

**UNIVERSIDAD PRIVADA BOLIVIANA
LA PAZ
FACULTAD DE INGENIERÍA Y ARQUITECTURA**



**Proyecto Segundo Parcial
Despliegue con Docker + EC2 + CI/CD**

Estudiantes:

Ivan Iver Poma Maidana - 70419
Wendy Evelyn Cáceres Vasquez - 68874
Andres Raul Sanchez Andrade - 69017

Docente: Rayner Mendieta Villalba

La Paz – Bolivia – 10 de Diciembre de 2025

Índice

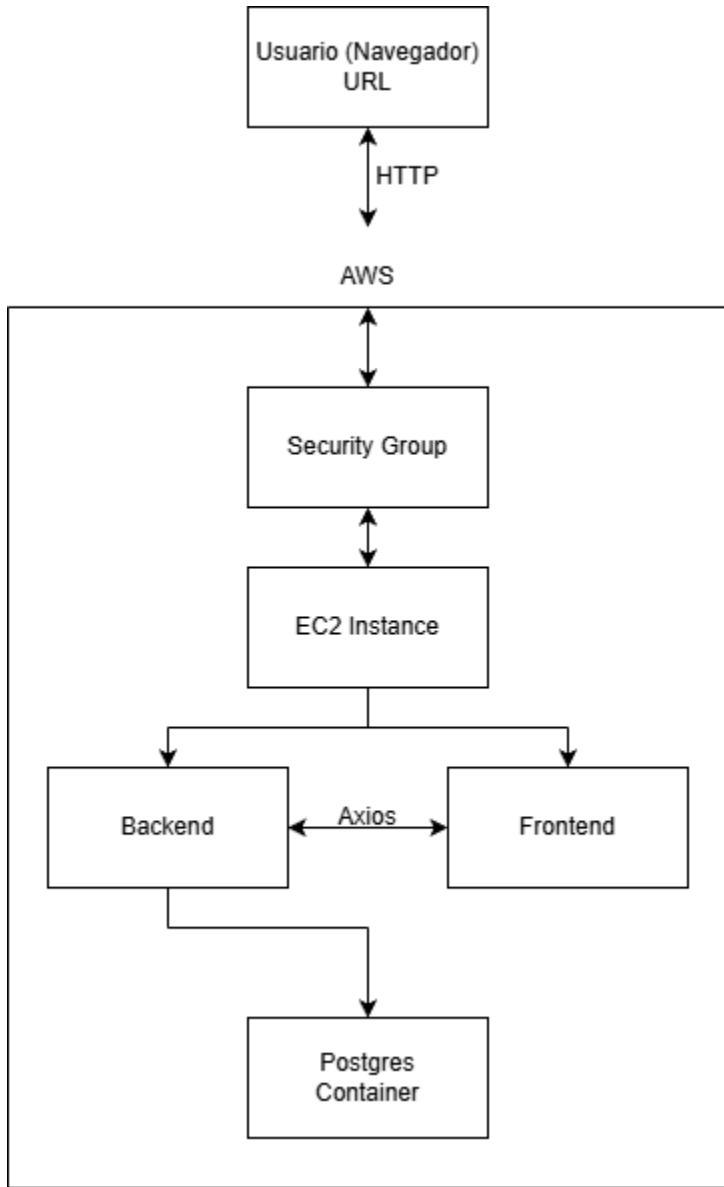
1. Requisitos y alcance.....	3
2. Arquitectura.....	5
3. Dockerfiles.....	6
4. CI/CD.....	7
5. Despliegue en EC2.....	10
6. Seguridad y secretos.....	13
7. Pruebas.....	14
8. Operación y mantenimiento.....	17
9. Conclusiones.....	17

1. Requisitos y alcance

Característica	Estado
Frontend SPA con CRUD completo	Implementado
Backend API REST con lógica de negocio y endpoints para la SPA	Implementado
Base de datos relacional con persistencia	Implementado
Cada componente en contenedor Docker independiente	Implementado
Dockerfiles reproducibles y buenas prácticas	Implementado
docker-compose.yml	No implementado
Despliegue final de los contenedores en una instancia EC2	Implementado
Pipeline CI/CD en GitHub Action	Implementado
Gestión de secretos mediante GitHub Secrets	Implementado
Security Groups configurados correctamente	Implementado

README con instrucciones de ejecución local y URL pública de la aplicación	Implementado
Monitoreo	Implementado
Auto-restart	Implementado
Rollback automatizado	No implementado
TLS/HTTPS	No implementado
Escalado simple	No implementado

2. Arquitectura



El flujo que seguimos es primero, cuando un usuario cualquiera accede desde cualquier navegador a la URL "<http://3.14.88.250/app/mapa>" .

