De Código a Producción

IaC con Terraform, Contenedores en ECR, Despliegue Automatizado en AWS EC2

via GitHub Actions y Exposición Segura via Cloudflare Tunnels

---

Documentacion Técnica del Proyecto

Omar Garcia

omargarcia.xyz

Table of Contents

1. Introducción

Este documento describe de forma detallada la arquitectura, decisiones técnicas y flujos de trabajo implementados en el proyecto de portfolio DevOps. El objetivo es que cualquier persona técnica pueda entender que se construyó, como funciona cada componente y por qué se tomaron las decisiones de diseño que se tomaron.

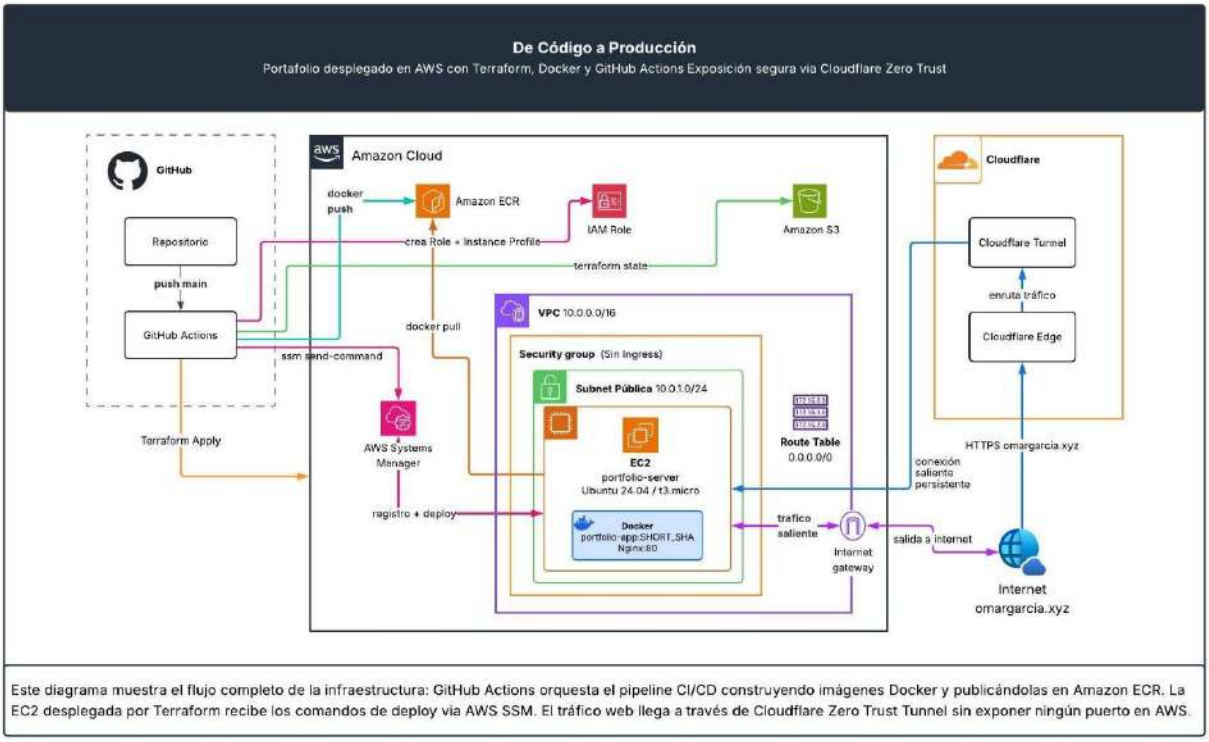
El proyecto implementa una infraestructura completa en AWS siguiendo las mejores prácticas de la industria: Infraestructura como Código (IaC), contenedores Docker, pipelines CI/CD automatizados y exposición segura a internet sin abrir puertos, todo de forma completamente automatizada desde el primer commit hasta el portfolio visible en producción en omargarcia.xyz.

2. Visión General de la Arquitectura

La arquitectura se divide en cuatro capas principales que trabajan en conjunto:

- Capa IaC: Terraform aprovisiona toda la infraestructura en AWS de forma declarativa y reproducible.
- Capa CI/CD: GitHub Actions automatiza el build, push y deploy en cada cambio de código.
- Capa de Aplicación: Docker empaqueta la aplicación y Nginx la sirve dentro de la EC2.
- Capa de Seguridad: Cloudflare Zero Trust Tunnel expone el portfolio sin abrir puertos en AWS.

