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

Iglesias Gonzalez, Àlex. *Implementación de una Plataforma Cloud-Native basada en Azure Kubernetes Service mediante Infraestructura como Código*. Treball de Final de Grau (Universitat Autònoma de Barcelona), 2026 (Enginyeria Informàtica)

This version is available at <https://ddd.uab.cat/record/334389>

under the terms of the  license.

Implementación de una Plataforma Cloud-Native basada en Azure Kubernetes Service mediante Infraestructura como Código

Álex Iglesias González

Resumen—Este trabajo aborda el diseño y la implementación de una plataforma *cloud-native* basada en Azure Kubernetes Service (AKS), definida íntegramente mediante Infraestructura como Código (IaC) con Terraform, en un entorno empresarial real del sector asegurador. La plataforma se concibe con carácter transversal para alojar microservicios de múltiples aplicaciones —entre ellas *Pólizas On Demand*— sobre una arquitectura de red Azure *Hub & Spoke*, con un clúster privado, salida controlada a través de *firewall*, integración de identidad con Microsoft Entra ID, gestión de secretos sin credenciales estáticas mediante OIDC y Workload Identity, y un registro de contenedores (ACR) privado. La operación del clúster se resuelve con un modelo GitOps basado en Argo CD y el patrón *App of Apps*, complementado con un conjunto de *pipelines* reutilizables de GitHub Actions que construyen, escanean y promocionan imágenes de forma determinista. El resultado es una plataforma funcional, segura y reproducible, validada mediante pruebas técnicas documentadas. Durante el desarrollo se diagnosticó y reportó un *bug* de producto de Azure Virtual Network Manager (AVNM), confirmado y resuelto con el soporte de Microsoft, que ilustra la complejidad de operar servicios gestionados en entornos corporativos.

Palabras clave—Kubernetes, AKS, Infraestructura como Código, Terraform, Cloud-Native, GitOps, Argo CD, Azure, Hub & Spoke, Workload Identity.

Abstract—*This work addresses the design and implementation of a cloud-native platform based on Azure Kubernetes Service (AKS), fully defined through Infrastructure as Code (IaC) with Terraform, in a real enterprise environment within the insurance sector. The platform is designed as a cross-cutting service to host microservices from multiple applications —including Pólizas On Demand— on top of an Azure Hub & Spoke network architecture, featuring a private cluster, controlled egress through a firewall, identity integration with Microsoft Entra ID, secretless authentication via OIDC and Workload Identity, and a private container registry (ACR). Cluster operations rely on a GitOps model built on Argo CD and the App of Apps pattern, complemented by reusable GitHub Actions pipelines that build, scan and promote images deterministically. The outcome is a functional, secure and reproducible platform, validated through documented technical tests. During development, a product bug in Azure Virtual Network Manager (AVNM) was diagnosed, reported, confirmed and resolved with Microsoft support, illustrating the complexity of operating managed services in corporate environments.*

Index Terms—Kubernetes, AKS, Infrastructure as Code, Terraform, Cloud-Native, GitOps, Argo CD, Azure, Hub & Spoke, Workload Identity.

1. INTRODUCCIÓN

La transformación hacia arquitecturas de microservicios y la adopción del paradigma *cloud-native* han convertido la orquestación de contenedores en la piedra angular de las plataformas modernas. Sin embargo, materializar este paradigma en un entorno empresarial real, con requisitos de seguridad, conectividad privada, compliance y gobernanza estrictos, supone todo un reto, cuya dificultad va mucho más allá de la mera configuración de unos cuantos servicios en la nube.

Este trabajo de fin de grado se ha llevado a cabo dentro del departamento de Sistemas Cloud de Occident, un grupo asegurador internacional que opera en más de 50 países. Actualmente, Occident opera una gran cantidad de recursos y servicios en la nube, entre ellos una plataforma de

orquestación de contenedores basada en Azure Kubernetes Service, que da soporte principalmente a *Pólizas On Demand*, una aplicación que sirve para contratar seguros de corta duración bajo demanda. Sin embargo, esta plataforma no sigue estándares modernos de Kubernetes e Infraestructura como Código, y no está alineada con los estándares de seguridad y gobernanza de Occident. Este hecho ha generado la necesidad de una plataforma de referencia moderna, completamente definida mediante IaC, que cumpla con los estándares de seguridad y gobernanza de Occident, que sea mantenible, reproducible y solucione los problemas de operación y trazabilidad que introduce la gestión manual de la plataforma.

El objeto del trabajo es, por tanto, diseñar e implementar una plataforma de orquestación de contenedores basada en AKS, definida completamente mediante Terraform, sobre la arquitectura de red *Hub & Spoke* de la organización. El clúster se concibe con carácter transversal privado para alojar tanto *Pólizas On Demand* como otras cargas de trabajo que

- A. Iglesias González. Grado en Ingeniería Informática. TFG desarrollado en el Departamento de Sistemas Cloud de GCO, Tecnología y Servicios. Junio de 2026.
Tutor de empresa: Joel Cantero Priego.
Tutor de universidad: David Gutiérrez Yana.

pueda tener la compañía en el presente y en el futuro. De igual manera, el Azure Container Registry se concibe como un recurso transversal privado que sirve tanto para alojar imágenes de contenedores del clúster como para otros casos de uso.

La arquitectura *Hub & Spoke* condiciona de forma determinante el diseño: un *Hub* centralizado concentra los servicios de conectividad compartidos, como Azure Firewall, DNS privado, IPAM y Azure Virtual Network Manager, mientras que cada carga de trabajo se despliega en un *Spoke* independiente. El clúster AKS se despliega como *Spoke*, cuyo tráfico de salida pasa íntegramente por el *firewall* del *Hub*, lo que añade requisitos de configuración gestionados con IaC: reglas de *firewall*, User Defined Routes (UDR), Network Security Groups (NSG), DNS privado e integración con Azure Virtual Network Manager (AVNM).

El alcance de este trabajo se limita al entorno de desarrollo; los servicios como Log Analytics, Azure Monitor y el propio *Hub* ya existían y se han integrado en la solución propuesta.

2. ESTADO DEL ARTE

El enfoque **Cloud-Native** combina contenedores, micro-servicios, entrega continua y orquestación automatizada. **Kubernetes**, desarrollado por Google y donado a la CNCF en 2015 [1], es el estándar de facto para la orquestación de contenedores, con capacidades de reconciliación declarativa, *self-healing*, descubrimiento de servicios y autoscaling.