Al pasar eso el tráfico entra a AWS a través del único Security Group previamente configurado con los puertos autorizados. Ya desde dentro de la Instance EC2 se ejecutan tres contenedores existentes con docker-compose, que serían los de Frontend, Backend y Postgres.

El Frontend consume el Backend mediante peticiones Axios. Mientras que el Backend se conecta al contenedor de PostgreSQL usando la URL interna que se encuentra en el .env.

Ventajas de usar esta arquitectura:

- Todo en una sola instancia EC2 lo cual genera un costo mínimo.
- Base de datos protegida, se tiene protegido el puerto para que no esté expuesto.
- Despliegue automatizado con GitHub Actions + SSH.
- Reinicio automático de contenedores (restart: unless-stopped).

3. Dockerfiles

Backend – Dockerfile

```
1   FROM node:18-alpine
2
3   WORKDIR /app
4
5   COPY package*.json ./
6
7   RUN npm ci --omit=dev
8
9   COPY . .
10
11  EXPOSE 3000
12
13  CMD ["npm", "start"]
```

Se logró implementar algunas optimizaciones.

- Primeramente se tiene una imagen base alpine.
- En la línea 7 se tiene “--omit=dev” eso hace que no se instale devDependencies como ser Jest, TypeScript, etc.
- Se tiene capas separadas de dependencias el Docker cachea npm ci cuando solo cambian archivos de código.
- Lo que genera una imagen que es ligera y segura.

Frontend – Dockerfile

```
1      # Build stage
2      FROM node:20-alpine AS build
3      WORKDIR /app
4      COPY package*.json ./
5      RUN npm ci
6      COPY . .
7
8      ARG VITE_API_URL
9      ENV VITE_API_URL=$VITE_API_URL
10
11     RUN npm run build
12
13     # Production stage - Usando Apache (httpd)
14     FROM httpd:alpine
15
16     # Copiar los archivos construidos al directorio de Apache
17     COPY --from=build /app/dist /usr/local/apache2/htdocs/
18
19     # Habilitar mod_rewrite para que funcionen las rutas de React
20     RUN sed -i '/LoadModule rewrite_module/s/^#//g' /usr/local/apache2/conf/httpd.conf
21
22     # Configurar FallbackResource para SPA (La forma más simple en Apache moderno)
23     # Esto le dice a Apache: "Si no encuentras el archivo, sirve index.html"
24     RUN echo "FallbackResource /index.html" >> /usr/local/apache2/conf/httpd.conf
25
26     EXPOSE 80
```

Se logró implementar algunas optimizaciones:

- Multistage build, es decir que la imagen final NO contiene Node.js ni node_modules.
- Usamos variables de entorno VITE_API_URL injectada en build time, lo que significa que el frontend sabe dónde está el backend sin hardcodear.
- FallbackResource /index.html se tiene un soporte perfecto para rutas de React Router (SPA)
- Máximo aprovechamiento de caché Docker: si solo cambian componentes lo que se hará sera solo se rebuilla la capa final no todo.