Flujo de extremo a extremo:




```
Developer (git push a main)
    |
    v
GitHub Actions
    |-- infra.yml  --> terraform apply --> AWS (VPC, EC2, ECR, IAM, S3)
    |-- ci-cd.yml  --> docker build   --> ECR
                   --> SSM deploy     --> EC2 --> Docker --> Nginx

Usuario --> Cloudflare Edge --> Cloudflare Tunnel --> EC2:80 --> Portfolio
```

3. Infraestructura como Código (Terraform)

Terraform define toda la infraestructura en archivos .tf versionados junto al código. Esto garantiza que la infraestructura sea reproducible, auditable y fácil de modificar. El estado se almacena en Amazon S3 (bucket: alexis-cv-terraform-state) para que tanto el equipo local como GitHub Actions lean el mismo estado.

3.1 Estructura de Modulos

Modulo	Recursos que crea	Dependencias
network	VPC, Subnet publica, Internet Gateway, Route Table	Ninguna
security	Security Group, IAM Role, Instance Profile	vpc_id de network
compute	EC2, ECR Repository	vpc_id, subnet_id, sg_id, iam_profile

3.2 Modulo Network

Crea la red base sobre la cual vive toda la infraestructura:

- VPC (10.0.0.0/16): Red privada virtual aislada con 65,536 IPs disponibles.
- Subnet Publica (10.0.1.0/24): Subred con 256 IPs donde vive la EC2.
- Internet Gateway: Puerta de salida hacia internet. Sin este recurso la EC2 no podría conectarse con Cloudflare, ECR ni SSM.
- Route Table: Define que todo el tráfico externo (0.0.0.0/0) salga por el Internet Gateway.

3.3 Modulo Security

Security Group (Firewall)

El Security Group no tiene ninguna regla de entrada. Ningún puerto está abierto al exterior porque:

- El tráfico web llega via Cloudflare Tunnel (conexión saliente, no entrante).
- El deploy se realiza via AWS SSM (no requiere puerto 22 ni SSH).
- Solo existe una regla de salida para que la EC2 pueda conectarse a Cloudflare, ECR y SSM.

IAM Role y Instance Profile

El IAM Role define la identidad de la EC2 dentro de AWS con unicamente dos politicas:

Politica	Permisos	Para que se usa
AmazonEC2ContainerRegistryReadOnly	Solo lectura en ECR	Descargar imágenes Docker desde ECR
AmazonSSMManagedInstanceCore	Comunicacion con SSM	Recibir comandos de deploy sin SSH

3.4 Modulo Compute

Amazon ECR

Repositorio privado de imágenes Docker en AWS. Cada imagen se almacena con el tag proyecto-cv-{SHA} donde SHA son los primeros 7 caracteres del commit. Esto permite trazabilidad completa y rollback a cualquier versión anterior.

EC2

Instancia Ubuntu 24.04 tipo t3.micro. La AMI se busca dinámicamente via data source de Terraform, evitando hardcodear el ID que varía por región. Al arrancar, el User Data ejecuta automaticamente:

1. Instalación de Docker y AWS CLI v2.
2. Instalación y activación del SSM Agent.
3. Instalación de cloudflared (agente del tunel de Cloudflare).
4. Configuración del ingress del tunel para enrutar omargarcia.xyz a localhost:80.

4. Contenedores Docker

La aplicación esta empaquetada en una imagen Docker basada en nginx:stable-alpine. Alpine Linux es una distribución minimalista de aproximadamente 5MB que reduce la superficie de ataque y el tamaño de la imagen final.

El Dockerfile implementa las siguientes buenas prácticas:

- Imagen base oficial con version fija (stable) para evitar cambios inesperados.
- COPY con --chown=nginx:nginx para que los archivos pertenezcan al usuario nginx, no a root.
- HEALTHCHECK cada 30 segundos via curl para que Docker detecte si el contenedor deja de responder.
- EXPOSE 80 documenta el puerto que usa el contenedor.