Azure Kubernetes Service (AKS) [2] es el servicio gestionado de Kubernetes de Microsoft Azure, en el que Microsoft gestiona el control plane. Se integra de forma nativa con Entra ID, Azure Monitor, Azure Policy y las *Virtual Networks*. En este proyecto se ha optado por **Azure CNI Overlay con data plane Cilium (eBPF)** [3], que asigna las IPs de los pods desde un espacio de direcciones lógico superpuesto, desacoplándolas de la red del *Spoke*. Esta decisión optimiza el consumo de direcciones IP a la vez que aprovecha las capacidades de *network policy* de Cilium.

Terraform (HashiCorp) [4] es una herramienta de IaC declarativa que calcula las acciones necesarias para alcanzar el estado deseado de la infraestructura. La organización ha desarrollado una biblioteca de módulos reutilizables con versionado semántico estricto y un proceso de *release* automatizado mediante GitHub Actions.

La **arquitectura Hub & Spoke** [5], [6], [7], documentada en el *Cloud Adoption Framework* y el *Well-Architected Framework* de Microsoft, centraliza en el *Hub* los componentes de gobernanza: Azure Firewall, AVNM, *Private DNS Zones* e IPAM. El tráfico de salida se configura con **UDR** en lugar de *Load Balancer*, para evitar la exposición accidental del clúster a Internet [8].

El modelo **GitOps** —popularizado por **Argo CD** [9]— propone gestionar el estado del clúster de forma declarativa desde un repositorio Git como única fuente de verdad, con reconciliación continua. El patrón **App of Apps** permite componer y ordenar el despliegue de los componentes de plataforma. Para la exposición de servicios, la **Gateway API** de Kubernetes [10] con implementaciones como **NGINX Gateway Fabric** [11] representa la evolución del clásico *Ingress*, con un modelo de roles y recursos más expresivo.

3. OBJETIVOS

El objetivo general de este trabajo es diseñar e implementar una plataforma *cloud-native* basada en Azure Kubernetes Service, definida completamente mediante IaC con Terraform, que permita operar microservicios de forma fiable, segura, eficiente y reproducible en un entorno empresarial real, siguiendo las mejores prácticas de la industria y los estándares modernos.

A nivel más atómico, los objetivos específicos del trabajo son los siguientes:

1. Diseñar la arquitectura de red del clúster AKS dentro de la topología *Hub & Spoke* (VNETs, Subnets, NSG, UDR, Firewall Network Rules, Firewall Application Rules).
2. Implementar el clúster AKS mediante módulos Terraform reutilizables con versionado semántico.
3. Configurar el modelo de red de AKS optimizando el consumo de direcciones IP.
4. Integrar los servicios de seguridad e identidad: Entra ID, RBAC, Azure Policy, Workload Identity.
5. Desplegar un ACR privado con Private Endpoints como servicio transversal.
6. Implementar los pipelines de CI/CD con GitHub Actions y conectividad privada (VNet Integration).
7. Implementar el modelo GitOps con Argo CD y el patrón *App of Apps* para la gestión declarativa del clúster.
8. Documentar las decisiones arquitectónicas, configuraciones y procedimientos operativos.

La monitorización avanzada (dashboards, alertas y métricas) y el backup se consideran operaciones de Día 2: la infraestructura necesaria queda preparada —Container Insights habilitado en el módulo del clúster, módulo `aks-backup-extension` desarrollado— pero su configuración completa queda fuera del alcance de este trabajo.

4. METODOLOGÍA

El trabajo se ha desarrollado siguiendo una **metodología iterativa e incremental** basada en principios ágiles, utilizando **Jira** como herramienta para la organización de tareas, agrupadas en épicas, y versionando el desarrollo del proyecto mediante **Git en GitHub** con un flujo de Pull Requests en el que se lanza `terraform plan` automático y `terraform apply` una vez se realiza merge de la Pull Request a main.

El desarrollo se ha dividido en 6 fases: investigación; infraestructura de soporte para AKS; refactorización, implementación y configuración de AKS; configuración de red de la plataforma; observabilidad y operación del entorno; seguridad y compliance. A lo largo del desarrollo del trabajo, se han producido solapamientos entre las distintas fases: una vez consolidada la base conceptual y realizado el análisis inicial de la infraestructura necesaria, la implementación práctica de la plataforma ha seguido un avance paralelo a investigaciones puntuales sobre recursos y/o servicios concretos, y como consecuencia de la naturaleza iterativa del proyecto, las distintas fases que conforman la implementación de la plataforma se han desarrollado de forma paralela, mediante iteraciones de prueba y error, hasta alcanzar un resultado funcional, satisfactorio y reproducible.

Los tres principios metodológicos que han guiado el trabajo han sido los siguientes:

- **Validación progresiva con pruebas documentadas.** Cada avance relevante se ha validado mediante tests y se han dejado documentados los resultados (operatividad del clúster, pull/push de imágenes, GitOps, secretos con CSI, políticas, etc.).
- **Diagnóstico estructurado de incidencias.** Ante problemas complejos se ha aplicado una metodología de diagnóstico de dentro hacia fuera, es decir, desde los elementos más atómicos hasta la apertura de un caso de soporte oficial con Microsoft.
- **Investigación complementaria en entorno libre.** Para acotar un bloqueo durante el desarrollo en el entorno corporativo, se ha replicado el stack (Terraform, Argo CD, Workload Identity, GitHub Actions) en un clúster sobre Raspberry Pi, sin las restricciones del entorno empresarial, permitiendo explorar alternativas y verificar su viabilidad técnica.

La Tabla 1 resume el stack tecnológico del proyecto.

Cuadro 1
Stack tecnológico del proyecto

Categoría	Herramienta / Tecnología
Orquestación	Azure Kubernetes Service (AKS)
IaC	Terraform
Cloud	Microsoft Azure
Identidades	Microsoft Entra ID, Workload Identity (OIDC)
Registro	Azure Container Registry (ACR)
Seguridad de red	Azure Firewall, NSG, AVNM
Gestión de secretos	Azure Key Vault, ESO, CSI Secrets Store
GitOps	Argo CD (App of Apps)
Ingress	Gateway API + NGINX Gateway Fabric
Monitorización	Azure Monitor, Log Analytics, Container Insights
CI/CD	GitHub Actions (workflows reutilizables)
Gestión	Git, GitHub, Jira

5. DISEÑO E IMPLEMENTACIÓN

Todo el código se ha estructurado en tres stacks de despliegue Terraform: `sis-aks-dev` para el clúster AKS, `sis-registry-dev` para el container registry y `sis-identity-dev` para la gestión de identidades en el pipeline de publicación de apps, que consumen una biblioteca común de módulos versionados.