4. CI/CD

Backend CI/CD – .github/workflows/ci-cd.yml

```
1  name: Backend CI/CD
2
3  on:
4    push:
5      branches: [ main ]
6      workflow_dispatch:
7
8  jobs:
9    deploy:
10      name: Build & Deploy
11      runs-on: ubuntu-latest
12      steps:
13        - name: Checkout code
14          uses: actions/checkout@v4
15
16        - name: Login to Docker Hub
17          uses: docker/login-action@v3
18          with:
19            username: ${{ secrets.DOCKER_USERNAME }}
20            password: ${{ secrets.DOCKER_PASSWORD }}
21
22        - name: Build and Push Docker Image
23          uses: docker/build-push-action@v5
24          with:
25            context: .
26            push: true
27            tags: ${{ secrets.DOCKER_USERNAME }}/exchange-backend:latest
28
29        - name: Deploy to EC2
30          uses: appleboy/ssh-action@master
31          with:
32            host: ${{ secrets.EC2_HOST }}
33            username: ${{ secrets.EC2_USER }}
34            key: ${{ secrets.EC2_SSH_KEY }}
35            script: |
36              docker pull ${{ secrets.DOCKER_USERNAME }}/exchange-backend:latest
37
38              docker stop exchange-backend || true
39              docker rm exchange-backend || true
40
41            # Asegurar red
42            docker network create exchange-network || true
43
44            # Crear archivo .env con el contenido del secreto BACKEND_ENV
45            echo "${{ secrets.BACKEND_ENV }}" > .env
46
47            docker run -d \
48              --name exchange-backend \
49              --network exchange-network \
50              --restart always \
51              -p 3000:3000 \
52              --env-file .env \
53              ${{ secrets.DOCKER_USERNAME }}/exchange-backend:latest
54
55            # Limpiar archivo .env por seguridad
56            rm .env
57
58            # --- VERIFICACIÓN DE DESPLIEGUE ---
59            echo "⏳ Esperando 10 segundos para verificar arranque..."
60            sleep 10
61            echo "💡 Logs de arranque:"
62            docker logs exchange-backend
63            if [ "$(docker inspect -f '{{.State.Running}}' exchange-backend)" = "true" ]; then
64              echo "✅ El backend está corriendo correctamente."
65            else
66              echo "❗ El contenedor se detuvo inesperadamente."
67              exit 1
68            fi
```

En nuestro archivo ci-cd.yml del backend tenemos lo siguiente:

- Construye y sube la imagen a Docker Hub (tag latest)
- SSH a EC2 lo que hace un pull de la nueva imagen
- Crea archivo .env temporal con secretos (DATABASE_URL, JWT_SECRET, etc.) usando el secreto BACKEND_ENV
- Ejecuta el contenedor con --restart always y en la red exchange-network
- Borra el .env por seguridad
- Verifica que el contenedor esté corriendo

Frontend CI/CD – .github/workflows/ci-cd.yml

```
1  name: Frontend CI/CD
2
3  on:
4    push:
5      branches: [ main ]
6      workflow_dispatch:
7
8  jobs:
9    deploy:
10      name: Build & Deploy
11      runs-on: ubuntu-latest
12      steps:
13        - name: Checkout code
14          uses: actions/checkout@v4
15
16        - name: Setup Node & Test
17          uses: actions/setup-node@v4
18          with:
19            node-version: '20'
20            cache: 'npm'
21
22        - run: |
23          npm ci
24          npm run lint
25
26        - name: Login to Docker Hub
27          uses: docker/login-action@v3
28          with:
29            username: ${{ secrets.DOCKER_USERNAME }}
30            password: ${{ secrets.DOCKER_PASSWORD }}
31
32        - name: Build and Push Docker Image
33          uses: docker/build-push-action@v5
34          with:
35            context: .
36            push: true
37            tags: ${{ secrets.DOCKER_USERNAME }}/exchange-frontend:latest
38            # Pasamos la variable de entorno como argumento de construcción (build-arg)
39            build-args:
40              VITE_APT_URL=${{ secrets.VITE_APT_URL }}
41
42        - name: Deploy to EC2
43          uses: appleboy/ssh-action@master
44          with:
45            host: ${{ secrets.EC2_HOST }}
46            username: ${{ secrets.EC2_USER }}
47            key: ${{ secrets.EC2_SSH_KEY }}
48            script: |
49              # Descargar la nueva imagen
50              docker pull ${{ secrets.DOCKER_USERNAME }}/exchange-frontend:latest
51
52              ## --- LIMPIEZA DE PUERTO 80 ---
53              # Apache se reinicia solo si lo matas con kill. Hay que detener el servicio.
54              sudo systemctl stop apache2 || true
55              sudo systemctl disable apache2 || true
56              # Por si acaso Nginx también está instalado como servicio
57              sudo systemctl stop nginx || true
58              sudo systemctl disable nginx || true
59
60              # Detener y eliminar el contenedor anterior (si existe)
61              docker stop exchange-frontend || true
62              docker rm exchange-frontend || true
63
64              # Correr el nuevo contenedor
65              docker run -d \
66                --name exchange-frontend \
67                --restart always \
68                -p 80:80 \
69                ${{ secrets.DOCKER_USERNAME }}/exchange-frontend:latest
```

