deploya

DOCUMENTO 3 DE 4 · MANUAL TÉCNICO

Manual técnico

Tecnologías, instalación local y en servidor, configuración, despliegue en AWS con Terraform, modo autoalojado, operación diaria, pruebas y solución de problemas.

01 **Requisitos**
Qué hace y para quién

02 **Diseño**
Cómo está hecho y por qué

03 **Manual técnico**
Cómo se instala y opera

04 **Manual de usuario**
Cómo se usa

INSTITUCIÓN	Facultad de Ingeniería Ingeniería en Informática y Sistemas
CURSO	Ingeniería de Software I · Proyecto final 2026
ENTREGA	Presentación final · 13 de noviembre de 2026
REPOSITORIO	`github.com/derekCmorales/deploya`

EQUIPO

Eddy Poroj
Cuentas, seguridad y notificaciones (M1, M10)

Javier Rodríguez
Suscripciones, pagos y administración (M2, M9)

Eduardo Portillo
Proyectos, observabilidad y design system (M3, M7)

Derek Calderón Morales
Motor de despliegue, arquitectura e infraestructura (M4–M6)

VERSIÓN	FECHA	ESTADO	FUENTE
1.1	13 de noviembre de 2026	Presentación final	docs/sitio/doc/03-deploya-manual-tecnico.html

CONTENIDO

01 **Introducción** 3

02 **Tecnologías utilizadas** 4

2.1 Stack 4

2.2 Estructura del repositorio 5

03 **Instalación local** 6

3.1 Requisitos 6

3.2 Levantar la plataforma con un comando 6

3.3 Cuentas de demostración 6

3.4 Desarrollo fuera de Docker 7

04 **Configuración** 8

4.1 Variables principales 8

4.2 Archivos de configuración 9

05 **Despliegue en AWS** 10

5.1 Infraestructura 10

5.2 Primera puesta en marcha 12

5.3 Landing en S3 y CloudFront 12

5.4 Costo del piloto 12

5.5 Comprobar que quedó 13

5.6 Integración continua y publicación 14

06 **Modo autoalojado** 16

07 **Operación** 19

7.1 Actualizar y respaldar 19

7.2 Recursos de datos: respaldo y restauración 19

7.3 Tareas programadas recomendadas 19

7.4 Bitácoras y diagnóstico 20

08 **Pruebas** 21

09 **Solución de problemas** 22

10 **Generar esta documentación** 23

01

Introducción

Este manual explica **cómo se instala, se configura, se despliega y se opera Deploya**. Está escrito para tres lectores:

- **Quien evalúa o prueba el proyecto** y quiere levantarlo en su computadora con un comando (capítulo 3).
- **Quien opera el servidor** de la nube de Deploya: publicar una versión, respaldar, vigilar (capítulos 5 y 7).
- **Quien instala Deploya en su propio servidor** con el modo autoalojado (capítulo 6).

> NOTA
>
> Todos los comandos se corren desde la raíz del repositorio (`github.com/derekCmorales/deploya`) salvo que se diga otra cosa. Los nombres de archivos y variables son los reales del repositorio.

Tecnologías utilizadas

2.1 Stack

Capa	Tecnología	Versión	Para qué
Lenguaje	TypeScript	5.7	Todo el sistema: API, trabajador y web
Entorno	Node.js	22	Ejecución de la API, el trabajador y la web
Gestor de paquetes	pnpm (workspaces)	9.15	Monorepo `apps/api`, `apps/web`, `e2e`
Backend	NestJS	10.4	Módulos, inyección de dependencias, guards, filtros
ORM y migraciones	Prisma	6.5	Esquema, cliente tipado, migraciones versionadas
Base de datos	PostgreSQL	16	Almacén único de la plataforma
Cola	Redis + BullMQ	7 · 6.3	Cola de despliegues, de operación y de muestreo
Contenedores	Docker Engine + `dockerode`	24+ · 5.0	Construir imágenes y correr las apps de los clientes
Enrutador de borde	Traefik	3.3	Rutas por archivo, HTTPS comodín, redirección
Certificados	Let's Encrypt (DNS-01)	—	Certificado `*.dominio` renovado solo
Correo	nodemailer + proveedor SMTP	6.10	Resend en producción, Mailpit en desarrollo
PDF	PDFKit	0.17	Comprobantes de pago
Frontend	Next.js + React	15 · 19	Panel del cliente y del administrador
Estilos	Tailwind CSS + tokens OKLCH	4.3	Design system v4.1
Componentes	Radix UI, Lucide, Framer Motion	—	Accesibilidad, iconos y movimiento
Pruebas	Jest, `@nestjs/testing`, `node --test`	29 · —	Unitarias de API y de lógica web
CI	GitHub Actions, gitleaks, shellcheck	—	Pruebas, secretos y scripts en cada PR
Infraestructura como código	Terraform + proveedor `hashicorp/aws`	1.9 · 5.80	EC2, red, IAM, snapshots, presupuesto y la landing
Nube	AWS: EC2, EBS, DLM, S3, CloudFront, ACM, SSM, Budgets	—	Piloto en us-east-2; la landing va por CloudFront