5.1. Módulos Terraform y versionado semántico

El equipo de Sistemas Cloud mantiene un repositorio de módulos reutilizables (`iac-azure-modules`) con un proceso de release automático. Un workflow de GitHub Actions, disparado ante cualquier cambio en ficheros `.tf`, autentica mediante una GitHub App (sin tokens personales), identifica todos los módulos afectados de forma transitiva por el cambio, ejecuta `terraform validate` sobre ellos y, finalmente, incrementa la versión semántica, creando una tag de Git y una release. Los stacks que despliegan los diferentes recursos de la plataforma referencian cada módulo por su tag exacto (p. ej. `kubernetes-cluster?ref=v1.x.y`),

de esta manera se garantiza que todos los despliegues que se realicen en cualquiera de los stacks (`sis-aks-dev`, `sis-registry-dev`, `sis-identity-dev`) son deterministas y fácilmente reproducibles.

En el marco de este TFG se han desarrollado y/o consolidado cuatro módulos:

- **kubernetes-cluster:** despliega un clúster AKS completo y seguro por defecto: clúster privado, RBAC de Entra ID, OIDC issuer, Workload Identity, Key Vault Secrets Provider (CSI), Container Insights, Azure Policy y la asignación automática del rol `AcrPull` al kubelet.
- **kubernetes-cluster-node-pool:** añade node pools adicionales con soporte de autoscaling, taints, labels, zonas de disponibilidad y rotación sin downtime (`temporary_name_for_rotation`).
- **container-registry:** despliega un Azure Container Registry con SKU Premium, acceso público deshabilitado y Private Endpoint hacia la zona `privatelink.azurecr.io`.
- **aks-backup-extension:** despliega la infraestructura completa de backup de AKS (véase Sección 5.8).

5.2. Red del Spoke

El Spoke de AKS se despliega sobre la VNet `vnet-kubernetes-dev`, cuyo address space se asigna de manera dinámica desde un pool de IPAM gestionado por Azure Virtual Network Manager en el Hub, gracias a este enfoque se evitan los solapamientos de rangos de IPs. La VNet se segmenta en subnets dedicadas para cada node pool (sistema, test, plataforma), para los Private Endpoints y para un futuro Private Link Service que conecte el Gateway con un Azure Front Door. Los servidores DNS apuntan al Inbound Endpoint del Private DNS Resolver que hay en el Hub. Este punto es clave para la resolución de recursos privados de la compañía en la red de Azure: sin este Inbound Endpoint, sería imposible que el clúster AKS resolviera el nombre del Azure Container Registry y, como consecuencia, no podría consumir las imágenes almacenadas en él. Algunas de estas zonas privadas son: `privatelink.spaincentral.azmk8s.io`, `privatelink.vaultcore.azure.net`, `privatelink.blob.core.windows.net`, `privatelink.azurecr.io`.

La conectividad y la seguridad de red se gobiernan desde Azure Virtual Network Manager mediante IaC (Fig. 1): una **Connectivity Configuration** dispara el peering Hub↔Spoke al incorporar la VNet a un Network Group; una **Routing Configuration** materializa la UDR por defecto `0.0.0.0/0 → Virtual Appliance` (IP del Azure Firewall), forzando todo el tráfico de salida a través del firewall (`outbound_type = userDefinedRouting`); y las **Security Admin Rules** añaden reglas de seguridad de alto nivel que complementan las NSG.

Adicionalmente, el NSG aplicado sobre los nodos del clúster incluye reglas explícitas de permiso para los health probes del Load Balancer (`168.63.129.16`), Azure Backup y el almacenamiento. Las reglas de salida del Azure Firewall se definen como un Rule Collection Group sobre la Firewall

Policy del Hub e implementan el conjunto de FQDN y reglas L3/L4 requeridos por AKS [8].

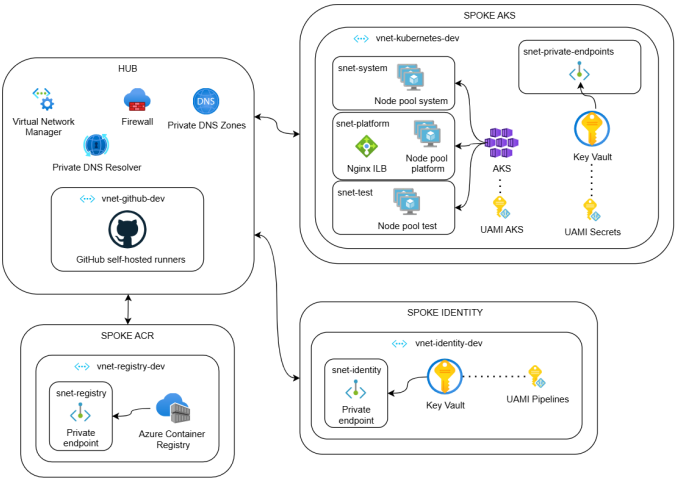


Figura 1. Arquitectura de la plataforma: Hub centralizado con Azure Firewall, AVNM, DNS Resolver y runners de GitHub Actions (vnet-github-dev), y tres Spokes independientes — AKS (con node pools, NGF/ILB, Key Vault y UAMIs de identidad), ACR (con Private Endpoint) e identidades de CI/CD (UAMI de pipelines y Key Vault de GitHub Actions).

5.3. Clúster AKS

El clúster aks-kubernetes-dev se despliega con una configuración de seguridad endurecida y orientada a producción:

- **Clúster privado** (private_cluster_enabled = true): el API Server es accesible únicamente desde la red interna, mediante resolución DNS privada en la zona privatelink.spaincentral.azmk8s.io.
- **Red:** Azure CNI Overlay con data plane y network policy Cilium (eBPF), service_cidr = 10.0.0.0/16, dns_service_ip = 10.0.0.10 y salida de tipo UDR (outbound_type = userDefinedRouting) hacia el firewall.
- **Identidad y RBAC:** RBAC de Azure/Entra ID habilitado, cuentas locales deshabilitadas (local_account_disabled = true), grupo administrador AzGroup.SistemasCloud.ADM y autenticación de operadores vía kubelogin. De esta manera, aunque se intente acceder al API Server desde la red interna, si no se dispone de una cuenta de Entra ID con los permisos necesarios, no es posible autenticarse y, por tanto, acceder al clúster.
- **Workload Identity:** OIDC issuer y Workload Identity están habilitados, de la misma manera que el Key Vault Secrets Provider (CSI) con rotación automática cada hora.
- **Operación:** SKU Standard con auto-upgrade channel patch y node image upgrade channel NodeImage. Image Cleaner mantiene limpio el disco de los nodos eliminando imágenes no utilizadas, y Container Insights (OMS Agent) envía la telemetría al Log Analytics corporativo.

