
This is the **published version** of the bachelor thesis:

Ye, Jiahao. *Diseño e implementación de una infraestructura de virtualización automatizada mediante Infrastructure as Code (IaC)*. Treball de Final de Grau (Universitat Autònoma de Barcelona), 2026 (Enginyeria Informàtica)

This version is available at <https://ddd.uab.cat/record/334362>

under the terms of the  license.

Diseño e implementación de una infraestructura de virtualización automatizada mediante *Infrastructure as Code* (IaC)

Jiahao Ye

Resumen— Este trabajo presenta el diseño, la implementación y la evaluación de una infraestructura de virtualización autoalojada y completamente automatizada mediante el paradigma *Infrastructure as Code* (IaC). Sobre un servidor doméstico con Proxmox VE como hipervisor, con Terraform y Ansible, se despliegan máquinas virtuales que alojan servicios de uso real —Nextcloud, Vaultwarden, PostgreSQL, una VPN WireGuard, un proxy inverso Nginx con TLS y un panel de servicios—. A todo ello se añade un sistema de copias de seguridad en dos capas y un mecanismo físico de protección frente a cortes de suministro basado en microcontroladores ESP32. La infraestructura se valida técnicamente: la reconstrucción completa de una máquina virtual desde código se consigue en unos 6 minutos, la restauración de una copia de seguridad en unos 53 segundos, las latencias de acceso por VPN se mantienen por debajo de 50 ms y el sistema sostiene unas 700 peticiones por segundo sin errores. Los resultados muestran que la IaC aporta automatización y control verificables, a la vez que evidencian sus límites prácticos: idempotencia parcial, desviación de configuración (*drift*) y la necesidad de ajustar las medidas de seguridad frente al comportamiento real de las aplicaciones.

Palabras clave— Infraestructura como código, Terraform, Ansible, Proxmox, virtualización, automatización, reproducibilidad, copias de seguridad, RTO, autoalojamiento.

Abstract— This work presents the design, implementation and evaluation of a self-hosted, fully automated virtualization infrastructure built under the *Infrastructure as Code* (IaC) paradigm. On a home server running Proxmox VE as hypervisor, virtual machines are deployed with Terraform and Ansible to host real services —Nextcloud, Vaultwarden, PostgreSQL, a WireGuard VPN, an Nginx reverse proxy with TLS and a service dashboard—. This is complemented by a two-layer backup system and a physical power-outage protection mechanism based on ESP32 microcontrollers. The infrastructure is validated quantitatively: a full rebuild of a virtual machine from code takes about 6 minutes, restoring a backup takes about 53 seconds, VPN access latencies stay below 50 ms, and the system sustains around 700 requests per second with no errors. The results show that IaC provides verifiable reproducibility and control, while also exposing its practical limits: partial idempotence, configuration drift, and the need to calibrate security measures against the real behaviour of the applications.

Keywords— Infrastructure as Code, Terraform, Ansible, Proxmox, virtualization, automation, reproducibility, backups, RTO, self-hosting.



1 INTRODUCCIÓN

LA administración tradicional de infraestructuras informáticas, basada en la configuración manual de servidores, presenta problemas bien conocidos: es lenta, gran cantidad

de errores humanos, difícil de documentar y, sobre todo, difícil de reproducir. Cada máquina configurada "a mano" acaba convertida en un *snowflake server*, un sistema único cuyo estado nadie puede reconstruir con garantías; cuando falla, su recuperación depende de la memoria del administrador y de una documentación que rara vez está completa. El paradigma *Infrastructure as Code* (IaC) responde a esta problemática tratando la infraestructura —máquinas virtuales, redes, almacenamiento y configuración de servicios— como código fuente: declarado, completo, versionado y reproducible [1].

Este Trabajo de Fin de Grado aborda la cuestión de for-

- E-mail de contacto: jiahao.ye2@autonoma.cat
- Mención realizada: Tecnologías de la Información
- Trabajo tutorizado por: Hing Fai Kevin Chow
- Curso 2025/26

ma práctica mediante la construcción de un *homelab*: una infraestructura de virtualización autoalojada y de uso real, diseñada y operada bajo los principios de la IaC. La motivación es doble. Por un lado, académica: comprobar de forma autónoma y medible las ventajas (y los límites) de la automatización declarativa. Por otro, personal y de libertad digital: disponer de servicios propios —almacenamiento de ficheros, gestor de contraseñas, acceso remoto— sin depender de proveedores externos, manteniendo el control y la privacidad de los datos. El carácter doméstico del entorno añade restricciones reales que hacen mayor el problema de ingeniería: hardware limitado, una única IP pública dinámica y la posibilidad de cortes de suministro eléctrico.

1.1. Objetivos

El objetivo general es *diseñar e implementar una infraestructura de virtualización completamente automatizada mediante IaC, capaz de ser destruida y reconstruida desde código de forma reproducible, y evaluarla cuantitativamente*. Este objetivo se concreta en los siguientes objetivos específicos:

- O1. Instalar y configurar un hipervisor *bare-metal* (Proxmox VE) con una arquitectura de red segmentada.
- O2. Automatizar el aprovisionamiento de las máquinas virtuales (VM) con Terraform mediante una arquitectura modular y reutilizable.
- O3. Automatizar la configuración de los servicios con Ansible siguiendo el patrón de roles y cuidando la idempotencia.
- O4. Desplegar servicios autoalojados de uso real, accesibles de forma segura mediante VPN, proxy inverso y TLS.
- O5. Proteger el estado de la infraestructura: copias de seguridad multinivel y resiliencia ante cortes de suministro.
- O6. Evaluar cuantitativamente la solución: tiempos de recuperación (RTO), pérdida máxima de datos (RPO), latencias de acceso y comportamiento bajo carga.

El resto del documento se organiza así: la sección 2 revisa el estado del arte; la 3 describe la metodología; la 4 presenta el diseño de la infraestructura; la 5 detalla la implementación; la 6 expone el método de evaluación, los resultados y su discusión; la 7 recoge las conclusiones y líneas futuras, y la 8 documenta el uso de la IA generativa. Los apéndices amplían el inventario, fragmentos de código, el registro de incidencias y las métricas.

2 ESTADO DEL ARTE

2.1. De la administración manual a la IaC

La IaC propone describir el estado deseado de la infraestructura en ficheros de texto versionados, de modo que las herramientas lo gestiona de forma automática y repetible [1]. Sus beneficios principales son la reproducibilidad (el entorno puede recrearse desde cero), la trazabilidad (cada cambio queda registrado y es revisable como cualquier código) y la recuperación ante desastres como procedimiento, no como improvisación. En la práctica se distinguen dos familias de herramientas complementarias: las de **aprovisionamiento**,

que crean los recursos (máquinas, discos, redes), y las de **gestión de configuración**, que llevan el interior de esos recursos al estado deseado (paquetes, servicios, ficheros). Este trabajo combina ambas.