En nuestro archivo ci-cd.yml del frontend tenemos lo siguiente:

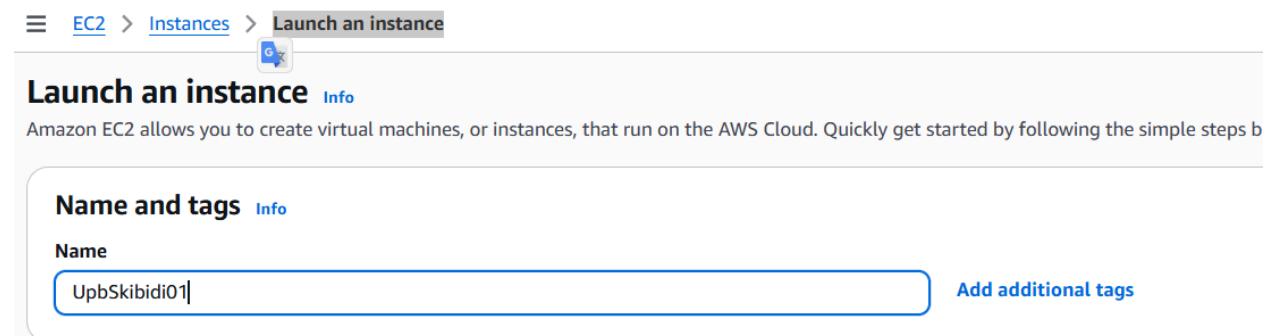
- Ejecuta npm ci y lint
- Inyecta VITE_API_URL como build-arg ya que el frontend sabe dónde está el backend sin hardcodear
- Construye y sube imagen a Docker Hub
- En EC2: detiene servicios web que puedan estar ocupando el puerto 80
- Despliega el nuevo contenedor en el puerto 80 con --restart always

Secrets utilizados:

- DOCKER_USERNAME, DOCKER_PASSWORD
- EC2_HOST, EC2_USER, EC2_SSH_KEY
- VITE_API_URL es la URL pública del backend
- BACKEND_ENV tiene el contenido completo del .env del backend

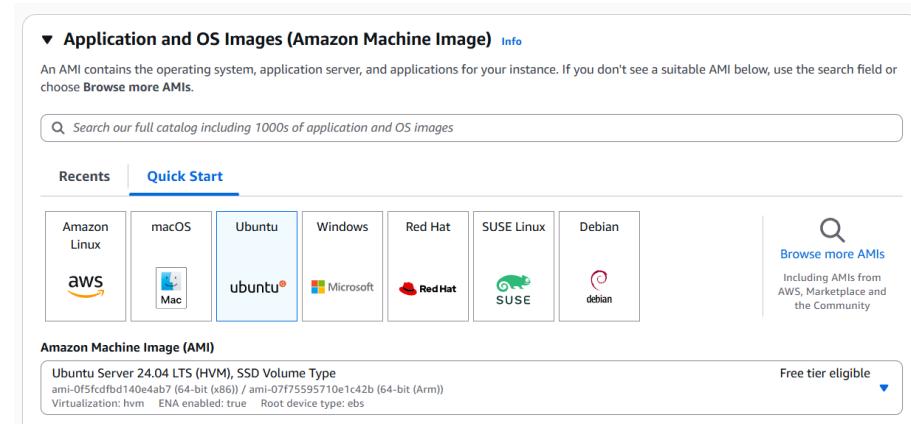
5. Despliegue en EC2

La creación de nuestro EC2 fue primero entrar a nuestra cuenta de AWS, luego nos vamos al apartado de EC2 y ponemos a Launch an instance y al entra ahí debemos colocar un nombre para nuestro EC2, en nuestro caso le pusimos “UpbSkibidi01” pero podemos poner el nombre que mejor nos convenga.