El clúster se organiza en tres node pools diferenciados por carga de trabajo (Tabla 2), todos con discos efímeros y OS SKU Azure Linux.

Cuadro 2
Node pools del clúster AKS

Pool	VM size	Min-Max	Taint / Propósito
system	D4ds_v5	2-10	CriticalAddonsOnly / Sistema
platnp	D4ads_v5	1-10	workload=platform / Argo CD, ESO, NGF
testnp	D4ds_v5	0-5	workload=test / Pruebas (scale-to-zero)

5.4. Registro de contenedores e identidades

El Azure Container Registry se despliega como servicio transversal en su propio Spoke (sis-registry-dev), con acceso público deshabilitado y accesible exclusivamente mediante Private Endpoint (privatelink.azurecr.io). El control de acceso se basa íntegramente en RBAC e identidades administradas:

- A la identidad de **kubelet** se le asigna automáticamente el rol AcrPull sobre el ACR para que pueda consumir las imágenes que sean necesarias para desplegar aplicaciones en el clúster.
- Las **identidades de CI** obtienen AcrPush mediante credenciales federadas (OIDC), sin secretos estáticos. El stack sis-identity-dev define una UAMI id-identity-acr-push-dev federada con SISCLD/central-pipelines, y un Key Vault privado (kv-identity-gha-dev) donde se custodia la clave privada de la GitHub App para realizar actualizaciones automáticas del repositorio GitOps.

5.5. Seguridad, identidad y gestión de secretos

La gestión de secretos se resuelve sin credenciales estáticas combinando dos patrones apoyados en Workload Identity (federación OIDC entre ServiceAccounts de Kubernetes y UAMIs de Azure):

- **External Secrets Operator (ESO):** materializa secretos de Azure Key Vault como objetos Secret de Kubernetes. Su ServiceAccount external-secrets se federa con la UAMI id-kubernetes-secrets-dev, que dispone del rol Key Vault Secrets User sobre los Key Vaults correspondientes.
- **Secrets Store CSI Driver:** monta secretos directamente en el contenedor como ficheros en un volumen efímero, con rotación automática.

En el plano de gobernanza, se asignan a nivel de suscripción **Azure Políticas** en modo Audit (reportan incumplimientos sin bloquear) para los siguientes controles: integridad de imagen (solo imágenes firmadas con Notation); probes obligatorios de readiness/liveness; requests de CPU y memoria obligatorios; límites de recursos diferenciados por tipo de carga (≤ 500 mCPU / 512 MiB para workloads generales; ≤ 1000 m / 1 GiB para namespaces de plataforma como Argo CD, ESO y NGF, mediante asignaciones de política independientes); registros permitidos (crregistrydev.azurecr.io y registros de confianza); y prohibición de escalada de privilegios (allowPrivilegeEscalation). El modo Audit es una decisión deliberada para esta fase de bootstrapping: permite

observar el cumplimiento real antes de endurecer las políticas a modo *Deny* en operaciones de Día 2.

5.6. GitOps con Argo CD (App of Apps)

La operación del clúster se resuelve con un modelo GitOps sobre Argo CD, instalado vía Helm desde Terraform. La gestión se estructura con el patrón App of Apps y dos Applications raíz definidas desde IaC:

- **platform-apps** → repositorio `argo-aks-platform` (`argocd-apps/`): gestiona los componentes de plataforma.
- **developer-apps** → repositorio `argo-aks-apps` (`argocd-apps/`): gestiona las aplicaciones de los equipos de desarrollo.

El árbol de plataforma usa *sync waves* para garantizar el orden de despliegue: Wave 0 instala los CRDs de la Gateway API; Wave 1 despliega NGINX Gateway Fabric y el ExternalSecret del certificado TLS; Wave 2 crea el Gateway compartido; Wave 3 publica las rutas de plataforma, incluida la HTTPRoute de la interfaz web de Argo CD.

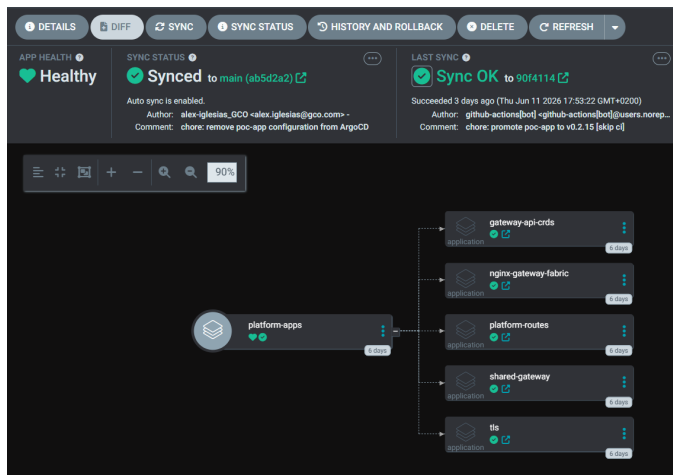


Figura 2. Application `platform-apps` en Argo CD: árbol App of Apps con los componentes de plataforma (Gateway API CRDs, NGINX Gateway Fabric, Gateway compartido, rutas y TLS) en estado Synced/Healthy.

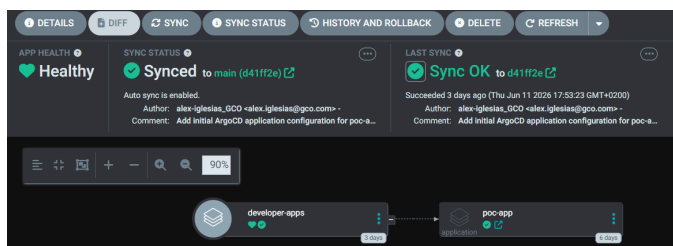


Figura 3. Application `developer-apps` en Argo CD: árbol App of Apps con las aplicaciones de los equipos de desarrollo (`poc-app`) en estado Synced/Healthy.

La autenticación de Argo CD contra los repositorios privados se realiza mediante una GitHub App cuya clave privada se custodia en Key Vault y se sincroniza al clúster con ESO, evitando Personal Access Tokens. El acceso a la UI de Argo CD se federa con Entra ID.

5.7. CI/CD: pipelines reutilizables y entrega determinista

La entrega de aplicaciones se apoya en un repositorio central de workflows reutilizables de GitHub Actions (*central-pipelines*) [12], [13], [14], que es el único poseedor de las credenciales de acceso al ACR. Las aplicaciones de los equipos consumen estos workflows a partir de una plantilla de repositorio (`aks-app-template`) validada con la POC `aks-manifest-lab`. Los runners son GitHub-hosted runners con VNet Integration [15] sobre la red `vnet-github-dev`, lo que les permite alcanzar directamente los servicios privados de Azure —ACR, Key Vault y API Server del clúster— sin exponer endpoints públicos ni requerir IPs de salida fijas.