Capa	Tecnología	Versión	Para qué
Recursos de datos	PostgreSQL + pgvector, Redis, almacenamiento S3 (`pgsty/silo`)	17 · 7 · —	Bases que el cliente crea con un clic; contenedor con volumen en el nodo
Especificación	OpenSpec	—	Un change por historia: propuesta, diseño, spec y tareas

2.2 Estructura del repositorio

Carpeta	Contenido
`apps/api/`	API NestJS y trabajador (`src/main.ts` y `src/trabajador.ts`), `prisma/` con esquema, migraciones y *seed*
`apps/api/src/modules/`	Los once módulos: `identidad` , `suscripciones` , `proyectos` , `construccion` , `orquestacion` , `datos` , `enrutamiento` , `observabilidad` , `herramientas` , `administracion` , `notificaciones`
`apps/api/src/adapters/`	Adaptadores reales (`reales/`), en memoria (`memoria/`), Prisma (`prisma/`) y *stubs*
`apps/web/`	Next.js: rutas por grupo (`(auth)` , `(billing)` , `(projects)` , `(admin)` , `(cuenta)`), componentes y tokens
`e2e/`	*Smoke* y recorrido de punta a punta
`infra/traefik/`	Configuración estática de Traefik para local y producción
`infra/terraform/`	Servidor en AWS, landing en S3 + CloudFront, snapshots diarios y presupuesto
`scripts/`	`instalar.sh` , `actualizar.sh` , `respaldar.sh` , `desplegar-vps.sh` y sus etapas en `lib/`
`openspec/`	Specs por módulo y changes archivados
`docs/`	Alcance, ADR, contratos, diagramas, design system y esta documentación (`docs/sitio/`)

03

Instalación local

3.1 Requisitos

Para	Necesitas
Levantar todo con Docker (recomendado)	Docker con Compose v2 y 4 GB de RAM libres
Desarrollar fuera de Docker	Además, Node.js 22 y pnpm 9 (`corepack enable`)
Generar esta documentación en PDF	Node.js 22 y Chromium (lo instala Playwright)

3.2 Levantar la plataforma con un comando

```
git clone https://github.com/derekCmorales/deploya.git
cd deploya
cp .env.example .env
# Edita .env y define ADMIN_CLAVE (obligatoria)
docker compose up --build
```

Compose arranca en orden: `postgres` y `redis` (con *healthcheck*), luego `migracion` (aplica las migraciones y el *seed*, y termina), luego `api`, `worker`, `traefik`, `mailpit` y `web`.

Servicio	Dirección
Panel web	`http://localhost:3000`
API (salud)	`http://localhost:3001/health`
Bandeja de Mailpit (correos de verificación y recuperación)	`http://localhost:8025`
Apps desplegadas	`http://<proyecto>.localhost`
Catálogo del design system	`http://localhost:3000/sistema`

3.3 Cuentas de demostración

El *seed* es idempotente: se puede correr las veces que haga falta sin duplicar nada ni reescribir contraseñas.

Cuenta	Rol	Estado para la demo	Contraseña
`admin@deploya.app` (o `ADMIN_CORREO`)	Administrador	Activa	`ADMIN_CLAVE`
`cliente@deploya.app`	Cliente	Activa, plan Sandbox	`CLIENTE_CLAVE` (o `ADMIN_CLAVE` si está vacía)

Cuenta	Rol	Estado para la demo	Contraseña
`vencida@deploya.app`	Cliente	Starter con la suscripción **Vencida**	Igual que `cliente@`
`suspendida@deploya.app`	Cliente	Starter con la suscripción **Suspendida**	Igual que `cliente@`

CONSEJO

Tarjetas de prueba de la pasarela simulada: `4242 4242 4242 4242` aprueba, `4000 0000 0000 0002` rechaza por fondos insuficientes y `4000 0000 0000 3220` tarda 5 segundos. Cualquier fecha futura y cualquier CVC de 3 dígitos.

