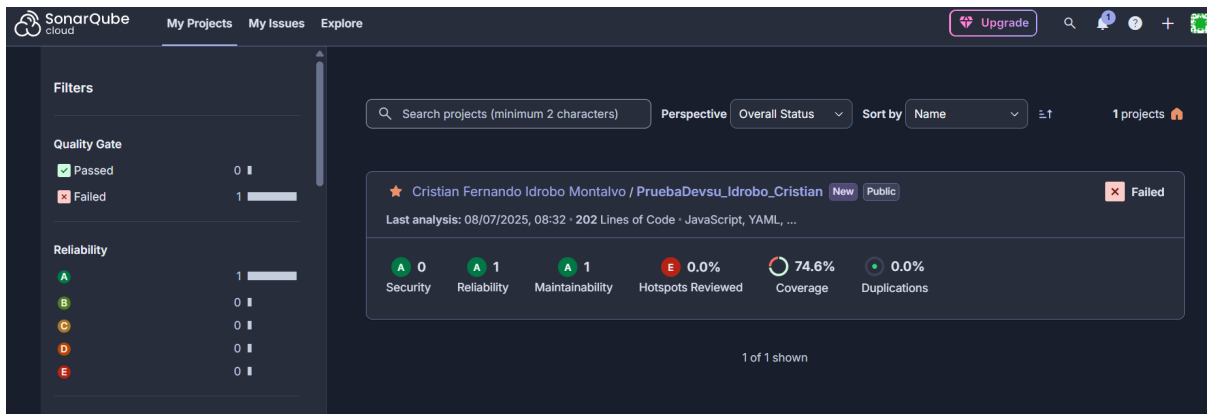
Una vez se obtengan esas variables se incluyen en los secretos del repositorio.

Repository secrets			New repository secret
Name		Last updated	
GHCR_PAT		4 days ago	 
SONAR_ORG		4 days ago	 
SONAR_PROJECT_KEY		4 days ago	 
SONAR_TOKEN		4 days ago	 

Igualmente se debe deshabilitar la opción de método automático de análisis para que siempre se analice cuando el pipeline se ejecute.