La plantilla organiza cada aplicación en dos directorios bien diferenciados: `app/src/`, que contiene el código fuente y el Dockerfile, y `app/k8s/`, que contiene los manifests de Kubernetes gestionados con Kustomize (`deployment.yaml`, `service.yaml` y `kustomization.yaml`). El fichero `kustomization.yaml` fija la versión exacta de la imagen mediante el campo `newTag`. Argo CD monitoriza el repositorio Git completo: cuando se produce un nuevo commit, compara el estado declarado en esa revisión con el estado real del clúster y reconcilia cualquier diferencia. Esta separación entre código fuente y manifests permite actualizar el estado del clúster sin reconstruir la imagen —por ejemplo, ante un cambio de configuración en los manifests.

Las aplicaciones que usan la plantilla orquestan el flujo completo mediante el workflow `app-ci.yml`, que se dispara ante cambios en `app/src/**` o en `app/k8s/**` y está compuesto por cinco jobs encadenados:

1. **detect-changes**: determina si los ficheros bajo `app/src/**` han cambiado. Este resultado controla si el job **publish** debe ejecutarse o saltarse.
2. **version**: calcula el siguiente tag semántico en modo *dry-run*, sin crear aún la etiqueta en Git.
3. **publish** (condicional): se ejecuta únicamente si **detect-changes** detectó cambios en el código fuente. Invoca el workflow reutilizable `build-push-dev.yml` de *central-pipelines*, que construye, escanea y publica la imagen.
4. **release**: se ejecuta siempre que **version** haya tenido éxito, independientemente de si **publish** corrió o fue saltado. Si hubo cambios en el código fuente, actualiza el campo `newTag` en `app/k8s/kustomization.yaml`; en cualquier caso crea el commit de release, el tag Git y el GitHub Release correspondiente.
5. **update-gitops-dev**: invoca el workflow reutilizable `update-gitops-dev.yml`, que lee la clave privada de la GitHub App desde el Key Vault de CI/CD (`kv-identity-gha-dev`), genera un token de instalación efímero y actualiza el campo `targetRevision` en el repositorio GitOps.

El flujo de construcción y publicación (`build-push-dev.yml`) ejecuta los siguientes pasos, cuya secuencia es clave porque otorga control efectivo sobre las imágenes que llegan al clúster:

1. **Checkout** del código de la aplicación.
2. **Lint del Dockerfile con Hadolint** (`failure-threshold: warning`): bloquea ante malas prácticas en la definición de la imagen.
3. **Configuración de Docker Buildx** (BuildKit).
4. **Build local de la imagen** (`push: false, load: true`) con caché de capas: la imagen se construye pero aún no se publica.
5. **Escanear de vulnerabilidades con Trivy** (`severity: CRITICAL,HIGH, exit-code: 1`): detiene el pipeline si la imagen contiene CVEs críticas o altas con corrección disponible.
6. **Login en Azure por OIDC** (`azure/login`): autenticación federada sin contraseñas almacenadas.
7. **Login en ACR** (`az acr login`) reutilizando la sesión OIDC.
8. **Push de la imagen y captura del digest**: se etiqueta y publica con el tag de versión y como `latest`, y se extrae el digest `sha256`: de contenido.

El hecho de que el escaneo y el lint se ejecuten *antes* del push implica que ninguna imagen con vulnerabilidades críticas o con un Dockerfile defectuoso llega al registro corporativo.

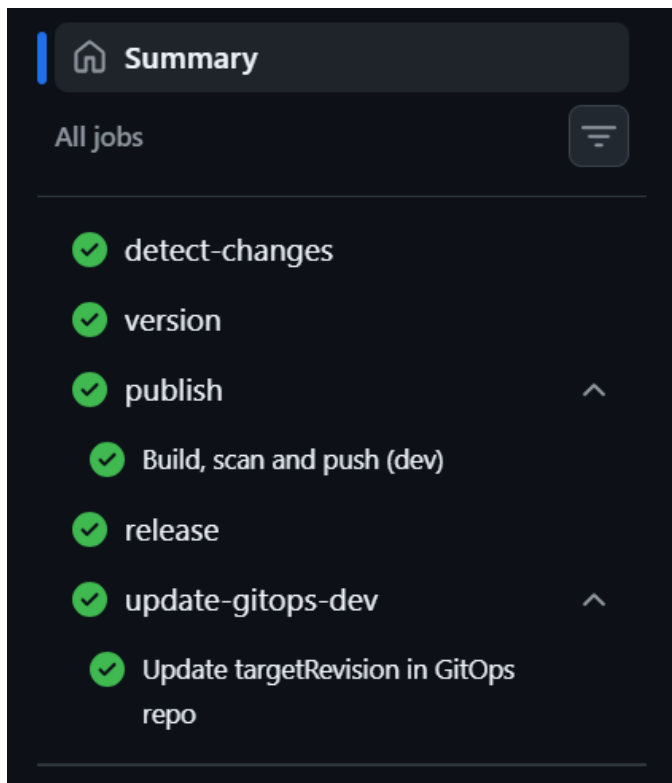


Figura 4. Jobs del pipeline `poc-app-ci` completados con éxito: *detect-changes*, *version*, *publish* (build/scan/push), *release* y *update-gitops-dev*.

El flujo de actualización de GitOps (`update-gitops-dev.yml`) cierra el ciclo: lee la clave privada de la GitHub App desde Key Vault, construye y firma un JWT en tiempo de ejecución [16], [17] para obtener un token de instalación de corta duración (1h), clona el repositorio GitOps y actualiza el campo `targetRevision` de la Application de Argo CD a la tag exacta recién

generada (con marca `[skip ci]` para evitar bucles). En paralelo, el job `release` actualiza el campo `newTag` del `kustomization.yml` de los manifiestos.

Esta doble fijación garantiza un estado determinista (la versión desplegada es siempre conocida y explícita) y un rollback trivial: basta con cambiar el `targetRevision` a la tag anterior para que Argo CD reconcilie el estado deseado.

La estrategia prevista para el despliegue a producción consiste en escanear nuevamente la imagen ya publicada en el ACR de desarrollo e importarla por digest —no por tag mutable— al ACR de producción, garantizando que la imagen que llega a producción es bit a bit idéntica a la validada en el entorno de desarrollo, sin necesidad de reconstruirla.