3.4 Desarrollo fuera de Docker

```
pnpm install
docker compose up postgres redis mailpit     # solo la base, la cola y el correo
set -a; . ./.env; set +a
pnpm --filter @deploya/api exec prisma migrate deploy
pnpm --filter @deploya/api prisma:seed
pnpm dev:api                                 # http://localhost:3001 con recarga
pnpm dev:web                                 # http://localhost:3000 con recarga
```

Con `MOTOR_ADAPTADORES=stub` (valor de `.env.example`) el motor corre **en memoria**: los despliegues avanzan por sus etapas sin Docker, útil para trabajar en la interfaz. Con `MOTOR_ADAPTADORES=docker` se usan BullMQ, git, Docker y Traefik reales, y hace falta el `worker`.

Configuración

Toda la configuración entra por **variables de entorno**. El código no tiene dominios, credenciales ni proveedores escritos: el mismo build corre en local, en AWS y en autoalojado.

4.1 Variables principales

Variable	Ejemplo / valores	Para qué
`DATABASE_URL`	`postgresql://deploya:deploya@localhost:5432/deploya`	Conexión de Prisma
`REDIS_URL`	`redis://localhost:6379`	Colas BullMQ
`PORT` · `URL_WEB` · `NEXT_PUBLIC_API_URL`	`3001` · `http://localhost:3000` · `http://localhost:3001`	Puertos y enlaces entre web y API (CORS, enlaces de correo)
`ADMIN_CORREO` · `ADMIN_CLAVE` · `CLIENTE_CLAVE`	—	Cuentas del *seed*; `ADMIN_CLAVE` es obligatoria
`CORREO_ADAPTADOR`	`smtp` · `consola`	Elige el adaptador de correo (ADR 0001)
`SMTP_HOST` · `SMTP_PORT` · `SMTP_USUARIO` · `SMTP_CLAVE` · `CORREO_REMITENTE`	`localhost` · `1025`	Servidor SMTP (Mailpit en desarrollo, Resend en producción)
`MOTOR_ADAPTADORES`	`docker` · `stub`	Motor real o en memoria
`DOMINIO_BASE` · `ESQUEMA_PUBLICO` · `CERT_RESOLVER`	`localhost` · `http` · vacío	Dónde y cómo se publican las apps; resolvedor vacío = sin TLS
`TRABAJADOR_CONCURRENCIA`	`1`	Construcciones simultáneas en el nodo
`MOTOR_TIEMPO_MAXIMO_CONSTRUCCION_MIN`	`20`	Tope de una construcción; proyectos grandes de Next.js o NestJS lo necesitan
`HOST_WEB` · `URL_LANDING`	`app.<dominio>` · `https://<dominio>`	La web vive en `app.` y quien llega a `/` sin sesión va a la landing (ADR 0011)
`CLAVE_CIFRADO_VARIABLES`	32 bytes exactos	Cifrado AES-256-GCM de las variables de los clientes
`GITHUB_TOKEN` · `GITHUB_API_URL`	opcional	Sube el límite de 60 peticiones por hora de la API pública
`MODO_INSTALACION`	`nube` · `autoalojado`	Elige las estrategias de catálogo, cuotas, registro y primer usuario
`NEXT_PUBLIC_DOMINIO_APPS` · `NEXT_PUBLIC_ESQUEMA_APPS`	`localhost` · `http`	La web arma la URL de cada proyecto

CUIDADO

Si se pierde `CLAVE_CIFRADO_VARIABLES`, las variables de entorno guardadas de **todos** los proyectos quedan ilegibles y sus despliegues fallan. En el servidor solo vive en `.env.vps` (permisos 600), nunca en el repositorio ni en Terraform, y se guarda aparte de los respaldos. También cifra las contraseñas de los recursos de datos.