2.2. Aprovisionamiento declarativo

Terraform [3] es la herramienta de aprovisionamiento de referencia: describe los recursos en el lenguaje declarativo HCL, mantiene un *estado* con lo realmente creado y ofrece el ciclo `plan/apply`, en el que el plan actúa como simulación revisable antes de tocar nada. Las alternativas principales son las herramientas nativas de cada nube (CloudFormation, Bicep), ligadas a un único proveedor; Pulumi, que sustituye el lenguaje declarativo por lenguajes de propósito general; y OpenTofu, bifurcación comunitaria de Terraform. Se eligió Terraform por su madurez y por la existencia de un *provider* mantenido para la API de Proxmox (`bpg/proxmox`) [4].

2.3. Gestión de configuración

Ansible [5] configura los sistemas mediante *playbooks* YAML compuestos de módulos idempotentes, en modelo *push* y sin agentes: solo requiere SSH y Python en los nodos gestionados. Frente a Puppet o Chef —que exigen desplegar agentes, mantener un servidor central y aprender lenguajes propios— y a SaltStack, Ansible minimiza la infraestructura de soporte, lo que encaja con un entorno pequeño. Su patrón de *roles* favorece además la modularidad y la reutilización.

2.4. Plataforma de virtualización

Proxmox VE [2] es una plataforma de virtualización libre basada en KVM/QEMU y LXC sobre Debian, con interfaz web y una API REST completa. Frente a VMware ESXi (modelo de licencias restrictivo), Hyper-V (ecosistema Windows) o la nube pública IaaS —que elimina el hardware propio a cambio de coste recurrente y de ceder la custodia del dato, lo contrario de la motivación del proyecto—, Proxmox ofrece coste cero, uso extendido en entornos profesionales y, sobre todo, una API automatizable desde Terraform. Como capa complementaria, las aplicaciones se ejecutan en contenedores Docker [11] dentro de las VM, combinando el aislamiento fuerte de la máquina virtual con la portabilidad y simplicidad de despliegue del contenedor.

2.5. Autoalojamiento de servicios

El movimiento *self-hosted* persigue recuperar la soberanía sobre los datos sustituyendo servicios de terceros por equivalentes propios: Nextcloud [13] como plataforma de ficheros frente a las nubes propietarias, y Vaultwarden [14], una re-implementación ligera del servidor de Bitwarden compatible con sus clientes oficiales, como gestor de contraseñas. Los retos técnicos característicos del autoalojamiento doméstico —exposición segura con una sola IP dinámica, certificados, energía— constituyen precisamente el espacio de problemas que este TFG resuelve con DNS dinámico [10], certificados de Let's Encrypt [9], una VPN WireGuard [8] y el mecanismo de resiliencia eléctrica de la sección 5.5.

2.6. Posicionamiento del trabajo

Conviene situar el enfoque adoptado frente a dos corrientes próximas. La primera es *GitOps*, que lleva la IaC un paso más allá: un operador (Flux, Argo CD) vigila continuamente el repositorio y *reconcilia* el sistema con lo declarado, en modelo *pull*. Este trabajo implementa el precursor *push* de ese modelo: Git es la única fuente de verdad, pero la aplicación de los cambios atraviesa una puerta explícita —revisar el plan de Terraform, lanzar el *playbook*—, lo que en un entorno unipersonal aporta una seguridad valiosa: ningún cambio llega al sistema sin que un humano haya leído antes qué va a ocurrir. La evolución natural, identificada como línea futura, es una tubería de integración continua que ejecute la validación y el plan en cada cambio [1]. La segunda corriente es la orquestación de contenedores con Kubernetes: para siete servicios sobre un único nodo físico, sus beneficios característicos (autoescalado, reprogramación de *Pods* entre nodos) no aplican, y su complejidad operativa sí. La combinación VM+Docker elegida ofrece el aislamiento necesario y, además, hace coincidir la unidad de aislamiento con la unidad de copia de seguridad, una propiedad que la sección 5.4 explica.

3 METODOLOGÍA

El desarrollo siguió un enfoque **iterativo e incremental**: las tareas se dividieron en bloques funcionales reducidos y cada bloque se verificó antes de avanzar, lo que permitió detectar pronto los errores de integración (Terraform–Proxmox, red de las VM clonadas). El proyecto se estructuró en seis fases: (1) aprendizaje y prototipado —hipervisor, red segmentada, plantilla base, VPN y proxy con TLS, primeras pruebas de Terraform y Ansible—; (2) desarrollo de la infraestructura con Terraform; (3) configuración de los servicios con Ansible; (4) endurecimiento y servicios adicionales —seguridad, panel, almacenamiento, copias, resiliencia eléctrica—; (5) evaluación cuantitativa; y (6) redacción de esta memoria. Al cierre de las fases 1 y 3 se entregaron sendos informes de progreso, que sirven de punto de partida documental.

Todo el código vive en un repositorio Git único (*tf-g-iac*) con dos áreas y dos puestos de trabajo: Terraform se edita desde el PC personal y Ansible desde la VM de control, resolviendo periódicamente la sincronización de código mediante fusión. Las variables sensibles están excluidas del repositorio; de hecho, un descuido inicial —archivos *vars/main.yml* con contraseñas llegaron a confirmarse— se corrigió quitando del índice (*git rm --cached*) y añadiendo el patrón a *.gitignore*, y la lección se incorporó al flujo de trabajo.

Dos principios operativos estructuran la metodología. Primero, **el código como única fuente de verdad**: no se admite ningún cambio manual persistente; toda corrección puntual se retroporta al rol de Ansible o al código de Terraform y se valida con una ejecución completa posterior. Segundo, **diagnóstico basado en hechos**: ante cada incidencia se verifica contra el sistema real (registros, *findmnt*, *printenv*, conexiones de prueba a la base de datos) antes de aceptar una hipótesis; como se verá, varias hipótesis iniciales resultaron falsas. Como apoyo educativo se emplearon herramientas de IA generativa, cuyo papel y límites se documentan en la sección 8.

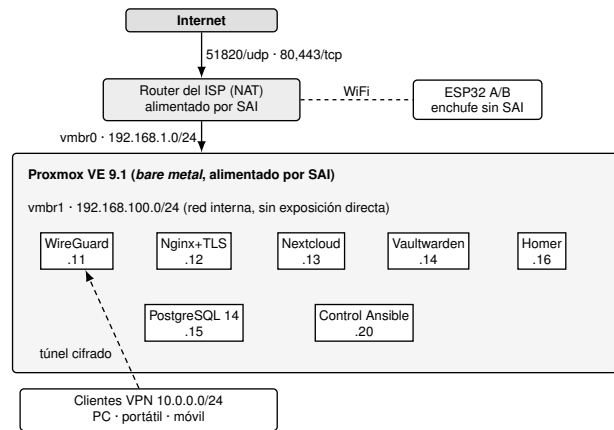


Fig. 1: Arquitectura de la infraestructura. El único tráfico entrante es la VPN y HTTPS; los servicios residen en la red interna y se alcanzan a través del proxy inverso. Los ESP32, sin SAI, actúan de «canario» de suministro eléctrico (sección 5.5).

