code quality A | CI Pipeline passing

# 🧩 DevOps: Trabajo Práctico 1

Bienvenido al repositorio del trabajo práctico 1 del cursado 2025 de DevOps, realizado por:

- Aldo Omar Andres.
- Agustín Nicolás Bravo Pérez.

Links relevantes:

- Sitio web.
- Railway.
- Repositorio.
- Presentación.
- Consigna.

## ✨ Aplicación: Lista de Tareas

Construimos una simple *todo application* con los siguientes componentes:

- Una app web desarrollada con React y Vite.
- Un servidor desarrollado con TypeScript y Express.js.
- Una base de datos Redis.

La aplicación permite ver la lista de tareas, crear tareas nuevas y actualizar o eliminar tareas existentes. Su arquitectura de software es la siguiente:

```
graph LR
    User(("Usuario")) --> Frontend["Frontend<br/>(React + Vite)"]
    Frontend --> Backend["Backend<br/>(TypeScript + Express.js)"]
    Backend --> Redis["Base de datos<br/>(Redis)"]
```

Flujo de datos:

1. El usuario interactúa con la UI (frontend).
2. React envía peticiones a la API REST (backend).
3. El backend procesa las peticiones y envía peticiones a Redis.
4. Redis responde las peticiones del backend, quien luego responde al frontend.

## 📂 Estructura del Proyecto

```
devops-practice
├── .github          # Definición de la GitHub Action
├── backend          # Servidor backend con TypeScript y Express.js
│   ├── package.json
│   └── Dockerfile
├── frontend         # App web frontend con React y Vite
│   ├── src
│   │   └── index.jsx
│   ├── package.json
│   └── Dockerfile
```

```
├── docker-compose.yml
└── README.md
```

## 👩‍💻 Desarrollo

Requisitos para levantar el proyecto:

- Docker.

1. Clonar el repositorio:

```
git clone https://github.com/AldoOmarAndres/devops-practice.git
cd devops-practice
```

2. Construir y ejecutar la aplicación usando Docker Compose:

```
docker compose up --build
```

3. Visitar la UI en `http://localhost:3000` y la API en `http://localhost:3001`.

Se pueden definir las siguientes variables de entorno:

- `frontend/.env`:

```
VITE_API_URL=http://localhost:3001/api
```

- `backend/.env`:

```
REDIS_URL=redis://localhost:6379
PORT=80
```

## 🚀 Despliegue

Se utiliza Railway para desplegar la aplicación. Se creó un proyecto `devops-practice`.

Con los siguientes comandos interactivos se crean tres servicios `frontend`, `backend` y `redis`:

```
railway login
railway link

railway add -s backend \
   -i agustinbravop/devops-practice-backend:latest \
   -v "REDIS_URL=redis://redis:6379?family=6" \
   -v "PORT=80"
```

Es necesario ir manualmente al servicio `backend` y generar una URL para habilitarlo al público. Esa URL `https://backend-production-ced8.up.railway.app` luego se pone en el paso `build-frontend` de la GitHub Action como el argumento `VITE_API_URL` agregando un `/api` al final. Esto es necesario porque Vite compila la aplicación al

momento de construir la imagen y no procesa variables de entorno en tiempo de ejecución. Existen workarounds para esto pero en este caso se prefirió mantener una solución simple.

```
railway add -s frontend \
   -i agustinbravop/devops-practice-frontend:latest

railway add -s redis \
   -i redis:7-alpine
```

Luego se necesita manualmente habilitar al público los servicios `frontend` y `backend`. También es necesario en ambos servicios habilitar los redespliegues automáticos cuando se actualiza la imagen con etiqueta `latest`. Esto no resultó simple de automatizar, demostrando un inconveniente de Railway: prioriza la experiencia de la GUI por sobre la CLI.

Se tiene una GitHub Action para el despliegue. Esta GitHub Action requiere las siguientes variables y secrets:

```
DOCKERHUB_USERNAME=
DOCKERHUB_TOKEN=
RAILWAY_TOKEN=
```

Pasos de un despliegue al hacer un `git push`:

1. GitHub Actions ejecuta todos los pasos de integración continua.
2. GitHub Actions construye las imágenes de contenedores y las publica en Docker Hub.
3. GitHub Actions notifica a Railway para redesplegar los servicios, lo cual descarga la imagen nueva de Docker Hub.

```
graph LR
    subgraph "Desarrollo"
        DEV[Dev]
        GIT[Repositorio<br/>GitHub]
    end

    subgraph "Pipeline CI/CD"
        GA[Integración<br/>continua]
        BUILD[Construir imagenes<br/> y pushear a<br/>Docker Hub]
        REDEPLOY[Redeploy]
    end

    PUSH[Docker Hub]

    subgraph "Producción en Railway"
            RF[Frontend]
            RB[Backend]
            RR[Redis]
    end

    %% Deployment flow
    DEV -->|git push| GIT
    GIT --> GA
    GA -->|build| BUILD
    BUILD -->|push| PUSH
    BUILD --> REDEPLOY
    REDEPLOY -->|redeploy| RF
    REDEPLOY -->|redeploy| RB
    RF -.->|pull latest| PUSH
    RB -.->|pull latest| PUSH

    %% Styling
    classDef dockerhub fill:#0ea5e9,stroke:#0284c7,stroke-width:2px,color:#ffffff
    classDef cicd fill:#8b5cf6,stroke:#7c3aed,stroke-width:2px,color:#ffffff
```

```
    classDef production fill:#10b981,stroke:#059669,stroke-width:2px,color:#ffffff
    classDef development fill:#f59e0b,stroke:#d97706,stroke-width:2px,color:#ffffff

    class DEV,GIT development
    class GA,BUILD,REDEPLOY cicd
    class RF,RB,RR production
    class PUSH dockerhub
```

## ⚒️ Tareas Pendientes

Esta lista NO es exhaustiva!

- [x] Migrar el frontend a Vite (Create React App está deprecado).
- [x] Agregar un paso de lint a la pipeline de CI.
- [x] Construir contenedores y publicarlos en un Package Registry.
- [x] Agregar la funcionalidad de eliminar tareas.
- [x] Agregar tests al frontend.
- [x] Opcional: agregar una UI de Redis.
- [x] Opcional: mejorar la UX de la app.
- [x] Opcional: probar configurar Railway para que el despliegue sea automático.
- [x] Documentar la arquitectura con un diagrama.
- [ ] Opcional: probar una herramienta de monorepos para ejecutar scripts (alguna alternativa a Turborepo).
- [ ] Opcional: desplegar un Redis Insight en lugar de usar la UI integrada en Railway.
- [x] Preparar un informe o presentación que resuma resultados obtenidos, dificultades encontradas, y oportunidades de mejora.