4.2 Archivos de configuración

Archivo	Uso
`.env.example` → `.env`	Desarrollo local y compose
`.env.vps.example` → `.env.vps`	Servidor de producción: dominio, token DNS, SMTP, secretos. Se copia a mano por Session Manager; **nunca se sube al repositorio**
`docker-compose.yml`	Servicios base para todos los entornos
`docker-compose.vps.yml`	Capa de producción (el nombre «vps» es histórico): HTTPS, certificado comodín, redirección y volúmenes de ACME
`docker-compose.http01.yml`	Variante con HTTP-01 para un servidor sin token DNS
`docker-compose.autoalojado.yml`	Capa del modo autoalojado
`infra/traefik/`	Configuración estática de Traefik; `plataforma.generada.yml` la escribe el script
`infra/terraform/terraform.tfvars`	Dominio, región, tipo de instancia, correos del presupuesto; sin secretos

Despliegue en AWS

El piloto corre en **una EC2 creada con Terraform** (ADR 0009): la misma plataforma de `docker compose` que en local, más HTTPS y el certificado comodín. La landing vive fuera del servidor, en S3 y CloudFront, para que siga arriba aunque la EC2 se reinicie (ADR 0011). Ningún secreto pasa por Terraform.

5.1 Infraestructura

Recurso	Qué es	Archivo
Servidor	EC2 `m7i-flex.large` (2 vCPU, 8 GB) con Ubuntu 24.04 en us-east-2, IMDSv2 de un salto y 4 GB de swap para construir	`main.tf`, `cloud-init.yml`
Red	VPC propia con subred pública, sin NAT; grupo de seguridad con 80 y 443; el 22 solo si se pide un CIDR	`red.tf`
Acceso	Session Manager (SSM) en vez de SSH; el perfil de la instancia solo tiene `AmazonSSMManagedInstanceCore`	`iam.tf`
Disco	EBS gp3 de 80 GB cifrado; guarda la base, las imágenes y los volúmenes de los recursos de datos	`main.tf`
Respaldo	Snapshot diario con Data Lifecycle Manager, se guardan 7	`respaldos.tf`
IP	IP elástica: no cambia aunque se reemplace la instancia	`main.tf`
Landing	Bucket S3 privado, CloudFront con OAC, certificado de ACM en us-east-1 y cabeceras de seguridad	`landing.tf`
Costo	AWS Budgets con tope de 70 USD y avisos por correo	`main.tf`
DNS	Cloudflare en «Solo DNS»: `*.<dominio>` → IP elástica; raíz y `www` → CloudFront	a mano
Correo	Resend con el dominio verificado (SPF, DKIM y DMARC)	`.env.vps`