5.8. Exposición de servicios, observabilidad y backup

Exposición de servicios. El punto de entrada al clúster se diseña con Gateway API y NGINX Gateway Fabric (NGF) [11], que despliega un Internal Load Balancer (ILB) privado. Se define un Gateway compartido con listeners HTTP/80 y HTTPS/443 que admite rutas de todos los namespaces; el TLS se termina en el Gateway con un certificado *wildcard* sincronizado desde Key Vault por ESO (formato PKCS12 → PEM), sin cert-manager. Este patrón está afectado por el incidente de AVNM descrito en la Sección 6.

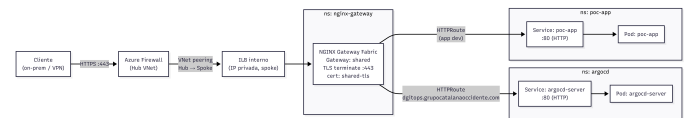


Figura 5. Flujo de tráfico entrante al clúster: cliente → Azure Firewall (Hub) → ILB privado (spoke) → NGINX Gateway Fabric (TLS terminate) → HTTPRoute → Service → pod.

Observabilidad. El clúster integra Container Insights (OMS Agent con autenticación gestionada) contra el Log Analytics corporativo ya existente, y el módulo de clúster incorpora una Alert Processing Rule para enrutar alertas Sev0–Sev2 a un action group. La configuración de métricas y dashboards avanzados se aborda como versión mínima viable, por tratarse de una operación de Día 2.

Backup. Se ha desarrollado el módulo `aks-backup-extension` [18], que despliega la infraestructura completa de backup: cuenta de almacenamiento privada de staging, Backup Vault (OperationalStore, redundancia local), la extensión `Microsoft.DataProtection.Kubernetes` en el clúster, el binding de Trusted Access (rol `backup-operator`), todas las asignaciones de RBAC necesarias, la política de retención (copia diaria a las 02:00 UTC, retención de 30 días) y la instancia de backup (todos los namespaces, con snapshots de volúmenes). El flag `cluster_bindings_enabled` desacopla los recursos compartidos de los bindings específicos del clúster, facilitando despliegues por etapas. La infraestructura queda preparada y documentada como base para su activación en operaciones de Día 2.

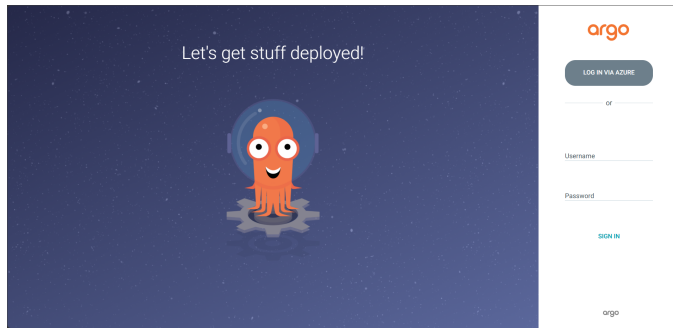


Figura 6. Acceso HTTPS a la interfaz web de Argo CD a través de NGINX Gateway Fabric con certificado *wildcard* sincronizado desde Key Vault, tras la resolución del bug de AVNM.

6. RESULTADOS Y VALIDACIÓN

6.1. Validaciones funcionales documentadas

Cada componente crítico se validó mediante pruebas técnicas cuyos resultados detallados —capturas, logs y comandos ejecutados— se recogen en los documentos de testing del proyecto, accesibles desde el índice de documentación. A continuación se resume el resultado de cada validación:

- **Operatividad del clúster y red interna:** clúster provisionado y operativo, conectividad confirmada desde una VM del Spoke, y resolución DNS interna correcta (CoreDNS, `kubernetes.default` y servicios ClusterIP).
- **Cadena de suministro de imágenes:** push automatizado de imágenes al ACR desde GitHub Actions con OIDC, y pull desde el clúster con programación en el node pool `testnp` respetando taints y tolerations.
- **GitOps:** Argo CD autenticándose contra repositorios privados —primero con PAT y después con GitHub App— y reconciliando manifiestos Kustomize hasta el estado `Synced/Healthy`.
- **Gestión de secretos:** validación de ESO (ClusterSecretStore y ExternalSecret en estado `SecretSynced`) y del CSI Secrets Store, incluyendo el montaje de un secreto de Key Vault como fichero en el contenedor y su rotación automática tras crear una nueva versión.
- **Gobernanza:** verificación de la aplicación de las Azure Políticas sobre el clúster.
- **SSO con Entra ID:** autenticación de usuarios en la UI de Argo CD federada con Entra ID mediante OIDC nativo, verificando el `claim groups` y la asignación correcta de rol `org-admin` al grupo de SistemasCloud.
- **Exposición de servicios:** acceso HTTPS a la UI de Argo CD mediante NGINX Gateway Fabric e HTTPRoute, con certificado *wildcard* desde Key Vault, tras la resolución del bug de AVNM.
- **Entrega extremo a extremo:** la POC `poc-app` (repositorio `aks-manifest-lab`) recorre con éxito todo el flujo CI/CD y queda desplegada y sincronizada por Argo CD en la versión `v0.2.16`.

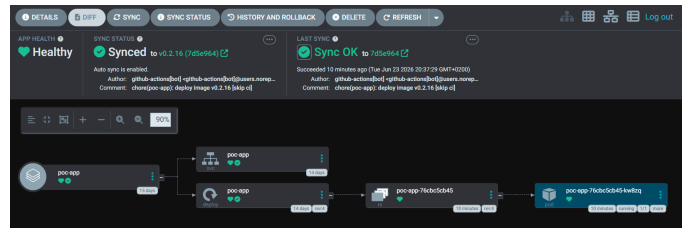


Figura 7. Application `poc-app` en Argo CD sincronizada a la versión `v0.2.16` en estado `Synced/Healthy`, resultado del pipeline de CI/CD extremo a extremo.

El flujo completo observado fue: push de código al repositorio `aks-manifest-lab` → ejecución del pipeline `poc-app-ci` en GitHub Actions (detección de cambios, versionado semántico, escaneo y publicación de imagen firmada en ACR) → commit automático sobre el repositorio de manifiestos actualizando el tag de imagen → reconciliación por Argo CD en un intervalo de aproximadamente 2–3 minutos → pod actualizado en el clúster con la nueva versión, en estado `Synced/Healthy`.

6.2. Incidencia de networking: bug de AVNM

La principal incidencia del proyecto afectó al patrón de ingress: los backend pools del ILB marcaban los nodos como inaccesibles (DOWN) pese a que los servicios de Kubernetes estaban saludables. Se siguió un **diagnóstico estructurado de dentro hacia fuera**: (1) verificación de las UDR para descartar tráfico asimétrico; (2) revisión de las NSG; (3) revisión de las reglas del firewall; (4) revisión de los manifiestos del Service de tipo LoadBalancer; (5) petición `curl` desde una VM de la misma VNet al endpoint `/healthz` del nodo, que devolvió HTTP 200 OK, confirmando que el servicio era accesible desde dentro de la red.