Como salida se puede observar los dashboards de análisis en SonarCloud

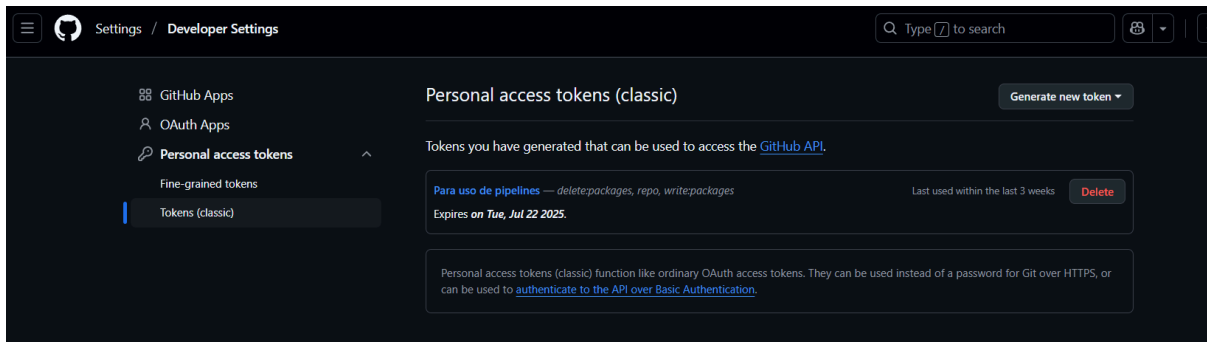


- **Etapla DOCKER PUBLISH**

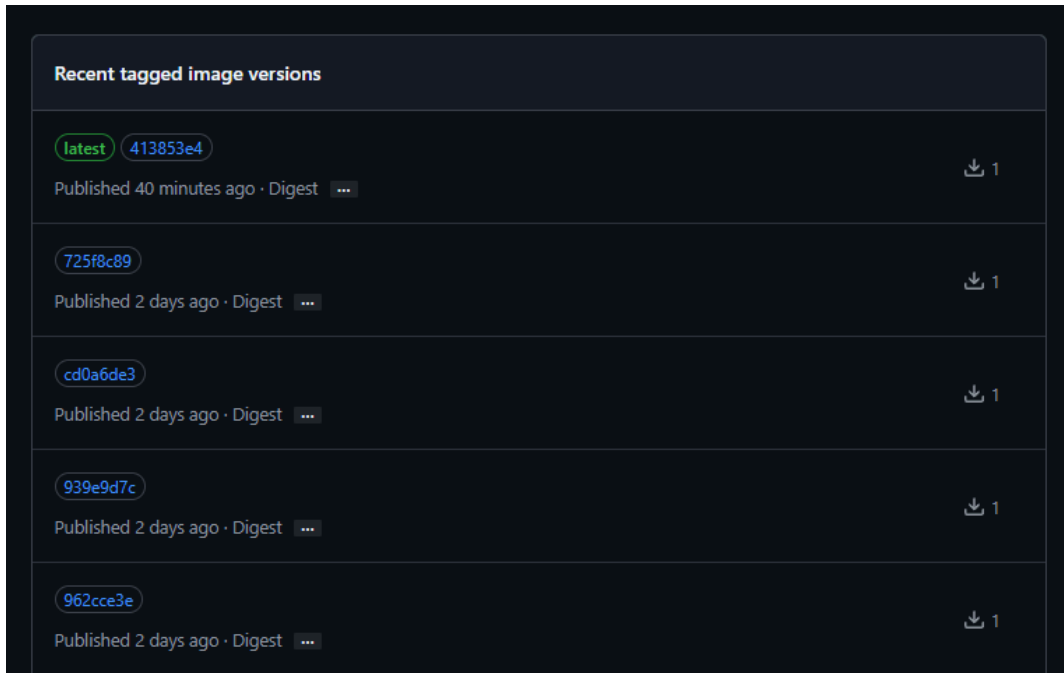
En esta etapa se configura Docker Buildx, se loguea en GitHub Container Registry, Determina un tag único (SHA de 8 caracteres) para los commits, se construye la imagen y push con un tag (sha8) y (latest), para tener un histórico de imágenes que por commits y una latest haciendo referencia al último push que está bien.

Su propósito es generar y publicar la imagen Docker para su posterior despliegue, como requisitos previos debemos configurar un GHCR\_PAT el cual es el personal access token con permisos de escritura de paquetes.





Como salda las imágenes estarán disponibles en los paquetes del repositorio.



- **Etapla DEPLOY K8S**

En esta etapa se aplica los manifiestos de Kubernetes (Deployment, ConfigMap, Secret, Service, HPA), Forza un rollout restart para que use la imagen latest, su propósito es desplegar la última versión de la aplicación en el clúster Minikube.

Como requisito previo:

- Runner self-hosted con Minikube configurado.

Crear un runner desde el repositorio de Github e instalarlo en nuestra máquina local windows.

```

  _____
 |  G I T H U B  |  A C T I O N S
 |  _____  |
 |  Self-hosted runner registration
 |  _____  |
 |
# Authentication
#
✓ Connected to GitHub
# Runner Registration
Enter the name of the runner group to add this runner to: [press Enter for Default]
Enter the name of runner: [press Enter for DESKTOP-R329FU5] local-runner
This runner will have the following labels: 'self-hosted', 'Windows', 'X64'
Enter any additional labels (ex. label-1,label-2): [press Enter to skip]
✓ Runner successfully added
✓ Runner connection is good
# Runner settings
Enter name of work folder: [press Enter for _work]
✓ Settings Saved.
Would you like to run the runner as service? (Y/N) [press Enter for N] Y
User account to use for the service [press Enter for NT AUTHORITY\Servicio de red]
Granting file permissions to 'NT AUTHORITY\Servicio de red'.
Service actions.runner.cfidrobo-PruebaDevsu_Idrobo_Cristian.local-runner successfully installed
Service actions.runner.cfidrobo-PruebaDevsu_Idrobo_Cristian.local-runner successfully set recovery option
Service actions.runner.cfidrobo-PruebaDevsu_Idrobo_Cristian.local-runner successfully set to delayed auto start
Service actions.runner.cfidrobo-PruebaDevsu_Idrobo_Cristian.local-runner successfully configured
Waiting for service to start...
Service actions.runner.cfidrobo-PruebaDevsu_Idrobo_Cristian.local-runner started successfully

```

- Variables de entorno en el runner:
  - KUBECONFIG apuntando al archivo kubeconfig del usuario.