4 DISEÑO DE LA INFRAESTRUCTURA

4.1. Hardware y plataforma

El servidor es un equipo doméstico con CPU Intel i5-10400F (6 núcleos/12 hilos), 48 GB de RAM (ampliada durante el proyecto; la incidencia asociada se recoge en el apéndice A.3), un SSD NVMe de 512 GB que aloja el sistema y los discos de las VM (*local-lvm*) y un HDD de 1 TB en un *pool ZFS* [17] (*datapool*) destinado a datos fríos y copias de seguridad. El servidor y el router están alimentados por un SAI. Sobre este hardware se instaló Proxmox VE 9.1 en modo *bare-metal*, y como base de las VM una plantilla Ubuntu Server 22.04 LTS [6] (VM 100) que Terraform clona.

4.2. Red segmentada y acceso

La red se diseñó en dos niveles (Fig. 1): el puente *vmbr0* conecta el host a la LAN doméstica (192.168.1.0/24) y el puente interno *vmbr1* (192.168.100.0/24) aloja todas las VM sin exposición directa; el host hace NAT (MASQUERADE) para su salida a internet. El router solo redirige dos puertos hacia dentro: UDP 51820 (WireGuard) y TCP 80/443 (Nginx). El nombre público *jiahaoye-tfg.duckdns.org* se mantiene con DNS dinámico [10] y el certificado TLS se emite y renueva mediante Let's Encrypt con desafío DNS-01 [9]. El circuito de acceso típico es: cliente → túnel WireGuard → Nginx (TLS y enrutado por subrutas) → servicio → PostgreSQL.

4.3. Inventario de servicios

La TABLA 1 resume las VM desplegadas. Todas las máquinas se configuran con Ansible; las creadas dentro del flujo completo de IaC están además gestionadas por Terraform.

4.4. Decisiones de diseño

Cinco decisiones articulan el diseño. (a) **Base de datos dedicada**: PostgreSQL [12] se instala de forma nativa en una VM propia y da servicio a Nextcloud y Vaultwarden,

TABLA 1: INVENTARIO DE MÁQUINAS VIRTUALES

ID	IP	Servicio	Gestión
101	.11	WireGuard (VPN)	A
102	.12	Nginx (proxy inverso, TLS)	A
103	.13	Nextcloud (Docker)	T+A
104	.14	Vaultwarden (Docker)	T+A
105	.15	PostgreSQL 14 (nativo)	T+A
106	.16	Homer (panel de servicios)	T+A
120	.20	Nodo de control Ansible	A

IPs en 192.168.100.0/24. T = Terraform, A = Ansible.

siguiendo el principio de separación de responsabilidades y centralizando copia y mantenimiento. (b) **Aplicaciones en Docker** dentro de sus VM, por simplicidad operativa y portabilidad sin renunciar al aislamiento. (c) **Exposición mínima**: aunque el dominio y el TLS son públicos, todas las rutas de aplicación están restringidas por IP a la VPN y a las redes internas; no hay motivo para exponer servicios sensibles a internet. (d) **Subrutas bajo un único dominio** (`/nextcloud`, `/vaultwarden`) para no multiplicar certificados ni subdominios. (e) **Plantilla única clonada por Terraform**, con un compromiso conocido sobre `cloud-init` que se analiza en las secciones 5.1 y 6.7.

5 IMPLEMENTACIÓN

5.1. Aprovisionamiento con Terraform

El proyecto Terraform se refactorizó desde ficheros planos hacia una arquitectura de **módulos reutilizables**: la lógica de creación de una VM se define una sola vez en `modules/vm/` y cada servicio es un bloque `module` que la instancia con sus parámetros (nombre, ID, recursos). Añadir un servicio nuevo —como ocurrió con el panel Homer [15]— se reduce a replicar un bloque, lo que demuestra que el diseño escala. Las VM se crean clonando la plantilla 100, con `agent { enabled = false }` para evitar esperas indefinidas del *provider* cuando la plantilla no ejecuta el agente QEMU.

El disco como código. Una revisión reveló que el módulo no declaraba el disco: las VM heredaban en silencio los 20 GB de la plantilla, de modo que el tamaño real no era reproducible y un cambio manual ni siquiera habría aflorado como *drift*. La solución fue un bloque `dynamic "disk"` condicionado a una variable `disk_size` (nula por defecto; apéndice A.2): solo la VM que recibe la variable renderiza el bloque, lo que permitió fijar 64 GB en PostgreSQL sin riesgo de intentar encoger el disco ya ampliado de Nextcloud. Con el mismo patrón se añadió un segundo disco de datos opcional sobre el *pool* ZFS. La regla operativa derivada: validar siempre que el plan sea `0 to destroy` antes de aplicar.

Deriva de configuración. Tras una ampliación de RAM hecha desde la interfaz web, `terraform plan` propuso revertirla en tres VM («3 to change»): el código y la realidad habían divergido. Se resolvió llevando los valores reales al código, e ilustra el coste de los cambios manuales: Terraform los detecta como *drift* y, si se aplicara sin revisar, los desharía.

Compromiso documentado. Las VM creadas a mano en

la fase 1 (WireGuard, Nginx y el nodo de control) quedaron fuera del estado de Terraform: importarlas habría exigido recrearlas, perdiendo las claves de los *peers* de WireGuard y los certificados. Su configuración sí está íntegramente en Ansible, por lo que ante una pérdida total bastaría recrearlas y reaplicar los *playbooks*.

5.2. Configuración con Ansible

Un nodo de control dedicado (VM 120) gestiona el resto de máquinas por SSH con clave `ed25519` y `sudo` sin contraseña. La configuración sigue el **patrón de roles**: `base` (paquetes comunes, cortafuegos UFW, `hostname`), `postgresql`, `postgresql_backup`, `disk_resize`, `nextcloud`, `vaultwarden`, `nginx`, `homer` y `wireguard`, orquestados por un único punto de entrada `site.yml` (apéndice A.2) que se ejecuta sin fallos sobre las siete VM. Las configuraciones se generan desde plantillas Jinja2 parametrizadas —por ejemplo, el sitio de Nginx itera sobre la lista de servicios— y los *handlers* reinician los servicios solo cuando algo cambia.

Reproducibilidad del inventario. Una tarea falló con una variable indefinida pese a estar declarada en el inventario del repositorio. La causa: `ansible.cfg` no fijaba el inventario y Ansible usaba por defecto el del sistema (`/etc/ansible/hosts`), un fichero local, divergente y no versionado. Fijar `inventory` al fichero del repositorio eliminó la dependencia oculta: una infraestructura que depende de ficheros fuera del control de versiones no es reproducible.