The screenshot shows the 'Launch an instance' wizard in the AWS Management Console. The first step, 'Name and tags', is completed with the name 'UpbSkibidi01' entered in the 'Name' field. The 'Add additional tags' button is visible. The next steps, 'Configure volume', 'Configure networking', and 'Review and launch', are shown as incomplete.

Después seleccionamos la imagen que vamos a usar y en nuestro caso seleccionamos Ubuntu y verificamos que sea la versión Free tier



The screenshot shows the 'Application and OS Images (Amazon Machine Image)' selection screen. The 'ubuntu' AMI is selected, highlighted with a blue border. Other options like Amazon Linux, macOS, Windows, Red Hat, SUSE Linux, and Debian are also listed. A search bar at the top right says 'Search our full catalog including 1000s of application and OS images'. A sidebar on the right says 'Browse more AMIs' and 'Including AMIs from AWS, Marketplace and the Community'. At the bottom, it shows 'Ubuntu Server 24.04 LTS (HVM), SSD Volume Type' and 'Free tier eligible'.

Luego en la instancia le podemos t3.small y vemos que sea Free tier

The screenshot shows the 'Instance type' section of the AWS instance creation wizard. It highlights the 't3.small' option, which is labeled as 'Free tier eligible'. Below the instance type, there is detailed pricing information for various operating systems. A note at the bottom states 'Additional costs apply for AMIs with pre-installed software'.

En la Key pair podemos usar alguna ya creada o podemos crear una nueva, pero necesitamos seleccionar que sea .pem para poder usarla luego.

The screenshot shows the 'Create key pair' dialog box. Under 'Key pair name', there is a text input field with the placeholder 'Enter key pair name'. Under 'Key pair type', the 'RSA' option is selected. Under 'Private key file format', the '.pem' option is selected. A note in the dialog says: '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.' At the bottom right are 'Cancel' and 'Create key pair' buttons.

En network settings seleccionamos la opción para acceder con HTTP que es la última

The screenshot shows the 'Network settings' section of the AWS instance creation wizard. Under 'Auto-assign public IP', the 'Enable' option is selected. In the 'Firewall (security groups)' section, the 'Create security group' button is selected. A note below says: 'We'll create a new security group called "launch-wizard-2" with the following rules:' There are three checkboxes for traffic rules: 'Allow SSH traffic from Anywhere', 'Allow HTTPS traffic from the internet', and 'Allow HTTP traffic from the internet'. A note at the bottom of the rules section says: 'Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.'

Y ya por ultimo le ponemos Launch Instance para así poder ver nuestra instancia y ver que este corriendo con las características que dijimos.

The screenshot shows the AWS Lambda function configuration page for 'UpbSkibidi01'. At the top, it displays the function name, ARN, state (Running), memory (t3.small), and a success rate of 3/3 checks passed. Below this, there's a summary table with various details about the instance:

Instance ID	Public IPv4 address	Private IPv4 addresses
i-0cb4c2be6a096f345	3.14.88.250 open address ↗	172.31.37.207
IPv6 address	Instance state	Public DNS
-	Running	ec2-3-14-88-250.us-east-2.compute.amazonaws.com open address ↗
Hostname type	Private IP DNS name (IPv4 only)	Elastic IP addresses
IP name: ip-172-31-37-207.us-east-2.compute.internal	ip-172-31-37-207.us-east-2.compute.internal	-
Answer private resource DNS name	Instance type	AWS Compute Optimizer finding
IPv4 (A)	t3.small	Opt-in to AWS Compute Optimizer for recommendations.
Auto-assigned IP address	VPC ID	Learn more ↗
3.14.88.250 [Public IP]	vpc-01759d3631f7c6f80 ↗	
IAM Role	Subnet ID	Auto Scaling Group name
-	subnet-067d2cf35a81f5ea ↗	-

Podemos agregar a un Security group y poner ahí los puertos que queremos que escuche