Las imágenes en ECR se nombran con el formato:

```
<account_id>.dkr.ecr.us-east-1.amazonaws.com/portfolio-devops-aws:proyecto-cv-a1b2c3d
```

Esto permite hacer rollback a cualquier versión anterior simplemente cambiando el tag en el comando docker run.

5. Pipeline CI/CD (GitHub Actions)

El ciclo de vida del desarrollo está completamente automatizado mediante dos workflows independientes, cada uno con responsabilidad clara.

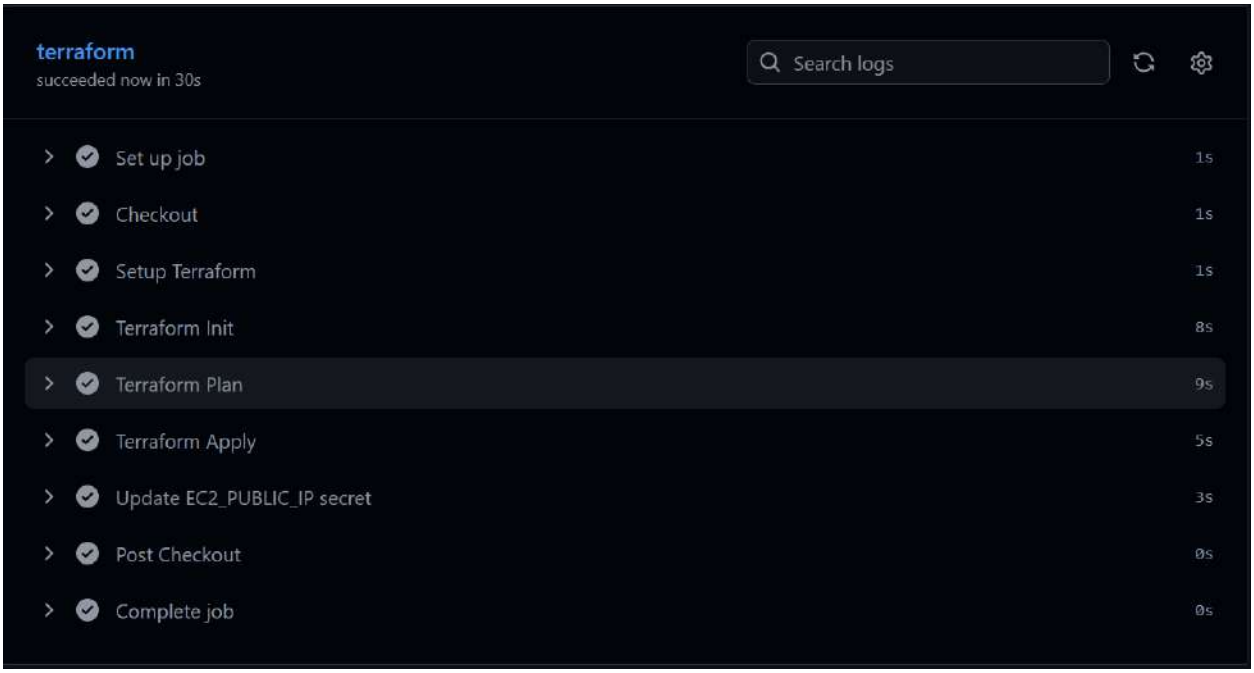
5.1 Workflow de Infraestructura (infra.yml)

Se dispara únicamente cuando hay cambios en la carpeta infra/:

Evento	Comportamiento	Proposito
Pull Request a main	terraform plan (solo lectura)	Revisar que cambios habrá antes de aprobar el PR
Push a main	terraform plan + terraform apply	Desplegar los cambios de infraestructura automáticamente

Las variables sensibles se pasan como TF_VAR_* para que Terraform las lea automáticamente sin hardcodear valores:

```
TF_VAR_cloudflare_token  -->  secrets.cloudflare_token  (sensitive=true)
TF_VAR_tunnel_id         -->  secrets.tunnel_id
TF_VAR_domain            -->  secrets.domain
```