Los límites de la idempotencia (caso de estudio). Al rotar las credenciales de la base de datos, el módulo `postgresql_user` actualizó la contraseña en PostgreSQL de forma idempotente, pero las aplicaciones *persisten* su copia de la credencial: el `config.php` de Nextcloud conserva la de la instalación inicial y el contenedor de Vaultwarden mantiene su entorno mientras no se recree. El resultado fueron tres copias divergentes de la misma contraseña y fallos de autenticación en cadena. El diagnóstico descartó la hipótesis inicial (un carácter especial en la contraseña) probando la conexión real con `psql`, y localizó las tres ubicaciones. La resolución: unificar credenciales, editar el `config.php` dentro del contenedor y forzar la recreación del de Vaultwarden —un `up -d` no recrea el contenedor si solo cambió el fichero de entorno—. Como mejora se identificó cerrar el ciclo en el propio rol (`occ config:system:set, --force-recreate`). La lección es general: *la idempotencia declarativa llega hasta donde llega el código; si la aplicación guarda su propia configuración interna, hay que gestionarla explícitamente.*

Los guards, contra la realidad. La migración de datos de Nextcloud al disco frío usaba como condición de guarda el fichero marcador `.ocdata`; las versiones modernas usan `.ncdata`. La primera pasada omitió la migración y el servicio arrancó sobre un disco vacío. El fallo fue seguro —el origen quedó intacto— y corregir el marcador era suficiente, pero dejó otra lección: las condiciones de idempotencia deben validarse contra el sistema real, no contra suposiciones.

TABLA 2: COPIAS `vzdump` DE LAS VM CON ESTADO

VM	Disco	Copia	Bloques a cero
103 Nextcloud	50 GiB	7,3 GB	66 %
104 Vaultwarden	20 GiB	4,1 GB	56 %
105 PostgreSQL	64 GiB	4,7 GB	83 %

5.3. Seguridad

La seguridad se planteó en capas. (1) **Perímetro mínimo**: solo se exponen 51820/udp y 80,443/tcp; todo lo demás exige VPN. (2) **TLS** con desafío DNS-01, que permite emitir y renovar el certificado sin exponer el puerto 80 ni publicar previamente el servicio. (3) **Cabeceras de seguridad** (HSTS, X-Frame-Options, nosniff, Referrer-Policy, con `always`) gestionadas desde la plantilla de Nginx [7]; se decidió razonadamente *no* añadir una `Content-Security-Policy` propia, porque Nextcloud y Vaultwarden emiten la suya y duplicarla genera conflictos. En el proceso se constató el modelo de herencia de Nginx: un `add_header` local anula todos los heredados del nivel superior. (4) **Limitación de tasa** por IP con dos zonas (general y de inicio de sesión). Su calibración dejó una de las lecciones centrales del trabajo: el límite inicial (10 r/s, `burst=20`) tumbó la propia interfaz de Nextcloud, cuya SPA emite más de cien peticiones simultáneas al cargar, devolviendo errores 503 —una medida de seguridad mal calibrada se convierte en una denegación de servicio autoinfligida—. Se recalibró a 30 r/s con `burst=200` `nodelay`, valor que la sección 6.5 valida bajo carga. (5) **Secretos**: variables sensibles fuera de Git, ficheros de entorno con permisos 0600 y el directorio de volcados de la base de datos en 0700 propiedad de `postgres`.

5.4. Almacenamiento y copias de seguridad

Disco como código, de extremo a extremo. Ampliar un disco atraviesa una pila de capas que solo conocen su propio tamaño: Terraform agranda el disco virtual y, dentro de la VM, el rol `disk_resize` recorre la cadena `growpart` → `pvresize` → `lvmextend` → `resize2fs`, en caliente y de forma idempotente (los códigos de salida «sin cambios» se tratan como éxito). Así se amplió PostgreSQL de 30 a 64 GB sin reinicio.

Almacenamiento caliente/frío. En Nextcloud se separó el dato del sistema: el SSD aloja sistema operativo, Docker y la aplicación, y un segundo disco de 300 GB sobre el `pool ZFS` (aprovisionamiento `sparse`) se monta como volumen en el directorio de datos de usuario. La verificación fue de extremo a extremo: un fichero subido desde el móvil a través de la VPN acaba físicamente en el HDD. El movimiento automático de ficheros entre niveles según su antigüedad queda como trabajo futuro.

Copias en dos capas. La infraestructura es desechable; los datos no. Se combinaron dos niveles complementarios: `vzdump` de Proxmox respalda a diario (03:00, modo `snapshot`, compresión `zstd`, retención de 7 días) las tres VM con estado; es *crash-consistent* —recuperable por el *journaling* del sistema de ficheros y el WAL de la base de datos— y *sparse-aware*: no copia bloques vacíos, por lo que el tamaño depende de los datos reales y no del disco asignado

(TABLA 2). El hueco de consistencia de aplicación lo cubre un `pg_dumpall` diario (02:30) con rotación, orquestado por Ansible. Ambas capas se **encadenan**: el volcado SQL fresco queda dentro de la VM justo antes de que `vzdump` la respalde, de modo que cada imagen de copia viaja al HDD llevando dentro un volcado consistente reciente. El RPO resultante es de aproximadamente 24 horas.

5.5. Resiliencia eléctrica: el «canario» ESP32

Un corte de luz más largo que la autonomía del SAI provocaría un apagado sucio, y el hipervisor no puede «ver» el corte precisamente porque él mismo está respaldado. La solución explota una **asimetría de alimentación**: el servidor y el router cuelgan del SAI, mientras que dos microcontroladores ESP32 [18] se alimentan de enchufes de pared y mueren en el instante del corte. El host los sondea por *ping* cada 5 s y aplica un doble filtro contra falsos positivos: **redundancia** (basta que responda uno de los dos; si solo calla uno, es avería de esa placa) y **persistencia** (12 fallos consecutivos de ambos, unos 100 s, descartan fallos transitorios de WiFi). Una fase de *armado* evita el bucle de autoapagado al volver la luz: el vigilante no actúa hasta haber visto vivo al menos un ESP32. Todo corre como servicio `systemd` con rearmado automático en el nodo Proxmox —que es quien debe apagarse y el único con visibilidad de la LAN—.

El desarrollo dejó dos hallazgos de ingeniería: el ahorro de energía WiFi del ESP32, activo por defecto, le hacía perder el 90 % de los *pings* (se desactiva con `esp_wifi_set_ps(WIFI_PS_NONE)`), y su pila de red no tolera ráfagas ICMP, lo que obliga a un sondeo espaciado —que es además lo que el diseño necesita—.

Notificación remota. Con el router en el SAI, el servidor conserva red durante el corte, lo que permitió añadir un aviso *push* mediante `ntfy` [16] justo antes del apagado, con un tiempo máximo corto: la notificación es «mejor esfuerzo» y nunca debe bloquear el `poweroff`. El servicio de notificación es deliberadamente *externo* a la infraestructura: una instancia autoalojada moriría con el propio corte que debe comunicar. El sistema completo se **probó en real**: al desconectar ambos ESP32, el servidor confirmó el corte, registró el evento con fecha y hora, envió la notificación —recibida en el móvil— y se apagó limpiamente; al restablecer la alimentación, el servicio rearmó la vigilancia por sí solo.