Figura 8. Respuesta HTTP 200 OK al endpoint `/healthz` del nodo desde una VM del Spoke, confirmando que el servicio era accesible desde la red interna y descartando un problema de configuración propio.

Descartada toda la configuración propia, se abrió un **caso de soporte oficial con Microsoft**. El análisis conjunto —y el posterior Root Cause Analysis (RCA) de Microsoft— determinó que se trataba de un **bug de producto de AVNM**: una Security Admin Rule de denegación de todo el tráfico inbound desde Internet (`deny-all-inbound-internet`) permanecía aplicada de forma persistente sobre las NICs de los nodos —bloqueando los health probes del ILB, cuyo origen (`168.63.129.16`) se clasifica como tráfico de Internet— incluso después de eliminarla del control plane.

Rule name	Network manager	Action	Direction	Protocol	Source
deny-all-inbound-inter...	vnm-connectivi...	Deny	Inbound	Any	Internet

Figura 9. Security Admin Rule `deny-all-inbound-internet` persistente sobre las NICs de los nodos, detectada durante el diagnóstico con el soporte de Microsoft. La regla permanecía activa en el plano de datos pese a haber sido eliminada del control plane de AVNM.

La causa raíz, según el RCA, fue un **conflicto de scopes** entre dos Network Managers (`vnm-connectivity-dev` y `vnm-connectivity-prod`): tras un cambio de scope, el snapshot de despliegue de uno de ellos quedó apuntando al mismo management group (`mg-root`) que el otro. Al no poder determinar la precedencia, AVNM dejó de aplicar reglas de cualquiera de los dos (comportamiento “por diseño” ante conflicto), de modo que el goal state de la VNet nunca se actualizaba y la regla deny previa seguía vigente. El error no se propagaba al estado por VNet visible para el cliente, lo que lo hacía indetectable por los canales de monitorización disponibles. Microsoft mitigó el problema actualizando el scope del snapshot, tras lo cual la regla obsoleta desapareció y los health probes del ILB se recuperaron.

Este hallazgo evidencia la complejidad de operar servicios gestionados en entornos corporativos, donde un componente de gobernanza centralizada puede introducir comportamientos no documentados e invisibles para el usuario.

6.3. Exploración del stack en entorno libre (Raspberry Pi)

Con el incidente de AVNM abierto y la exposición de servicios bloqueada en el entorno corporativo, se realizó una exploración técnica en un entorno libre para validar el comportamiento de NGINX Gateway Fabric de forma aislada y descartar un problema propio de la configuración, montando un clúster Kubernetes *bare-metal* sobre Raspberry Pi sin dependencias de Azure.

Las principales diferencias respecto al entorno corporativo fueron:

- **Gestión de secretos:** al no disponer de Azure Key Vault, se desplegó un gestor de secretos dentro del propio clúster, manteniendo el principio de no usar credenciales estáticas.
- **Exposición de servicios:** NGINX Gateway Fabric se configuró con un Service de tipo `NodePort`. Un servidor NGINX externo al clúster actuaba como proxy inverso, redirigiendo las peticiones HTTP entrantes al puerto expuesto por el `NodePort`.
- **DNS y acceso:** la propia Raspberry Pi actuaba como servidor DNS de la *tailnet* gestionada con Tailscale, resolviendo los hostnames de los servicios mediante CoreDNS. Este mecanismo permitió acceder a la UI de Argo CD por nombre desde cualquier nodo de la red privada.
- **Acceso al clúster desde runners:** Tailscale se utilizó también para permitir la ejecución de `terraform plan/apply` contra el API Server privado del clúster.

El objetivo de este entorno fue doble: explorar alternativas de exposición de servicios durante el bloqueo, y descartar

que el problema del ILB se debiera a la configuración de NGINX Gateway Fabric o de las HTTPRoutes. La UI de Argo CD se expuso correctamente con el stack Gateway API + NGF + NodePort, confirmando que NGF funcionaba como se esperaba y que el fallo en el entorno corporativo era ajeno a la configuración del Gateway.

7. ANÁLISIS DE COSTES

El coste real acumulado de los tres *stacks* de infraestructura Azure desplegados durante el TFG, en el período febrero–junio de 2026 (aproximadamente 5 meses), asciende a **2.350,45 €** (Tabla 3), con una previsión de cierre de 2.442,66 € y un coste mensual medio de ≈ 470 €/mes.

Cuadro 3
Coste acumulado por stack (feb–jun 2026)

Stack	Real (€)	Previsión (€)
sis-aks-dev	2.136,28	2.223,29
sis-registry-dev	210,73	215,93
sis-identity-dev	3,44	3,44
Total	2.350,45	2.442,66

Dentro de **sis-aks-dev** (91 % del total), las máquinas virtuales de los node pools representan el 69 % del gasto (1.481,24 €), seguidas de Microsoft Defender for Cloud (257,74 €, servicio transversal de la suscripción no atribuible directamente al proyecto) y la tarifa del plano de control AKS en tier *Standard* (253,83 €, ≈ 50 €/mes). El node pool `testnp`, configurado con escala a cero (`min_count = 0`), ha reducido el coste en los períodos sin carga de pruebas.

En **sis-registry-dev** (9 % del total), el ACR en SKU *Premium* supone 184,95 €. El SKU Premium es necesario para deshabilitar el acceso público y usar exclusivamente *Private Endpoint*.

El coste de **sis-identity-dev** (3,44 €) es marginal y corresponde a las operaciones del Key Vault de GitHub Actions.

Cabe recalcar que no se imputan los servicios transversales preexistentes de la organización (Hub VNet, Azure Firewall, Log Analytics, Private DNS), debido a que son compartidos con otras cargas de trabajo y existían con anterioridad al proyecto.

8. CONCLUSIONES Y TRABAJO FUTURO

El trabajo demuestra la viabilidad de implementar una plataforma *cloud-native* completa sobre AKS empleando exclusivamente Infraestructura como Código en un entorno empresarial real. El stack resultante —clúster privado con Azure CNI Overlay y Cilium, tres node pools diferenciados, tráfico de salida controlado por firewall, identidad federada sin secretos estáticos, ACR privado, gobernanza con Azure Policy y operación GitOps con Argo CD y el patrón App of Apps— es coherente con las prácticas actuales de la industria y plenamente reproducible.