### Habilitar scripts permanentemente en el sistema desde PowerShell

Set-ExecutionPolicy RemoteSigned -Scope LocalMachine -Force

- Permisos de red (icaccls) para que el servicio de red pueda leer .kube y .minikube.

### Otorgar acceso de lectura y escritura a los servicios de red

# Otorga Control total (lectura+escritura) recursivamente a "Servicio de red" sobre .kube  
icaccls "C:\Users\Cristian\.kube" /grant "NT AUTHORITY\Servicio de red):(OI)(CI)F /T

# Y lo mismo para todo el perfil de Minikube  
icaccls "C:\Users\Cristian\.minikube" /grant "NT AUTHORITY\Servicio de red):(OI)(CI)F /T

Como salida obtenemos pods actualizados en estado Running con la versión más reciente de la imagen.

```

C:\Users\Cristian>kubectll get pods
NAME                                READY    STATUS    RESTARTS    AGE
nodejs-app-dfffd7489d-lwtkm        1/1      Running   1 (85s ago)  3m19s
nodejs-app-dfffd7489d-qp8k4        1/1      Running   1 (59s ago)  2m40s

```

### 5.3. Workflow de GitHub Actions

```
name: CI Pipeline

on:
  push:

env:
  REGISTRY: ghcr.io
  IMAGE_OWNER: ${ github.repository_owner }
  IMAGE_REPO: ${ github.event.repository.name }

jobs:
  # 1) VALIDATE: asegurar que venga de main
  validate:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Ensure branch is main
        run: |
          if [[ "${GITHUB_REF}" != "refs/heads/main" ]]; then
            echo "❌ Not on main (${GITHUB_REF}). Exiting."
            exit 1
          fi
          echo "✅ On main, proceeding."

  # 2) BUILD: instalar deps y (opcional) build
  build:
    needs: validate
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '18'
      - name: Install dependencies
        run: npm ci
      - name: Build (if defined)
        run: npm run build || echo "No build step"

  # 3) TEST: correr unit tests
  test:
    needs: build
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v3
        with:
          node-version: '18'
      - name: Install deps
        run: npm ci
      - name: Run tests
```

```
env:  
  NODE_ENV: test  
run: npm test
```

#### # 4) VULNERABILITY SCAN: npm audit

vulnerability:

needs: test

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4
- uses: actions/setup-node@v3  
 with:  
 node-version: '18'
- name: Audit dependencies and save report  
 run: |  
 npm ci  
 npm audit --audit-level=moderate --json > audit-report.json || true
- uses: actions/upload-artifact@v4  
 with:  
 name: audit-report  
 path: audit-report.json

#### # 5) SONAR ANALYSIS: SonarCloud

sonar-analysis:

needs: vulnerability

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4
- name: Setup Node.js  
 uses: actions/setup-node@v3  
 with:  
 node-version: '18'
- name: Install deps (incluye sonar-scanner si lo usas como dev-dep)  
 run: npm ci
- name: Run tests with coverage  
 env:  
 NODE\_ENV: test  
 run: npm test
- name: SonarCloud Scan  
 env:  
 SONAR\_TOKEN: \${{ secrets.SONAR\_TOKEN }}  
 run: |  
 npx sonar-scanner \  
 -Dsonar.organization=\${{ secrets.SONAR\_ORG }} \  
 -Dsonar.projectKey=\${{ secrets.SONAR\_PROJECT\_KEY }} \  
 -Dsonar.host.url=https://sonarcloud.io \  
 -Dsonar.login=\$SONAR\_TOKEN \  
 -Dsonar.sources=. \  
 -Dsonar.tests=. \  
 -Dsonar.test.inclusions="\*\*/\*.test.js" \  
 -Dsonar.exclusions=