6 EVALUACIÓN Y RESULTADOS

6.1. Método: métricas y escenarios

Se adoptaron dos métricas estándar de continuidad: el **RTO** (*Recovery Time Objective*), tiempo real de recuperación de un servicio tras un fallo, y el **RPO** (*Recovery Point Objective*), pérdida máxima de datos expresada en tiempo. Sobre ellas se definieron los escenarios: **A)** recuperación de una VM sin estado (el panel Homer) destruyéndola y reconstruyéndola íntegramente desde código, que mide el RTO puro de la IaC sin riesgo de datos; **B)** restauración no destructiva de una copia `vzdump` de una VM con estado a un identificador temporal, que verifica que las copias son restaurables; y **C)** destrucción real de la base de datos en producción con restauración del volcado, que se descartó por

TABLA 3: ESCENARIO A: TRAMOS DEL RTO (VM HOMER)

Tramo	Contenido	Tiempo
T ₁	Terraform (<code>apply</code>)	53 s
T ₂	Manual: IP, claves SSH, <code>sudo</code>	≈ 2,5 min
T ₃	Ansible (<code>site.yml</code>)	2 min 33 s
RTO	Recuperación completa	≈ 6 min

prudencia operativa: su valor incremental quedaba cubierto por la combinación de A y B. Se midieron además la latencia de acceso por VPN y el comportamiento bajo carga.

6.2. Escenario A: reconstrucción desde código

La reconstrucción *no* es totalmente automática, y esa es precisamente una de las observaciones más valiosas: la VM clonada nace con la IP de la plantilla y sin acceso para Ansible, lo que introduce una «isla manual» entre los dos tramos automáticos. Por ello el RTO se midió en tres tramos (TABLA 3): T₁, el `apply` de Terraform; T₂, la intervención manual (cambio de IP, intercambio de claves SSH y `sudo` sin contraseña); y T₃, la reconfiguración completa con Ansible. La destrucción (2 s) no forma parte del RTO: es la provocación del fallo, no su recuperación.

T₁ resultó estable entre repeticiones (54 s y 53 s). Con el agente QEMU deshabilitado, Terraform da por creada la VM en cuanto Proxmox la arranca, por lo que el arranque real del sistema se absorbe al inicio de T₂. T₃ está dominado por la red, no por la lógica: incluye la actualización del sistema y la instalación de Docker desde cero, lo realista en una reconstrucción. La verificación final fue exigente: la VM reconstruida respondió 200 OK con todas las cabeceras de seguridad y con `Last-Modified` y `Content-Length` idénticos a los previos a la destrucción; la reconstrucción no solo «arranca», sino que sirve exactamente el mismo contenido. Una incidencia metodológica mereció registro: el primer intento de T₂ se hizo por SSH a la IP que se iba a cambiar y la sesión quedó colgada al aplicar la nueva configuración, contaminando la medida; se repitió el tramo limpio desde la consola del hipervisor, porque la métrica que se defiende debe estar libre de ruido.

6.3. Escenario B: restauración de copias

Antes de restaurar se comprobó que las copias **automáticas** se generan solas: el almacén contenía los artefactos de las 02:30 (`pg_dumpall`) y de las 03:00 (`vzdump`) creados sin intervención. La copia de la VM de PostgreSQL se restauró con `qmrestore` a un identificador temporal en **52,8 s**, y la VM restaurada se arrancó con la tarjeta de red desconectada (`link_down`) para evitar el conflicto de IP con la VM productiva, que no se tocó en ningún momento. La verificación confirmó que el clúster levanta y que las bases de datos reales (`nextcloud`, `vaultwarden`) sobreviven intactas al ciclo copia–restauración; después se eliminó la VM temporal. La restauración aislada del volcado SQL se omitió de forma razonada: el diseño encadenado garantiza que cada imagen `vzdump` verificada contiene ya un volcado fresco; de ejecutarse, exigiría un clúster temporal en otro

TABLA 4: DOS VÍAS DE RECUPERACIÓN, COMPARADAS

Vía	Reconstruye	Tiempo
IaC (Terraform+Ansible)	VM desde código	≈ 6 min
Restauración (<code>qmrestore</code>)	VM y sus datos	52,8 s

La restauración depende de la última copia (RPO ≈ 24 h); la vía IaC no depende de copia alguna.

TABLA 5: LATENCIA DE SERVICIO A TRAVÉS DE LA VPN

Servicio	Media	Rango
Homer (/)	19,8 ms	18,9 – 20,8 ms
Nextcloud (/nextcloud/)	45,8 ms	45,2 – 46,8 ms
Vaultwarden (/alive)	21,0 ms	20,3 – 22,3 ms

Estado estacionario; el establecimiento inicial del túnel añade ≈ 2 s a la primera petición.

puerto, pues `pg_dumpall` incluye los roles globales y no debe restaurarse sobre producción.

La TABLA 4 compara ambas vías: la restauración es mucho más rápida porque copia un estado ya hecho, pero solo recupera hasta la última copia; la reconstrucción desde código tarda más pero no depende de que exista copia alguna. Son complementarias —defensa en profundidad—, no rivales.

6.4. Latencia de acceso por VPN

Las primeras medidas resultaron erráticas, con picos de unos 2 s mezclados con valores de decenas de milisegundos. La causa no era el servidor: WireGuard rehace el *handshake* del túnel tras un periodo de inactividad y carga ese coste sobre la primera petición. Distinguir la **latencia de establecimiento** de la **latencia de servicio** en estado estacionario es imprescindible para no concluir que «Nextcloud tarda 2 s» cuando en realidad tarda 46 ms. Tras calentar el túnel, cinco medidas por servicio dieron los valores de la TABLA 5: todas por debajo de 50 ms y con una consistencia de 1–2 ms. Que Nextcloud duplique a los demás es lo esperable: ejecuta una aplicación PHP completa, frente al HTML estático de Homer o la ruta mínima de comprobación de Vaultwarden.

6.5. Comportamiento bajo carga

Con Apache Bench se sometió al circuito completo (VPN → Nginx → Homer) a tres niveles de carga (TABLA 6). El sistema sostuvo un rendimiento estable de unas 685–715 peticiones/s con **cero fallos** hasta 500 conexiones concurrentes; lo único que creció fue la latencia (13 → 640 ms). El sistema mostró una degradación progresiva: al aumentar la carga, las respuestas tardaron más, pero no se produjeron errores ni peticiones rechazadas.