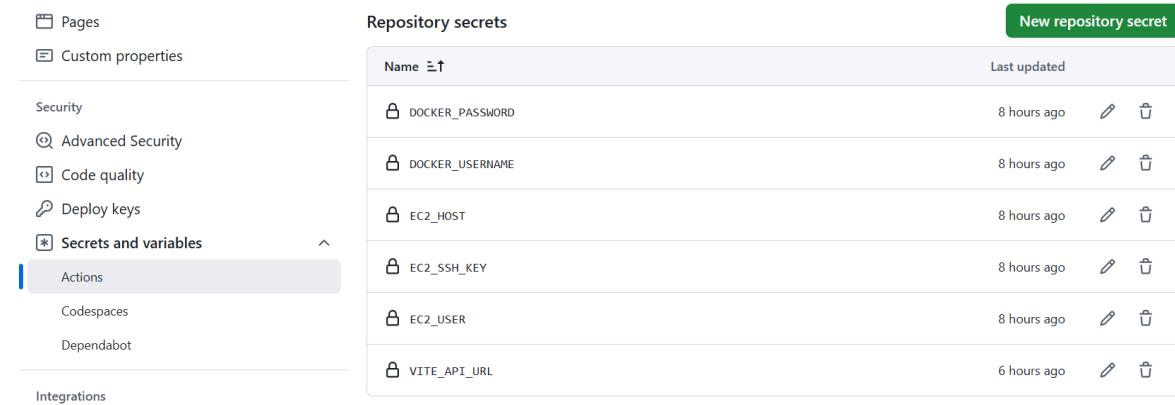
Name	Security group rule ID	Port range	Protocol	Source
-	sgr-06b7d58427f2938f5	5173	TCP	0.0.0.0/0
-	sgr-07571d73d8ad10044	80	TCP	0.0.0.0/0
-	sgr-0a93b4f20e4e2d102	3000	TCP	0.0.0.0/0
-	sgr-0371ff1d5c4a61f6f	22	TCP	0.0.0.0/0
-	sgr-0906d34365de4017a	443	TCP	0.0.0.0/0
-	sgr-0037aeaacea290796	4000	TCP	0.0.0.0/0

▼ Outbound rules

Name	Security group rule ID	Port range	Protocol	Destination
-	sgr-02a0de5ed337be370	All	All	0.0.0.0/0

6. Seguridad y secretos

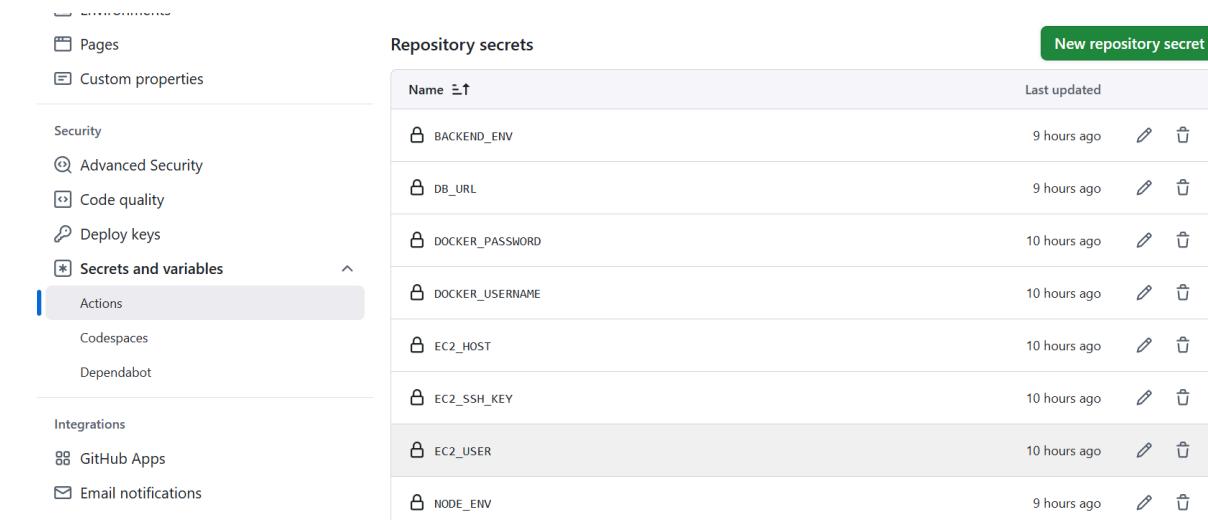
Frontend



The screenshot shows the 'Repository secrets' page for a GitHub repository. On the left, there's a sidebar with 'Actions' selected. The main area lists seven secrets:

Name	Last updated	Actions
DOCKER_PASSWORD	8 hours ago	
DOCKER_USERNAME	8 hours ago	
EC2_HOST	8 hours ago	
EC2_SSH_KEY	8 hours ago	
EC2_USER	8 hours ago	
VITE_API_URL	6 hours ago	

Backend



The screenshot shows the 'Repository secrets' page for a GitHub repository. On the left, there's a sidebar with 'Actions' selected. The main area lists eight secrets:

Name	Last updated	Actions
BACKEND_ENV	9 hours ago	
DB_URL	9 hours ago	
DOCKER_PASSWORD	10 hours ago	
DOCKER_USERNAME	10 hours ago	
EC2_HOST	10 hours ago	
EC2_SSH_KEY	10 hours ago	
EC2_USER	10 hours ago	
NODE_ENV	9 hours ago	

En ambos casos tanto del Frontend como del Backend se crearon secrets para que no esté visible en el código y se tenga mas seguridad, la mayoría de estos se puede encontrar en los archivos Docker en ambos repositorios. En el backend la última secret de `NODE_ENV` se encuentra en el archivo `\src\database\connection.js`. En el frontend la última secret de `VITE_API_URL` se encuentra en el archivo Dockerfile.

Se recomienda migrar a GitHub Container Registry o Amazon ECR en lugar de Docker Hub para mayor privacidad, implementar rotación automática de claves SSH cada 90 días y adoptar AWS Secrets Manager o Parameter Store para credenciales sensibles como `DATABASE_URL` y `JWT_SECRET`. También se sugiere inyectar variables de entorno directamente en el comando `docker run` en lugar de usar un archivo `.env`, añadir escaneos de vulnerabilidades con Trivy o Docker Scout en el pipeline, configurar un usuario non-root en los contenedores, activar Docker Content Trust para firmar imágenes, y usar tags con el hash de commit para facilitar rollbacks y garantizar la integridad de las imágenes, elevando así el nivel de seguridad a estándares empresariales.

7. Pruebas

Pruebas git hub actions

Frontend:

The screenshot shows the GitHub Actions interface for the repository 'certi-devops-team / proyecto2-front'. The left sidebar shows 'Frontend CI/CD' with sections for Management, Caches, Attestations, Runners, Usage metrics, and Performance metrics. The main area displays 'All workflows' with 9 workflow runs. The runs are listed in descending order of start time, showing various 'fix: project' and 'deploy: fix' jobs. Each run includes a status indicator (green for success), the job name, the commit hash, and the date/time.

Workflow Run	Job	Commit Hash	Status	Date
1	fix: project	5d3472f	Success	Dec 9, 6:03 PM GMT-4
2	fix: project	8ebd273	Success	Dec 9, 6:00 PM GMT-4
3	deploy: fix	5452609	Success	Dec 9, 4:41 PM GMT-4
4	deploy: fix	1189bb1	Success	Dec 9, 4:39 PM GMT-4
5	deploy: fix	b0df2f7	Success	Dec 9, 4:32 PM GMT-4
6	deploy: fix	a0d7217	Success	Dec 9, 4:22 PM GMT-4
7	deploy: fix	a582f3c	Success	Dec 9, 4:17 PM GMT-4

Backend:

The screenshot shows the GitHub Actions interface for the repository 'certi-devops-team / proyecto2-back'. The left sidebar shows 'Backend CI/CD' with sections for Management, Caches, Attestations, Runners, Usage metrics, and Performance metrics. The main area displays 'All workflows' with 16 workflow runs. The runs are listed in descending order of start time, showing various 'deploy: test', 'deploy: secrets', and 'deploy: test' jobs. Each run includes a status indicator (green for success), the job name, the commit hash, and the date/time.