-Dsonar.javascript.lcov.reportPaths=coverage/lcov.info

# 6) BUILD & PUSH: image tagged con SHA corta y 'latest'

docker-publish:

needs: sonar-analysis

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4
- name: Setup Docker Buildx
  - uses: docker/setup-buildx-action@v3
- name: Login to GHCR
  - uses: docker/login-action@v3
  - with:
    - registry: \${{ env.REGISTRY }}
    - username: \${{ github.actor }}
    - password: \${{ secrets.GHCR\_PAT }}
- name: Determine image tag
  - id: tag
  - run: echo "sha=\${GITHUB\_SHA::8}" >> \$GITHUB\_OUTPUT
- name: Build & push
  - run: |
    - REPO\_LOWER=\$(echo "\${{ env.IMAGE\_REPO }}" | tr '[:upper:]' '[:lower:]')
    - IMAGE=\${{ env.REGISTRY }}/\${{ env.IMAGE\_OWNER }}/\$REPO\_LOWER
    - docker build -t \$IMAGE:\${{ steps.tag.outputs.sha }} -t \$IMAGE:latest .
    - docker push \$IMAGE:\${{ steps.tag.outputs.sha }}
    - docker push \$IMAGE:latest

# ) DEPLOY K8S

deploy-k8s:

needs: docker-publish

runs-on: self-hosted

env:

KUBECONFIG: C:\Users\Cristian\.kube\config

steps:

- uses: actions/checkout@v4
- name: Check context
  - shell: cmd
  - run: |
    - echo KUBECONFIG = %KUBECONFIG%
    - kubectl config current-context
    - kubectl get nodes
- name: Deploy to local K8s
  - shell: cmd
  - run: |
    - setlocal enabledelayedexpansion
    - REM Contexto minikube
    - kubectl config use-context minikube
    - kubectl apply -f k8s/deployment.yaml
    - kubectl apply -f k8s/configmap.yaml

```
kubectl apply -f k8s/secret.yaml
```

REM Forzar rollout para que tire la última imagen latest  
`kubectl rollout restart deployment/nodejs-app`

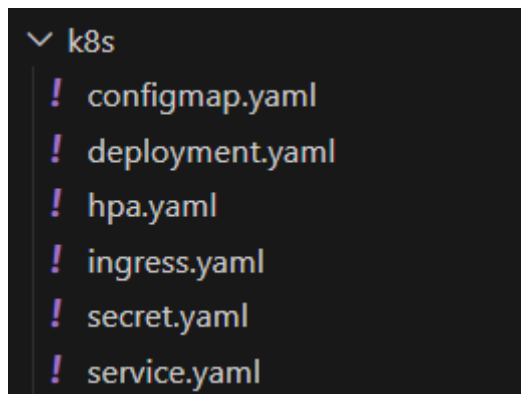
REM Esperar hasta que el nuevo rollout esté listo (timeout 120s)  
`kubectl rollout status deployment/nodejs-app --timeout=120s`

REM Service y HPA  
`kubectl apply -f k8s/service.yaml`  
`kubectl apply -f k8s/hpa.yaml`

```
endlocal
```

## 6. Configuración Kubernetes

En la raíz del proyecto incluir los archivos necesarios de K8s.



**configmap.yaml:** Su propósito es externalizar valores de configuración no sensibles (como el puerto y la ruta de la base de datos) para inyectarlos como variables de entorno en los pods sin tener que reconstruir la imagen Docker.

**secret.yaml:** Sirve para almacenar de forma segura credenciales y datos sensibles (como usuario y contraseña de la base de datos), se codifican en base64 y manteniéndolos fuera del código fuente.

**deployment.yaml:** Define el despliegue de la aplicación: número de réplicas igual a 2, estrategia de rolling update, contenedor con imagen :latest, probes de readiness y liveness para el health, política de pull (Always) y la inyección de ConfigMap y Secret como variables de entorno.