Que el limitador de tasa no rechazara ni una petición es un resultado correcto, no un fallo, y se explica técnicamente: la zona limita por IP de origen y toda la carga procedía de una sola máquina; con `burst=200 nodelay`, Nginx sirve la ráfaga de inmediato y, como el backend responde en ≈20 ms, el cubo se vacía más rápido de lo que se llena. El resultado cierra la narrativa de la sección 5.3: la misma medida que mal calibrada (`burst=20`) rompía Nextcloud absorbe ahora

TABLA 6: PRUEBAS DE CARGA SOBRE EL CIRCUITO COMPLETO

Concurrencia	Peticiones/s	Fallos	Mediana
10	679	0	13 ms
100	715	0	131 ms
500	685	0	640 ms

100, 2 000 y 5 000 peticiones totales respectivamente.

5 000 peticiones a 500 concurrentes sin un solo rechazo; está dimensionada para frenar el abuso sostenido sin penalizar jamás el uso legítimo.

6.6. Alcance y validez de las medidas

Las cifras anteriores deben leerse con su alcance. Las latencias se midieron con $n = 5$ muestras por servicio tras calentar el túnel, separando el coste único del *handshake* (≈ 2 s) del régimen estacionario; son medias sobre una red doméstica y un instante concretos, no un acuerdo de nivel de servicio. La prueba de carga se lanzó desde un equipo de la propia LAN, una decisión deliberada: aísla el rendimiento de la pila del servidor (Nginx, aplicación, base de datos) de la variabilidad de Internet y de la VPN, por lo que las ≈ 700 peticiones/s son una cota de la pila local, no del acceso remoto. El RTO se midió de extremo a extremo una sola vez, con una repetición del tramo automatizado que confirmó su estabilidad (54 s frente a 53 s); el tramo de configuración depende del estado de los espejos y de la caché de paquetes, por lo que presentará varianza entre ejecuciones. En conjunto, los valores absolutos son específicos de este hardware; lo transferible es el **método**: instrumentar cada tramo por separado, verificar el resultado con un criterio de fidelidad explícito y descartar las mediciones contaminadas —como la primera del escenario A, repetida desde la consola tras detectar que el cambio de IP cortaba la propia sesión SSH desde la que se cronometraba—.

6.7. Discusión

El hallazgo más relevante de la evaluación es que la «isla manual» T_2 supone el 40–45 % del RTO total y es íntegramente evitable: la vía canónica es reconstruir la plantilla desde una imagen *cloud* con cloud-init, de modo que Terraform inyecte la configuración de red (`initialization { ip_config }`) y la VM nazca lista para Ansible. La mejora queda así justificada con un dato, no como carencia genérica; las alternativas intermedias (ejecución remota desde el puesto de Terraform, que no alcanza la red interna, o el agente invitado, deshabilitado a propósito) se evaluaron y descartaron con criterio. Medir y publicar honestamente la isla manual aporta más valor que ocultarla. En cuanto a las limitaciones: el RPO de 24 h es mejorable aumentando la frecuencia de copia; y el escenario A usa una VM sin estado —representativa del flujo completo de IaC—, mientras que para las VM con estado la recuperación combina ese flujo con la restauración medida en la sección 6.3.

Merece mención, por último, el coste operativo de toda la protección: la ventana diaria de copia de las tres VM con estado ronda los tres minutos en total y, gracias al tratamiento de los bloques vacíos, los ficheros ocupan entre un 56 y

un 83 % menos que los discos que respaldan (TABLA 2); la resiliencia eléctrica, por su parte, se sostiene con dos microcontroladores de coste marginal y un servicio del sistema. La protección del estado resulta, en la práctica, casi gratuita frente al valor que aporta: una vía de restauración diaria y verificada de 52,8 s.

7 CONCLUSIONES

Se ha construido una infraestructura de virtualización autoalojada, operativa y de uso real, gestionada de extremo a extremo como código, y se ha evaluado con métricas. Los seis objetivos se cumplieron: hipervisor y red segmentada (O1); aprovisionamiento modular con Terraform, incluido el disco como código (O2); configuración íntegra por roles de Ansible con un único punto de entrada (O3); servicios reales accesibles de forma segura por VPN con TLS, verificados desde múltiples dispositivos (O4); protección del estado mediante copias encadenadas en dos capas y resiliencia eléctrica probada en real (O5); y evaluación cuantitativa: RTO de reconstrucción ≈ 6 min, restauración en 52,8 s, latencias inferiores a 50 ms y ≈ 700 peticiones/s sin errores (O6).

La tesis del trabajo queda comprobada con matices que la potencian. La IaC aporta una reproducibilidad *verificable* —la VM reconstruida sirve contenido byte a byte idéntico— y convierte la recuperación ante desastres en un procedimiento con números. Pero exige comprender sus límites: la idempotencia termina donde las aplicaciones persisten estado propio; los cambios manuales generan deriva que el plan detecta y castiga; la frontera real no separa herramientas, sino *infraestructura* (reproducible desde código) de *estado* (que exige copias por capas); las medidas de seguridad deben calibrarse contra el comportamiento real de las aplicaciones; y lo que no puede ser código —la configuración del propio hipervisor— debe quedar documentado como procedimiento reproducible.

Como líneas de continuación se identifican: reconstruir la plantilla con cloud-init para eliminar el tramo manual del RTO (la mejora de mayor impacto, justificada con dato); adoptar los módulos `community.docker` para idempotencia plena de los contenedores; endurecer Vaultwarden (cerrar el registro, panel de administración); el movimiento automático de datos entre los niveles caliente y frío; monitorización con Prometheus y Grafana; aumentar la frecuencia de copias para reducir el RPO; y los módulos opcionales ya diseñados de IA local (Open WebUI como pasarela hacia Ollama) y sincronización de notas (Memos).

En el plano formativo, el proyecto ha obligado a integrar competencias transversales de la mención de Tecnologías de la Información —redes, virtualización, administración de sistemas, seguridad y automatización— alrededor de un sistema real cuyo comportamiento no perdona las suposiciones. Las lecciones más valiosas (los límites de la idempotencia, la calibración de la seguridad contra el comportamiento real de las aplicaciones, la disciplina de retroportar todo cambio al código) no estaban en la documentación de las herramientas: emergieron de operar la infraestructura y de medirla.

8 USO DE LA IA GENERATIVA

Durante todo el proyecto se ha utilizado IA generativa (Claude, de Anthropic) como herramienta de apoyo, con un papel deliberadamente acotado y que se documenta aquí con transparencia. Sus usos concretos fueron: (a) resolución de dudas conceptuales y explicación de mecanismos (el modelo de herencia de cabeceras de Nginx, la cadena de capas LVM, la diferencia entre consistencia de caída y de aplicación); (b) contraste de decisiones de arquitectura, valorando alternativas antes de decidir; (c) acompañamiento en el diagnóstico de incidencias, proponiendo hipótesis que *siempre* se verificaron contra el sistema real —de hecho, varias hipótesis iniciales resultaron incorrectas y fue la verificación fáctica la que condujo a la causa raíz—; y (d) apoyo a la estructuración y redacción de los informes y de esta memoria a partir de los registros técnicos elaborados por el autor durante las sesiones de trabajo.