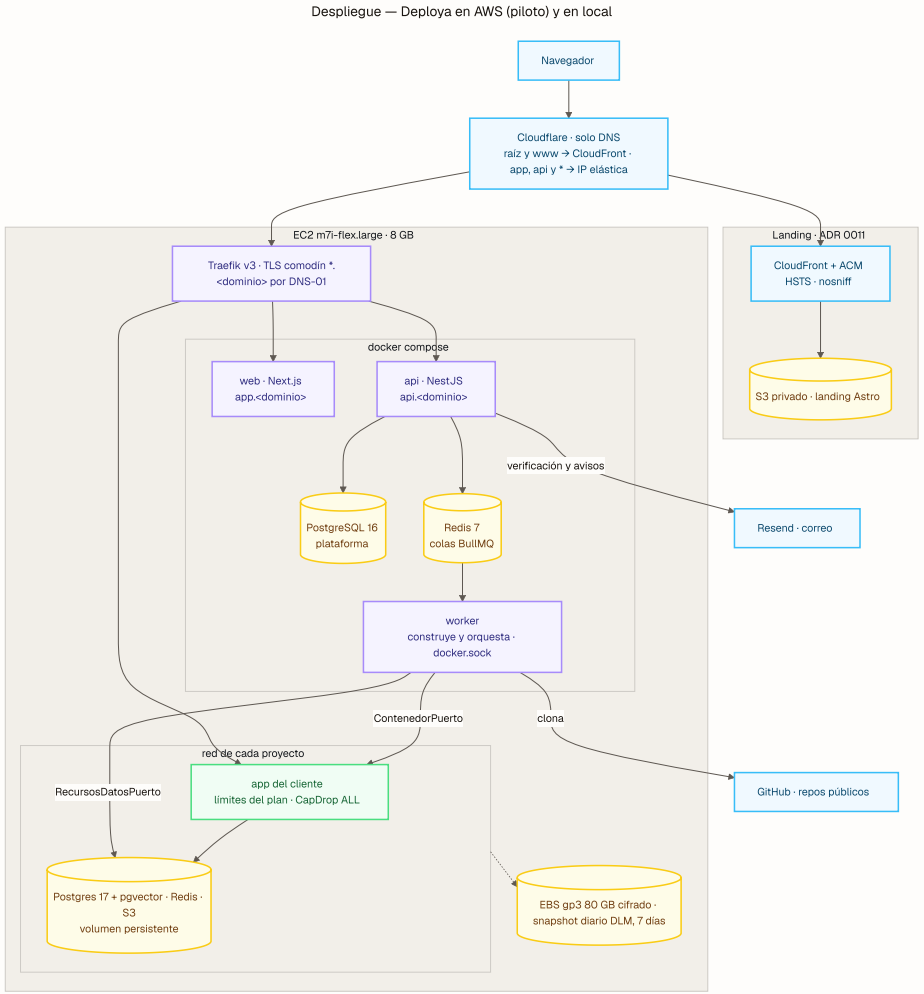


Figura 1. Despliegue — Deploya en AWS (piloto) y en local

5.2 Primera puesta en marcha

Requisitos: Terraform 1.9 o más, AWS CLI v2 con el plugin de Session Manager y credenciales de la cuenta.

```
cd infra/terraform
cp terraform.tfvars.example terraform.tfvars   # dominio y correos del presupuesto
terraform init
terraform plan -out=piloto.tfplan
terraform apply piloto.tfplan
terraform output siguientes_pasos              # imprime los pasos con la IP y el ID reales
```

1. **DNS en Cloudflare**, en «Solo DNS»: lo que imprime `terraform output registros_dns`.
2. **Esperar a cloud-init** (5 a 10 min): `aws ssm start-session --target <id>` y `cloud-init status --wait`. Deja Docker, el usuario `deploya` y el repositorio en `/srv/deploya`.
3. **Copiar** `.env.vps` dentro de la sesión de SSM, sin que pase por Terraform ni por un chat: `sudo -iu deploya`, `cd /srv/deploya && umask 077 && cat > .env.vps` y `chmod 600 .env.vps`.
4. **Publicar**: `RAMA=main scripts/desplegar-vps.sh`.
5. **Guardar** `CLAVE_CIFRADO_VARIABLES` fuera del servidor.

`desplegar-vps.sh` es **idempotente**: hace `git pull --ff-only` de la rama, genera la ruta de la plataforma para Traefik, corre `docker compose --env-file .env.vps -f docker-compose.yml -f docker-compose.vps.yml up -d --build`, repite migraciones y *seed* y espera a que la API responda. Cada publicación posterior es solo ese comando.

Con `HOST_WEB=app.<dominio>` la web queda en `app.<dominio>`, la API en `api.<dominio>`, cada proyecto en `<proyecto>.<dominio>` y el S3 de un proyecto en `<proyecto>-s3.<dominio>`. La raíz es la landing.

5.3 Landing en S3 y CloudFront

La landing es un sitio estático de Astro en el repositorio `deploya_landing`. Se publica con AWS CLI desde ese repositorio:

```
LANDING_BUCKET=$(terraform -chdir=infra/terraform output -raw landing_bucket) \
LANDING_DISTRIBUCION=$(terraform -chdir=infra/terraform output -raw landing_distribucion) \
pnpm deploy:aws     # aws s3 sync + invalidación de CloudFront
```

La primera vez, ACM necesita ver los CNAME de validación en Cloudflare antes de emitir el certificado: `terraform apply -target=aws_acm_certificate.landing`, crear los registros de `registros_dns_validacion_landing` y repetir el `apply`.

5.4 Costo del piloto

Configuración	5 días	Mes
`m7i-flex.large` + 80 GB + IPv4 + snapshots (la del piloto)	≈ 13.5 USD	≈ 82 USD

Configuración	5 días	Mes
`m7i-flex.xlarge` (16 GB), plan B si Nimbo convive con otros clientes	≈ 25 USD	≈ 152 USD
Landing (S3 + CloudFront), ACM, SSM, DLM, Budgets	0	≈ 0

El modelo completo, con la capacidad por nodo y el margen por plan, está en docs/costos.md. `terraform destroy` borra también el disco: antes, respaldo y copia de la clave de cifrado.