**service.yaml:** Expone internamente el Deployment bajo un nombre DNS (nodejs-app-svc) y mapea el puerto 80 del servicio al puerto 8000 del contenedor, permitiendo que otros recursos del clúster lo consuman.

**hpa.yaml:** Configura el HorizontalPodAutoscaler para escalar automáticamente las réplicas del Deployment según el uso de CPU, manteniéndolo entre 2 y 5 pods para adaptarse a la carga.

**ingress.yaml:** Exponer el servicio al exterior del clúster mediante un hostname (nodejs-app.local) y reglas de ruta, aprovechando un Ingress Controller (por ejemplo nginx) y simplificando el acceso HTTP sin necesidad de asignar puertos dinámicos.

## 7. Pasos Previos antes de hacer un push a GitHub.

### 7.1. Iniciar Minikube

Se recomienda tener levantado minikube con un contenedor docker e iniciarlo con:

```
minikube stop  
minikube start
```

	Name	Container ID	Image	Port(s)	CPU (%)	Last s	Actions
	minikube	b348a8166413	k8s-miniku		15.21%	54 min	

### 7.2. Editar el archivo hosts

En windows se edita el archivo con la dirección de nuestra aplicación.

```
Archivo  Editar  Ver  H1  B  
#  
127.0.0.1 localhost  
::1 localhost  
[redacted]  
127.0.0.1 nodejs-app.local  
# Added by Docker Desktop  
[redacted] host.docker.internal  
[redacted] gateway.docker.internal  
# To allow the same kube context to work on the host and the container:  
127.0.0.1 kubernetes.docker.internal  
# End of section
```

### 7.2. Redirección de puertos locales

Permite acceder a un servicio Kubernetes desde tu máquina local redirigiendo el puerto 80 del servicio nodejs-app-svc al puerto 8000 del equipo.

```
kubectrl port-forward svc/nodejs-app-svc 8000:80
```