Todas las decisiones, la ejecución de los cambios y la validación final corresponden al autor; la información se contrastó con la documentación oficial citada en la bibliografía, y ningún fragmento de código se incorporó sin comprenderlo y probarlo, en coherencia con la metodología de la sección 3. La experiencia resultante es que la IA actúa como un multiplicador de productividad y de aprendizaje cuando se combina con verificación empírica y criterio propio, y que declarar su uso forma parte de la honestidad académica del trabajo.

AGRADECIMIENTOS

A mi tutor, Hing Fai Kevin Chow, por la orientación, la disponibilidad y el seguimiento del trabajo a lo largo del curso. A mi familia y amigos, por el apoyo constante durante el grado.

REFERENCIAS

- [1] K. Morris, *Infrastructure as Code: Dynamic Systems for the Cloud Age*, 2.ª ed. Sebastopol, CA: O'Reilly Media, 2020.
- [2] Proxmox Server Solutions GmbH, «Proxmox VE Administration Guide». <https://pve.proxmox.com/pve-docs/>
- [3] HashiCorp, «Terraform Documentation». <https://developer.hashicorp.com/terraform/docs>
- [4] «Terraform Provider for Proxmox (bpg/proxmox)». <https://registry.terraform.io/providers/bpg/proxmox>
- [5] Red Hat, Inc., «Ansible Documentation». <https://docs.ansible.com/>
- [6] Canonical Ltd., «Ubuntu Server Documentation». <https://documentation.ubuntu.com/server/>
- [7] F5, Inc., «NGINX Documentation». <https://docs.nginx.com/>
- [8] J. A. Donenfeld, «WireGuard: Fast, Modern, Secure VPN Tunnel». <https://www.wireguard.com/>
- [9] Internet Security Research Group, «Challenge Types — Let's Encrypt». <https://letsencrypt.org/docs/challenge-types/>
- [10] DuckDNS, «Specification and install guides». <https://www.duckdns.org/spec.jsp>
- [11] Docker, Inc., «Docker Documentation». <https://docs.docker.com/>
- [12] The PostgreSQL Global Development Group, «PostgreSQL 14 Documentation». <https://www.postgresql.org/docs/14/>
- [13] Nextcloud GmbH, «Nextcloud Administration Manual». <https://docs.nextcloud.com/>
- [14] D. García et al., «Vaultwarden — wiki del proyecto». <https://github.com/dani-garcia/vaultwarden/wiki>
- [15] B. Wirtz, «Homer: a very simple static homepage for your server». <https://github.com/bastienwirtz/homer>
- [16] P. Heckel, «ntfy — Send push notifications to your phone». <https://docs.ntfy.sh/>
- [17] OpenZFS Project, «OpenZFS Documentation». <https://openzfs.github.io/openzfs-docs/>
- [18] Espressif Systems, «ESP-IDF Programming Guide: Wi-Fi driver». <https://docs.espressif.com/projects/esp-idf/>

APÉNDICE

A.1. Inventario detallado y repositorio

La TABLA 7 amplía el inventario con los recursos asignados a cada VM. La red VPN (10.0.0.0/24) tiene como pasarela el servidor WireGuard (10.0.0.1) y como clientes el PC (10.0.0.2), el portátil (10.0.0.3) y el móvil (10.0.0.4), con túnel dividido: solo el tráfico hacia las redes del laboratorio atraviesa la VPN.

El repositorio único `tfg-iac` se organiza en dos áreas, editadas desde puestos distintos:

```
tfg-iac/
|-- terraform/                (PC personal)
|   |-- main.tf variables.tf providers.tf
|   |-- modules/vm/
|       |-- main.tf variables.tf outputs.tf
|-- ansible/                  (VM 120, nodo de control)
|   |-- ansible.cfg site.yml
|   |-- inventario/hosts.ini
|   |-- roles/
|       |-- base/ postgresql/ postgresql_backup/
|       |-- disk_resize/ nextcloud/ vaultwarden/
|       |-- nginx/ homer/ wireguard/
```

A.2. Fragmentos de código representativos

Disco como código (Terraform). Bloque condicional del módulo `modules/vm`: solo se renderiza si la VM recibe la variable, aislando el cambio.

```
variable "disk_size" {
  type    = number
  default = null
}

dynamic "disk" {
  for_each = var.disk_size != null ? [1] : []
  content {
    datastore_id = "local-lvm"
    interface    = "virtio0"
    size         = var.disk_size
  }
}
```

Punto de entrada único (Ansible). Extracto de `site.yml`: el rol común precede a los roles específicos de cada grupo.

```
- hosts: all
  become: true
  roles: [base]

- hosts: postgresql
  become: true
  roles: [disk_resize, postgresql,
         postgresql_backup]

- hosts: nextcloud
  become: true
  roles: [nextcloud]

- hosts: nginx
  become: true
  roles: [nginx]
# ... vaultwarden, homer, wireguard
```

Limitación de tasa y subruta (Nginx). Zona por IP de origen y bloque de Vaultwarden conservando el prefijo de la subruta y el soporte WebSocket (las dos causas del 404 del apéndice A.3):

```
limit_req_zone $binary_remote_addr
              zone=general:10m rate=30r/s;

location /vaultwarden/ {
  limit_req zone=general burst=200 nodelay;
  proxy_pass
    http://192.168.100.14:8080/vaultwarden/;
```

```
proxy_set_header Host $host;
proxy_set_header Upgrade $http_upgrade;
proxy_set_header Connection "upgrade";
proxy_http_version 1.1;
}
```

Esqueleto del «canario» (nodo Proxmox). Estructura de `canary.sh`; la URL de notificación se omite deliberadamente (en el servicio público de ntfy el nombre del *topic* actúa como secreto).

```
ESP_A=192.168.1.50; ESP_B=192.168.1.51
UMBRAL=12; INTERVALO=5
NTFY_URL="https://ntfy.sh/<topic>" # censurado
armado=0; fallos=0
while true; do
  if ping -c1 -W1 $ESP_A >/dev/null \
    || ping -c1 -W1 $ESP_B >/dev/null; then
    armado=1; fallos=0
  elif [ $armado -eq 1 ]; then
    fallos=$((fallos+1))
    if [ $fallos -ge $UMBRAL ]; then
      log "CORTE DE LUZ CONFIRMADO"
      curl --max-time 5 -d "Corte de luz: \
apagado limpio" "$NTFY_URL" || true
      sync
      [ "$DRY_RUN" = "1" ] || poweroff
    fi
  fi
  sleep $INTERVALO
done
```

A.3. Registro de incidencias y lecciones

La TABLA 8 consolida las incidencias más formativas del proyecto. Más allá de su resolución, constituyen la evidencia empírica de las lecciones de la sección 7.