Workflow Run	Job	Commit Hash	Status	Date
1	deploy: test	a67dc33	Success	Dec 9, 3:48 PM GMT-4
2	deploy: test	341dd94	Success	Dec 9, 3:47 PM GMT-4
3	deploy: test	56fff54	Success	Dec 9, 3:45 PM GMT-4
4	deploy: secrets	36a4181	Success	Dec 9, 3:41 PM GMT-4
5	deploy: secrets	6e58263	Success	Dec 9, 3:36 PM GMT-4
6	deploy: secrets	53f3c57	Success	Dec 9, 3:34 PM GMT-4
7	deploy: secrets	de3cfe5	Success	Dec 9, 3:21 PM GMT-4

Editar una casa de cambio:

PLEASE SPEED DYLAN 12.0000 12.0000

Ubicación en el mapa



Editar Casa de Cambio

Nombre: CASA EDITADA

Dirección: CALLE EDITADA

Moneda: DYLAN

Compra: 12.0000

Venta: 12.0000

Comisión (%): 2.00

Capital Mínimo: 455.00

Leaflet | © OpenStreetMap

ARIT. , Trinidad

CHAR. dda S/N, Potosí

abierta

Moneda: DYLAN

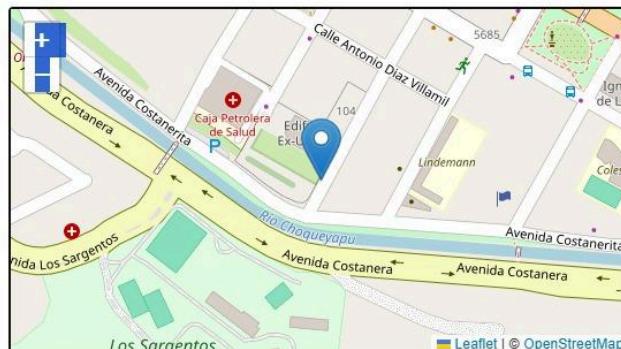
Compra: 12.0000

Venta: 12.0000

Comisión (%): 2.00

Capital Mínimo: 455.00

Leaflet | © OpenStreetMap



CASA EDITADA CALLE EDITADA DYLAN 12.0000 12.0000



Creación de un usuario:

Nombre
Usuario

Apellido
Creado

Email
 usuario@acme.com

Contraseña


Repetir Contraseña


Registrarme como administrador

REGISTRARME

[Ya tienes cuenta? Inicia sesión](#)

Casa de Cambios Bolivia

- Dashboard
- Mapa
- Cotizaciones
- Historial
- Casas Admin
- Perfil**
- Transacciones
- Transacciones Dashboard

Perfil de Usuario

Email: usuario@acme.com
Nombre: Usuario
Apellido: Creado
Rol: admin
Moneda Preferida: USD
Umbral de Alerta: 0.00
Alertas Habilitadas: No

Configuración de Alertas y Moneda

Umbral de Alerta (USD):

Moneda Preferida:

GUARDAR CONFIGURACIÓN

[CERRAR SESIÓN](#)

8. Operación y mantenimiento

Reinicio automático: Todos los contenedores (frontend, backend y base de datos) están configurados con la política --restart always (frontend y backend) y unless-stopped (PostgreSQL). En caso de caída del contenedor o reinicio de la instancia EC2, Docker los levanta automáticamente sin intervención manual.

Logs: Los logs están disponibles en tiempo real mediante los comandos:

```
docker logs -f exchange-frontend
```

```
docker logs -f exchange-backend
```

```
docker logs -f exchange-postgres
```

9. Conclusiones

El proyecto cumplió con la mayoría de los objetivos del segundo parcial: se desarrolló una aplicación full-stack completa para la gestión de casas de cambio en Bolivia, con CRUD completo, autenticación segura y funcionalidades avanzadas (mapa interactivo, gráficos y alertas). Se implementó contenerización profesional con Dockerfiles multistage y buenas prácticas, despliegue en una instancia EC2 de AWS, y un pipeline CI/CD totalmente automatizado mediante GitHub Actions que construye, publica y despliega imágenes Docker en producción con cada push a main. Aprendimos a integrar de forma real todo el ciclo de vida de una aplicación moderna (desarrollo → build → test → deploy → monitoreo), a manejar secretos de forma segura, a optimizar imágenes Docker para producción y a automatizar despliegues sin downtime. Aunque nos quedamos por hacer mejoras como ser TLS/HTTPS, rollback automatizado y escalado horizontal, consideramos que la solución actual es robusta, segura, de bajo costo y completamente operativa en producción, demostrando un dominio sólido de las herramientas DevOps exigidas en el curso.