Los resultados con mayor impacto organizacional son la **plantilla de repositorio para equipos de desarrollo** y los **pipelines reutilizables**: permiten que los equipos desplieguen y gestionen sus aplicaciones de forma autónoma y declarativa, con garantías de trazabilidad, rollback sencillo

y seguridad de la cadena de suministro (control de imágenes mediante lint y escaneo previo al push).

La principal limitación identificada fue la dependencia de la resolución del bug de AVNM para completar la exposición de servicios mediante ILB. El diagnóstico estructurado y la investigación complementaria en un entorno libre permitieron acotar el problema con precisión y confirmar que el diseño era correcto.

Las principales lecciones técnicas extraídas del proyecto son:

- **Gobernanza centralizada como primera clase:** los componentes como AVNM y Azure Policy deben modelarse y validarse desde el inicio del diseño; el conflicto de scopes de AVNM fue indetectable por los canales habituales de monitorización porque actuaba por debajo de la visibilidad del plano de datos del cliente.
- **Audit antes de Deny:** el modo `Audit` previo al modo `Deny` en Azure Policy no es solo una precaución, sino una práctica que en este proyecto descubrió restricciones demasiado agresivas antes de que bloquearan el entorno productivo.
- **Modularización IaC desde el primer día:** el coste inicial de estructurar el código en módulos reutilizables permitirá extender la plataforma a los entornos de preproducción y producción reutilizando exactamente los mismos módulos sin reescribir código.

Como **trabajo futuro** (operaciones de Día 2) se contempla: la activación del backup —cuya infraestructura queda ya preparada con el módulo `aks-backup-extension`; la monitorización avanzada (métricas, dashboards y alertas), para la que la integración de logs en AKS está activa pero los dashboards y alertas quedan pendientes; el endurecimiento de las Azure Policies del modo `Audit` al modo `Deny`; y la extensión de la plataforma a los entornos de preproducción y producción, reutilizando los módulos y pipelines ya validados.

9. USO DE LA IA GENERATIVA

Durante el desarrollo del TFG se ha utilizado IA generativa —principalmente GitHub Copilot integrado en Visual Studio Code— como herramienta complementaria en cuatro ámbitos: (1) asistencia en la redacción y reformulación de documentación técnica y de la memoria; (2) revisión de código Terraform mediante autocompletado y sugerencias; (3) exploración y comprensión de tecnologías y documentación de servicios de Azure; y (4) interpretación de mensajes de error durante la implementación.

En todos los casos, la IA se ha empleado como apoyo para consolidar conocimientos, sin sustituir la documentación oficial y en ningún caso como herramienta para tomar decisiones. Todo el contenido generado con su ayuda ha sido revisado, adaptado y contrastado con fuentes de información antes de incorporarse al trabajo.

ACKNOWLEDGMENTS

El autor agradece al tutor de empresa, Joel Cantero Priego, y al equipo del departamento de Sistemas Cloud de GCO,

Tecnología y Servicios, su acompañamiento técnico y la oportunidad de desarrollar el proyecto en un entorno empresarial real; así como al tutor de universidad, David Gutiérrez Yana, por su orientación a lo largo del trabajo.

REFERENCIAS

- [1] Cloud Native Computing Foundation, “Kubernetes Project — CNCF,” <https://www.cncf.io/projects/kubernetes/>, 2024, [Online; consultado en 2025].
- [2] Microsoft, “Azure Kubernetes Service (AKS) Documentation,” <https://learn.microsoft.com/en-us/azure/aks/>, 2025, [Online; consultado en 2025].
- [3] —, “Azure CNI Overlay Networking for AKS,” <https://learn.microsoft.com/en-us/azure/aks/concepts-network-azure-cni-overlay>, 2025, [Online; consultado en 2025].
- [4] HashiCorp, “Terraform AzureRM Provider Documentation,” <https://registry.terraform.io/providers/hashicorp/azurermlatest/docs>, 2025, [Online; consultado en 2025].
- [5] Microsoft, “Hub-Spoke Network Topology in Azure,” <https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/hybrid-networking/hub-spoke>, 2025, [Online; consultado en 2025].
- [6] —, “Cloud Adoption Framework for Azure,” <https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/>, 2025, [Online; consultado en 2025].
- [7] —, “Azure Well-Architected Framework,” <https://learn.microsoft.com/en-us/azure/well-architected/>, 2025, [Online; consultado en 2025].
- [8] —, “Outbound Network and FQDN Rules for AKS Clusters,” <https://learn.microsoft.com/en-us/azure/aks/outbound-rules-control-egress>, 2025, [Online; consultado en 2025].
- [9] Argo Project, “Argo CD Documentation — App of Apps Pattern,” <https://argo-cd.readthedocs.io/>, 2025, [Online; consultado en 2025].
- [10] Kubernetes SIG-Network, “Kubernetes Gateway API Documentation,” <https://gateway-api.sigs.k8s.io/>, 2025, [Online; consultado en 2025].
- [11] NGINX, “NGINX Gateway Fabric Documentation,” <https://docs.nginx.com/nginx-gateway-fabric/>, 2025, [Online; consultado en 2025].
- [12] GitHub, “GitHub Actions Documentation,” <https://docs.github.com/actions>, 2025, [Online; consultado en 2025].
- [13] —, “Reusing Workflow Configurations,” <https://docs.github.com/en/actions/reference/workflows-and-actions/reusing-workflow-configurations>, 2025, [Online; consultado en 2025].
- [14] —, “OIDC with Reusable Workflows,” <https://docs.github.com/en/actions/how-to-secure-your-work/security-harden-deployments/oidc-with-reusable-workflows>, 2025, [Online; consultado en 2025].
- [15] —, “About Azure Private Networking for GitHub-Hosted Runners,” <https://docs.github.com/en/enterprise-cloud@latest/admin/configuring-settings/configuring-private-networking-for-hosted-compute-products/about-azure-private-networking-for-github-hosted-runners-in-your-enterprise>, 2025, [Online; consultado en 2025].
- [16] —, “Generating a JSON Web Token (JWT) for a GitHub App,” <https://docs.github.com/en/apps/creating-github-apps/authenticating-with-a-github-app/generating-a-json-web-token-jwt-for-a-github-app>, 2025, [Online; consultado en 2025].
- [17] —, “actions/create-github-app-token — Getting Started,” <https://deepwiki.com/actions/create-github-app-token/2-getting-started>, 2025, [Online; consultado en 2025].
- [18] Microsoft, “Azure Kubernetes Service Backup Overview,” <https://learn.microsoft.com/en-us/azure/backup/azure-kubernetes-service-backup-overview>, 2025, [Online; consultado en 2025].