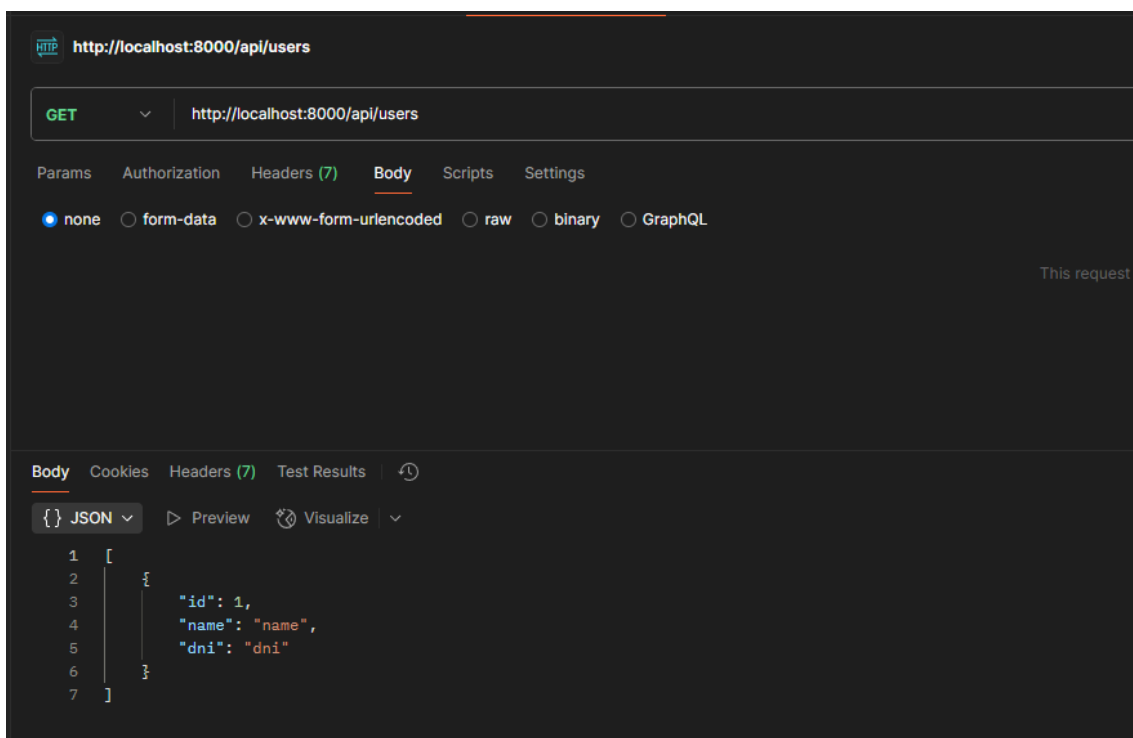
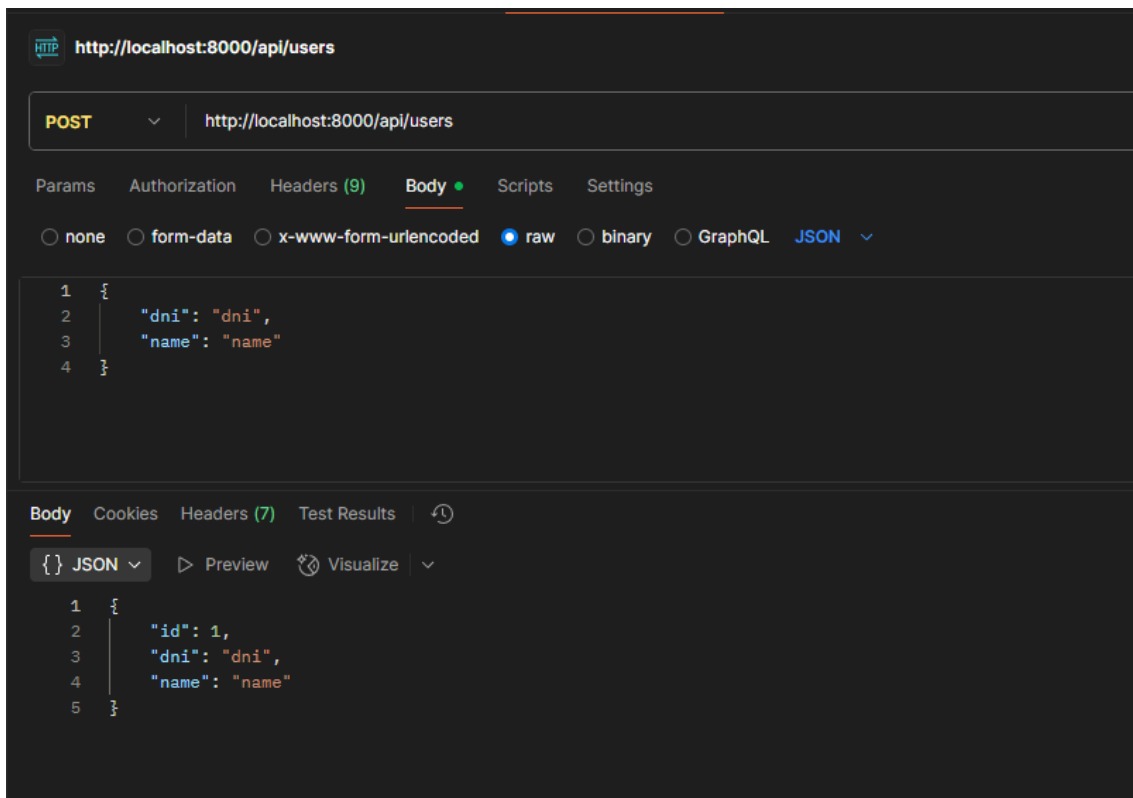
### 7.3. Escalar el despliegue a cero réplicas

Para realizar pruebas se sugiere escalar el despliegue a 0 réplicas para volver a hacer un deploy con el pipeline.

```
kubectrl scale deployment nodejs-app --replicas=0
```

## 8. Pruebas

Una vez que se desplegó correctamente la aplicación en el equipo se procede hacer pruebas con Postman.



Para verificar los pods encendidos y los logs se puede ocupar lo siguiente:

```
kubectl get pods
```

```
kubectl logs deployment/nodejs-app
```