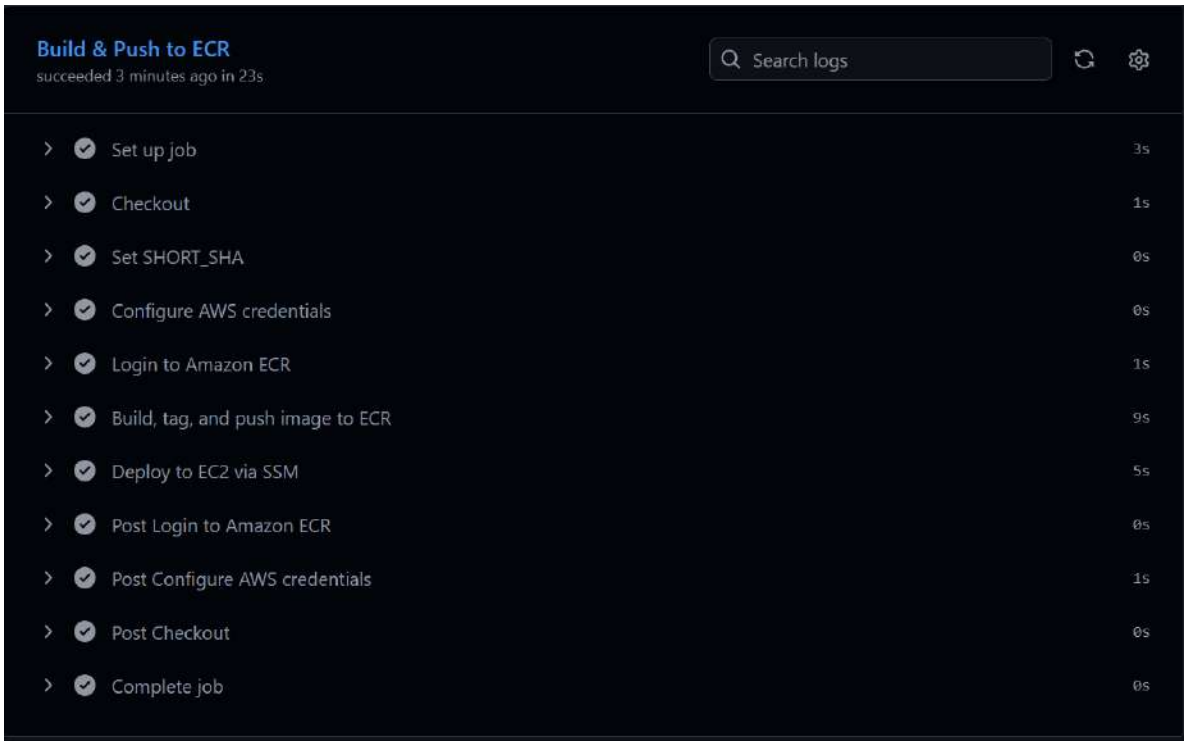


Pipeline infra.yml

5.2 Workflow de Aplicación (ci-cd.yml)

Se dispara en cada push a main, excepto cuando los cambios son solo en infra/. Ejecuta los siguientes pasos:

5. Checkout: descarga el código del repositorio.
6. Set SHORT_SHA: extrae los primeros 7 caracteres del SHA del commit para usar como tag de la imagen.
7. Configure AWS credentials: configura el acceso a AWS usando los secrets del repositorio.
8. Login to Amazon ECR: se autentica en el registro privado de imágenes.
9. Build & Push: construye la imagen Docker y la sube a ECR con el tag proyecto-cv-{SHORT_SHA}.
10. Deploy via SSM: obtiene el Instance ID de la EC2 por su tag y envía los comandos de deploy a través de AWS SSM.



Pipeline CI-CD.yml

El proceso de deploy que SSM ejecuta en la EC2:

```
1. docker login en ECR para autenticarse
2. docker stop portfolio-app   (detener contenedor anterior)
3. docker rm portfolio-app     (eliminar contenedor anterior)
4. docker pull <imagen>:<tag>  (descargar nueva imagen desde ECR)
5. docker run -d -p 80:80     (levantar nuevo contenedor)
```

Gracias a SSM, todo este proceso ocurre sin abrir el puerto 22 ni usar llaves SSH. GitHub Actions le dice a SSM que ejecute los comandos, SSM los envía al agente instalado en la EC2, y la EC2 los ejecuta y devuelve el resultado.