A.4. Métricas consolidadas y procedimientos

La TABLA 9 consolida las métricas. Se documentan además, como procedimientos reproducibles, las dos operaciones que quedan fuera del código:

(a) Incorporación de una VM recién clonada (origen del tramo T₂). Desde la *consola web* de Proxmox de la VM —nunca por SSH a la IP que se va a cambiar—: sustituir la IP de la plantilla por la definitiva en el fichero de `netplan`, deshabilitar la gestión de red de `cloud-init` (que de lo contrario restaura la IP original al reiniciar) y aplicar la configuración. Después, desde el nodo de control: retirar la huella SSH antigua si la VM se ha recreado (`ssh-keygen -R`), copiar la clave pública (`ssh-copy-id`) y, dentro de la VM, habilitar `sudo` sin contraseña para el usuario de automatización. La verificación es `ansible -m shell -a "whoami" --become`, que debe devolver `root`.

(b) Configuración del hipervisor (vive fuera de Git por naturaleza): el registro de los almacenes sobre el *pool* ZFS (`pvesm add` para copias y para discos de VM con aprovisionamiento *sparse*), el trabajo de copia programado (`pvesh create /cluster/backup`, modo *snapshot*, compresión `zstd`, retención 7) y la instalación del servicio `canary.service` de `systemd`. Cada uno queda recogido con sus órdenes exactas en la documentación del repositorio, de modo que un hipervisor nuevo pueda reproducirlos.

TABLA 7: INVENTARIO DETALLADO DE MÁQUINAS VIRTUALES

ID	Servicio	IP (192.168.100.x)	vCPU	RAM	Disco(s)	Gestión
100	Plantilla ubuntu-base (Ubuntu 22.04)	.10	2	2 GB	30 GB (SSD)	—
101	WireGuard (VPN)	.11	2	2 GB	30 GB (SSD)	Ansible
102	Nginx (proxy inverso, TLS)	.12	2	2 GB	30 GB (SSD)	Ansible
103	Nextcloud (Docker)	.13	2	8 GB	50 GB (SSD) + 300 GB (HDD)	Terraform + Ansible
104	Vaultwarden (Docker)	.14	1	8 GB	20 GB (SSD)	Terraform + Ansible
105	PostgreSQL 14 (nativo)	.15	1	6 GB	64 GB (SSD)	Terraform + Ansible
106	Homer (panel)	.16	1	1 GB	30 GB (SSD)	Terraform + Ansible
120	Nodo de control Ansible	.20	2	6 GB	30 GB (SSD)	Ansible

Los valores de las VM 101, 102 y 120 proceden de la plantilla (compromiso de la sección 5.1). El acceso al panel de Proxmox y a los servicios requiere VPN.

TABLA 8: INCIDENCIAS DESTACADAS: CAUSA RAÍZ, RESOLUCIÓN Y LECCIÓN

Incidencia	Causa raíz	Resolución	Lección
Fallo de autenticación en Nextcloud y Vaultwarden tras rotar credenciales	La aplicación persiste su copia de la credencial (<code>config.php</code> ; entorno del contenedor); la BD sí se actualizó	Unificar credenciales, editar <code>config.php</code> y recrear el contenedor (<code>up -d</code> no recrea por sí solo)	La idempotencia termina donde la aplicación guarda estado propio
Variable de inventario «indefinida» pese a estar declarada	<code>ansible.cfg</code> sin <code>inventory</code> : se usaba el fichero local del sistema, divergente y sin versionar	Fijar el inventario del repositorio en <code>ansible.cfg</code>	Las dependencias implícitas fuera de Git rompen la reproducibilidad
<code>terraform plan</code> propone reducir la RAM de tres VM	Ampliaciones hechas a mano desde la interfaz web (deriva de configuración)	Llevar los valores reales al código y verificar <code>plan</code> limpio	El código es la única fuente de verdad; el plan detecta la deriva
Interfaz de Nextcloud vacía con errores 503	Limitación de tasa estricta (<code>burst=20</code>) frente a una SPA que emite >100 peticiones al cargar	Recalibrar a 30 <code>r/s</code> , <code>burst=200</code> <code>nodelay</code> ; validar bajo carga	Una medida de seguridad mal calibrada es una denegación de servicio autoinfligida
Migración de datos al HDD omitida	Condición de guarda con marcador obsoleto (<code>.ocdata</code> frente a <code>.ncdata</code>)	Corregir el marcador y relanzar; el origen quedó intacto	Validar los <i>guards</i> contra el sistema real; fallar hacia el lado seguro
<code>rm</code> «sin efecto» sobre la copia antigua del SSD	El <code>bind mount</code> del HDD enmascara la ruta del host subyacente	Verificar con <code>findmnt</code> y <code>docker exec</code> ; posponer la limpieza	Los montajes pueden ocultar las rutas sobre las que se cree operar
404 de la app móvil de Bitwarden (el navegador sí entraba)	Doble causa en capas: faltaba <code>DOMAIN</code> y la barra final de <code>proxy_pass</code> recortaba la subruta	Definir <code>DOMAIN</code> ; conservar el prefijo y añadir <code>WebSocket</code>	Diagnosticar por capas: aplicación y proxy fallaban a la vez
Sesión SSH colgada al cambiar la IP de una VM clonada	La nueva configuración de red elimina la IP por la que viaja la propia sesión	Operar desde la consola del hipervisor; repetir la medición limpia	Las métricas que se defienden deben estar libres de ruido
El ESP32 pierde el 90 % de los <i>pings</i>	Ahorro de energía WiFi activo por defecto	Desactivarlo: <code>esp_wifi_set_ps(WIFI_PS_NONE)</code>	Los valores por defecto orientados a batería penalizan la disponibilidad
Pérdidas con <i>ping</i> en ráfaga pese a buena señal	La pila de red del módulo no tolera ICMP a ritmo alto	Sondeo espaciado (cada 5 s), coherente con el diseño	Adaptar el sondeo a las limitaciones del dispositivo
El servidor no arranca (POST) con 4 módulos de RAM	Perfil XMP inestable al mezclar series distintas del mismo fabricante	Desactivar XMP y operar a la velocidad base JEDEC	En un servidor, la estabilidad prima sobre el rendimiento marginal

TABLA 9: MÉTRICAS CONSOLIDADAS DE LA EVALUACIÓN

Métrica	Valor
RTO de reconstrucción IaC (VM sin estado)	≈ 6 min (T_1 53 s; T_2 ≈2,5 min; T_3 2 min 33 s)
Tramo manual T_2 sobre el RTO	40–45 % (evitable con <code>cloud-init</code>)
Restauración <code>vzdump</code> (64 GiB, <i>sparse</i>)	52,8 s
RPO (copias diarias encadenadas)	≈ 24 h (retención 7 días)
Copia de las 3 VM con estado	≈ 3 min en total
Latencia de servicio por VPN	19,8 / 45,8 / 21,0 ms (Homer / Nextcloud / Vaultwarden)
Establecimiento del túnel VPN	≈ 2 s (solo primera petición)
Rendimiento bajo carga	685–715 pet./s, 0 fallos hasta 500 concurrentes
Ejecución completa de <code>site.yml</code> (estado estable)	1–2 min, 0 fallos en 7 VM