5.5 Comprobar que quedó

```
curl -I https://<dominio>                    # la landing, desde CloudFront
curl -I https://app.<dominio>                # la web
curl -I https://api.<dominio>/health         # 200 con el certificado *.<dominio>
curl -I https://algo.<dominio>               # 404 de Traefik, pero con el certificado comodín
curl -I http://hola-deploya.<dominio>        # 301 hacia https
echo | openssl s_client -connect <dominio>:443 -servername x.<dominio> 2>/dev/null \
  | openssl x509 -noout -subject -enddate
```

5.6 Integración continua y publicación

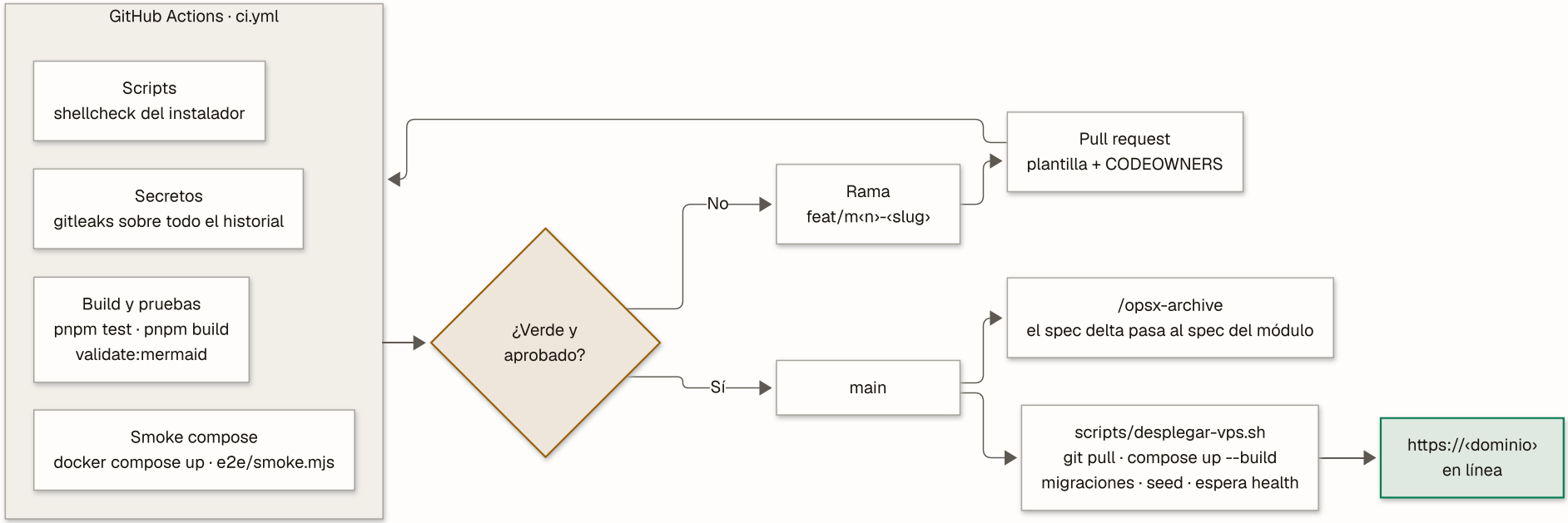


Figura 2. Integración continua y publicación — de la rama al servidor en AWS

Job de CI	Qué comprueba
Build y pruebas	`pnpm install --frozen-lockfile`, `prisma generate`, `pnpm test`, `pnpm build`, `pnpm validate:mermaid`
Secretos (gitleaks)	Ningún secreto en todo el historial del repositorio público
Scripts (shellcheck)	`instalar.sh`, `respaldar.sh`, `actualizar.sh` y sus etapas
Smoke compose	`docker compose up --build`, espera la salud y corre `e2e/smoke.mjs`

06

Modo autoalojado

Deploya es de código abierto (AGPL-3.0) y se instala en un servidor propio. En autoalojado no hay cobro: un único plan «Autoalojado» sin vencimiento, límites por proyecto que fija el administrador y registro abierto, con invitación o cerrado (ADR 0008).