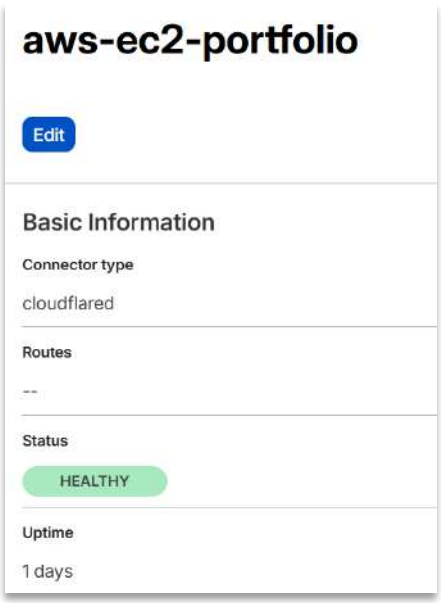
6. Exposición Segura - Cloudflare Zero Trust Tunnel

Esta es la parte más importante desde el punto de vista de seguridad. El portfolio se expone a internet sin abrir ningún puerto en AWS.

6.1 Como funciona el tunel

A diferencia de un servidor web tradicional donde el servidor espera conexiones entrantes (abriendo puertos), con Cloudflare Tunnel la EC2 actúa como cliente: es ella quien inicia la conexión hacia Cloudflare.

11. Al arrancar, cloudflared establece una conexión saliente persistente hacia la red de Cloudflare por el puerto 443.
12. Cloudflare registra esta conexión y sabe que la EC2 está disponible.
13. Cuando un usuario visita omargarcia.xyz, el DNS apunta a Cloudflare (no a la EC2).
14. Cloudflare Edge recibe el tráfico, aplica SSL y protección DDoS.
15. Cloudflare inyecta el tráfico por el tunel ya establecido hacia cloudflared en la EC2.
16. cloudflared reenvía el tráfico a localhost:80 donde corre el contenedor Docker con Nginx.



Tunnel en CloudFlared

6.2 Ventajas de seguridad

Aspecto	Servidor tradicional	Con Cloudflare Tunnel
Puertos abiertos	80, 443 expuestos	Ninguno
IP del servidor	Visible publicamente	Oculta detras de Cloudflare
SSL	Configuracion manual	Automatico con Let's Encrypt
Proteccion DDoS	Opcional (costo extra)	Incluida sin costo adicional
Quien inicia conexion	Cliente externo al servidor	La propia EC2 hacia Cloudflare

7. Decisiones de Seguridad

El proyecto aplica el principio de mínimo privilegio en cada capa:

Practica	Implementacion	Beneficio
Sin puertos abiertos	Security Group sin reglas ingress	Superficie de ataque cero
Sin SSH	Deploy via AWS SSM Session Manager	No hay llaves privadas que gestionar
Sin credenciales en servidor	IAM Role con Instance Profile	No hay AWS keys hardcodeadas en la EC2
Secrets cifrados	GitHub Secrets + Terraform sensitive	Nunca aparecen en los logs
IAM de minimo privilegio	Policy personalizada para GitHub Actions	Sin AdministratorAccess
Imagenes escaneadas	scan_on_push=true en ECR	Deteccion automatica de vulnerabilidades

8. Secrets y Variables de GitHub Actions

8.1 Secrets

Secret	Descripcion
AWS_ACCESS_KEY_ID	Credenciales del usuario IAM github-actions para autenticarse en AWS
AWS_SECRET_ACCESS_KEY	Clave secreta del usuario IAM de GitHub Actions
CLOUDFLARE_TUNNEL_TOKEN	Token de autenticación del 14ctua de Cloudflare
TUNNEL_ID	ID único del 14ctua para configurar el ingress en cloudflared
DOMAIN	Dominio público del portfolio (omargarcia.xyz)
GH_PAT	Personal Access Token para 14ctualizer secrets via GitHub CLI

8.2 Variables

Variable	Descripcion
IMAGE_NAME	Nombre base de la imagen Docker (ej: proyecto-cv). Se combina con SHORT_SHA para formar el tag completo proyecto-cv-a1b2c3d.