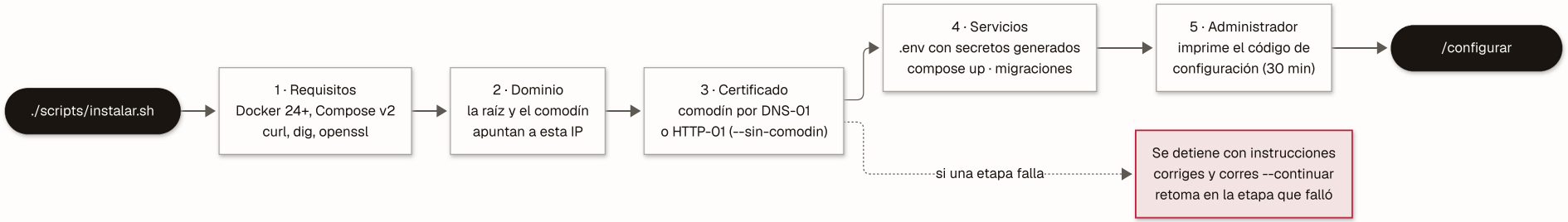


Figura 3. Instalador autoalojado — cinco etapas que comprueban antes de escribir

```
./scripts/instalar.sh                 # instala por etapas
./scripts/instalar.sh --continuar     # retoma después de corregir lo que falló
./scripts/instalar.sh --sin-comodin   # certificado por HTTP-01 si no hay token DNS
./scripts/instalar.sh --codigo-nuevo  # genera otro código de configuración
```

Etapa	Comprueba	Aplica
1 · Requisitos	Docker 24+, Compose v2, `curl`, `dig`, `openssl`	—
2 · Dominio	Que la raíz y un nombre del comodín resuelvan (registro A, no CNAME) a esta máquina	—
3 · Certificado	Token del proveedor DNS o, con `--sin-comodin`, el puerto 80	Estrategia DNS-01 o HTTP-01
4 · Servicios	—	`.env` con permisos 600 y secretos generados (32 bytes), `compose up`, migraciones
5 · Administrador	—	Imprime una sola vez el **código de configuración** (vence en 30 min); solo su hash sha256 va al `.env`

Con el código, se abre `https://<dominio>/configurar`: el asistente pide el código, crea la cuenta de administrador, prueba el SMTP (o sigue sin correo) y elige la política de registro. Lo guardado vive en la tabla `ConfiguracionInstalacion`, no en archivos, y se cambia después en **Admin · Instalación**.

07

Operación

7.1 Actualizar y respaldar

```
./scripts/respaldar.sh                          # base + configuración, sin la clave de cifrado
./scripts/respaldar.sh --restaurar AAAA-MM-DD  # vuelve a un respaldo
./scripts/actualizar.sh                         # respalda, migra y reinicia api, web y worker
./scripts/actualizar.sh --volver                # restaura el respaldo previo a la actualización
```

- `respaldar.sh` guarda la base y la configuración, conserva los **7 más recientes** y no pisa dos respaldos del mismo día. La clave de cifrado **no** entra al respaldo: se guarda aparte.
- `actualizar.sh` **no detiene** los contenedores de las apps de los clientes: solo reinicia la plataforma.
- Además, **Data Lifecycle Manager** toma un snapshot diario del disco completo y guarda 7: es la copia fuera del servidor.

7.2 Recursos de datos: respaldo y restauración

Cada PostgreSQL, Redis o S3 que crea un cliente es un contenedor `deploya-datos-<subdominio>-<tipo>` con un volumen del mismo nombre (ADR 0012). El volumen sobrevive a despliegues, reversiones y reinicios; solo se borra al eliminar el recurso o el proyecto dueño.

```
docker system df -v | grep deploya-datos-              # uso de disco por volumen
C=deploya-datos-nimbo-api-postgres
docker exec $C pg_dump -U deploya -Fc app > app.dump    # respaldo lógico sin detenerlo
docker exec -i $C pg_restore -U deploya -d app --clean < app.dump
```

> **IMPORTANTE**
>
> El disco se **mide y se avisa** (`usoDiscoMb` frente a `discoDatosMb` del plan), no se corta. Restaurar un volumen completo desde un snapshot de DLM está paso a paso en `docs/infra-vps.md §5`.

7.3 Tareas programadas recomendadas

Tarea	Frecuencia	Comando
Respaldo	Diaria (`cron`)	`./scripts/respaldar.sh`
Limpiar imágenes huérfanas	Semanal	`docker image prune -f`
Vigilar la salud	Cada 5 min	`curl -fsS https://api.<dominio>/health`
Ciclo de suscripciones	Diaria, 00:00 UTC	Lo programa la API (BullMQ); no requiere `cron`

Tarea	Frecuencia	Comando
Medición del disco de los recursos de datos	Periódica	Lo programa el trabajador; guarda `usoDiscoMb`
Muestreo de métricas	Cada minuto	Lo programa el trabajador; purga lo de más de 24 h

7.4 Bitácoras y diagnóstico

```
docker compose logs -f api         # peticiones, errores de dominio, fallos de SMTP
docker compose logs -f worker      # pipeline de despliegue, docker build, salud
docker compose logs -f traefik     # rutas y emisión del certificado (ACME)
docker compose ps                  # estado y salud de cada servicio
```

El panel también muestra el estado: **Admin · Infraestructura** (CPU, memoria y disco del nodo, cola y última muestra de cada entorno) y **Admin · Instalación** (salud de Docker, DNS, certificado, disco y correo).

Pruebas

Comando	Qué corre
`pnpm test`	Pruebas de la API (Jest), de la web (`node --test`) y del repositorio
`pnpm test:cov`	Cobertura de la API (hoy 96.8 % de líneas)
`pnpm check`	Lo mismo que CI: pruebas, build y diagramas
`pnpm test:smoke`	*Smoke* contra una plataforma levantada
`pnpm diagramas:sync`	Copia cada `.mmd` a los documentos que lo embeben

Las pruebas unitarias no usan Docker, red, base ni reloj reales: los servicios reciben dobles de sus puertos y un `Reloj` falso. Cada escenario de OpenSpec tiene su prueba con el mismo nombre.

Solución de problemas

Síntoma	Causa probable	Qué hacer
`docker compose up` termina sin API	Falta `ADMIN_CLAVE` en `.env`	Defínela y vuelve a levantar
Una base de datos no se ve desde la app	El proyecto no se ha redesplegado desde que se creó o compartió el recurso	Las variables se inyectan en el siguiente despliegue: redespliega
La construcción se corta a los 20 minutos	Proyecto grande y tope por defecto	Sube `MOTOR_TIEMPO_MAXIMO_CONSTRUCCION_MIN` en `.env.vps`
La web llama a `localhost:3001` en el servidor	La imagen de la web se construyó sin los `build.args`	`docker compose … up -d --build web` con `.env.vps`
Traefik sirve su certificado por defecto	Error de ACME: token sin permiso, DNS sin propagar o límite de Let's Encrypt	`docker compose logs traefik`; revisa el token y los registros A
`certificado: null` en la vista de despliegue	`acme.json` vacío (aún no se emite) o la API sin el volumen `traefik_acme`	Espera la emisión; revisa el montaje de solo lectura
502 al abrir un proyecto	El contenedor no está en la red del proyecto	Revisa la bitácora del `worker`
El correo no llega	`SMTP_*` incorrectas o dominio sin verificar	La API registra el fallo; revisa SPF y DKIM en el proveedor
«GitHub no disponible» al crear un proyecto	Límite de 60 peticiones por hora agotado	Define `GITHUB_TOKEN` (un token sin permisos basta)
Un despliegue falla en Construcción	Error del `Dockerfile` o de la receta, o más de 10 min	La vista de despliegue marca la línea del error en la bitácora
Un despliegue falla en Ejecución	El contenedor no respondió HTTP en 60 s	La app debe escuchar en el puerto interno (`PORT`, 8080 por defecto)
Variables ilegibles	Cambió `CLAVE_CIFRADO_VARIABLES`	Restaura la clave original; la versión activa sigue sirviendo
Disco lleno en el nodo	Imágenes acumuladas	`docker image prune -f`; la retención conserva 5 artefactos por proyecto

Generar esta documentación

Los cuatro documentos viven como HTML en `docs/sitio/doc/`, con el design system v4.1 en modo claro, y se publican tal cual en **Cloudflare Pages** junto con sus PDF. Los diagramas salen de los `.mmd` de `docs/diagramas/`.

```
pnpm diagramas:sync                    # los .md y .mmd al día
node scripts/docs-sitio/construir.mjs  # diagramas, índice con páginas y los cuatro PDF
```

El script redibuja los diagramas marcados, genera cada PDF con Chromium, numera el índice con las páginas reales (con `pdftotext`) y vuelve a generar el PDF. El sitio queda en `docs/sitio/`: Cloudflare Pages lo sirve sin paso de construcción.