9. Flujo Completo End-to-End

9.1 Cambio en la Aplicación

17. El desarrollador hace git push a main con cambios en app/.
18. GitHub Actions dispara ci-cd.yml automaticamente.
19. Se calcula el SHORT_SHA del commit (ej: a1b2c3d).
20. Se autentica en AWS y en Amazon ECR.
21. Se construye la imagen Docker con el tag proyecto-cv-a1b2c3d.
22. La imagen se sube (push) a Amazon ECR.
23. GitHub Actions obtiene el Instance ID de la EC2 via AWS CLI.
24. Se envían los comandos de deploy a la EC2 via AWS SSM.
25. La EC2 descarga (pull) la nueva imagen desde ECR.
26. La EC2 detiene el contenedor anterior y levanta el nuevo.
27. El usuario visita omargarcia.xyz y ve la versión actualizada en segundos.

9.2 Cambio en la Infraestructura

28. El desarrollador hace cambios en archivos .tf dentro de infra/.
29. Se abre un Pull Request a main.
30. GitHub Actions dispara infra.yml y ejecuta terraform plan.
31. El equipo revisa en el PR que cambios se realizaran en AWS.
32. Al aprobar y mergear el PR, terraform apply despliega los cambios automáticamente.
33. El estado actualizado se guarda en el bucket S3.

10. Estructura del Proyecto

Ruta	Descripcion
app/Dockerfile	Definición de la imagen Docker basada en nginx: stable-alpine
app/src/	Código fuente HTML/CSS/JS del portfolio
infra/main.tf	Orquestación de módulos (network, security, compute)
infra/variables.tf	Declaración de variables del root module
infra/outputs.tf	Outputs: URL de ECR e IP de la EC2
infra/provider.tf	Configuración del provider AWS con tags por defecto
infra/backend.tf	Backend remoto S3 para el estado de Terraform
infra/terraform.tfvars	Valores locales para desarrollo (en. Gitignore)
infra/modules/network/	VPC, Subnet, Internet Gateway, Route Table
infra/modules/security/	Security Group, IAM Role, Instance Profile
infra/modules/compute/	EC2, ECR Repository, User Data
.github/workflows/ci-cd.yml	Pipeline de la aplicación (build, push, deploy)
.github/workflows/infra.yml	Pipeline de infraestructura (terraform apply)

11. Stack Tecnologico

Tecnologia	Rol en el proyecto
Terraform >= 1.5.0	Aprovisionamiento de toda la infraestructura (IaC)
AWS EC2 t3.micro Ubuntu 24.04	Servidor donde corre el contenedor Docker
Amazon ECR	Registro privado de imagenes Docker
Amazon S3	Estado remoto de Terraform
AWS IAM	Identidad y permisos de la EC2 y GitHub Actions
AWS SSM Session Manager	Deploy remoto sin SSH ni puertos abiertos
Docker nginx:stable-alpine	Contenedor de la aplicacion
GitHub Actions	CI/CD automatizado (build, push, deploy)
Cloudflare Zero Trust Tunnel	Exposición segura sin puertos abiertos

12. Despliegue Manual

Si se requiere recrear la infraestructura desde cero sin GitHub Actions:

```
# 1. Inicializar Terraform y configurar backend S3
terraform init

# 2. Generar el plan de ejecucion
terraform plan -out=plan.out

# 3. Aplicar la infraestructura
terraform apply plan.out

# 4. Recrear solo la EC2 sin tocar VPC ni ECR
terraform apply -replace="module.compute.aws_instance.app_server"
```

13. Glosario

Termino	Definicion
IaC	Infrastructure as Code. Practica de definir la infraestructura en archivos de codigo versionables.
Terraform	Herramienta de IaC que aprovisiona infraestructura en multiples proveedores cloud de forma declarativa.
VPC	Virtual Private Cloud. Red privada virtual aislada dentro de AWS.
ECR	Elastic Container Registry. Registro privado de imagenes Docker de AWS.
SSM Session Manager	Servicio de AWS que permite ejecutar comandos en EC2 sin SSH ni puertos abiertos.
IAM	Identity and Access Management. Servicio de AWS para gestionar identidades y permisos.
Cloudflare Tunnel	Servicio Zero Trust que expone servicios locales mediante conexion saliente, sin abrir puertos.
SHORT_SHA	Primeros 7 caracteres del hash del commit de Git. Identifica la version exacta desplegada.
CI/CD	Continuous Integration / Continuous Deployment. Automatizacion del ciclo de build y deploy.
Zero Trust	Modelo de seguridad que verifica cada acceso de forma explícita sin asumir confianza en la red.