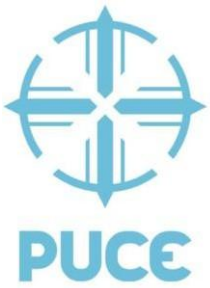


PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

ESCUELA HÁBITAT, INGENIO Y CREATIVIDAD

TRABAJO DE INTEGRACIÓN CURRICULAR PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN

PROCESOS DE DESPLIEGUE Y GESTIÓN DE ENTORNOS EN LA NUBE MEDIANTE INFRAESTRUCTURA COMO CÓDIGO (IAC) EN LA EMPRESA ECUALATINO

AUTOR: LINKIN DANNY FARINANGO PAVÓN

TUTOR: GALO HERNÁN PUETATE HUERA

IBARRA – ECUADOR

JULIO, 2025

CERTIFICACIÓN TUTOR

En mi calidad de Tutor del Trabajo de titulación titulado “Procesos de despliegue y gestión de entornos en la nube mediante infraestructura como código (IaC) en la empresa Ecualatino, presentado por el estudiante Linkin Danny Farinango Pavón, con cédula de ciudadanía N°, 100429075-3 para obtener el Título de Ingeniero en Tecnologías de la Información.

Certifico que el trabajo cumple con todos los parámetros establecidos, mediante el cual el estudiante demuestra el desarrollo de competencias en el campo de conocimiento de su profesión con un nivel de argumentación coherente, para ser sometido a la evaluación por parte de los lectores.

Adicionalmente, se adjunta el certificado de porcentaje de originalidad de TURNITIN.


14/8/25, 9:05 Turnitin - Informe de Originalidad - PROCESOS DE DESPLIEGUE Y GESTIÓN DE ENTORNOS EN LA NUBE MEDIANTE INFRA...

Turnitin Informe de Originalidad

Procesado el: 14-ago-2025 09:03 -05
Identificador: 2729543884
Número de palabras: 18845
Entregado: 1

Índice de similitud 5%

Similitud según fuente
Fuentes de Internet 4%
Publicaciones: 1%
Trabajos del estudiante: 3%

PROCESOS DE DESPLIEGUE Y GESTIÓN DE ENTORNOS EN LA NUBE MEDIANTE INFRAESTRUCTURA COMO CÓDIGO (IAC) EN LA EMPRESA ECUALATINO Por LINKIN DANNY FARINANGO PAVON

Coincidencia del < 1% (Internet desde 04-oct-2022)
https://dspace.pucesi.edu.ec/bitstream/11010/538/1/TESIS%20FINAL.pdf

Coincidencia del < 1% (Internet desde 09-jun-2023)
https://dspace.pucesi.edu.ec/bitstream/11010/770/1/TESIS%20RAMS%c3%89S%20TOBO.pdf

Coincidencia del < 1% (Internet desde 16-jun-2025)
http://www.veranoregional.org/issn/2023.pdf

Coincidencia del < 1% (trabajos de los estudiantes desde 07-ene-2025)
Submitted to Universidad Europea Miguel de Cervantes on 2025-01-07

Galo Hernán Puetate Huera
Firmado digitalmente por Galo Hernán Puetate Huera
Fecha: 2025.08.14 10:40:49 -05'00'

(f):_____
Mgs. Galo Hernán Puetate Huera
TUTOR DE TRABAJO
C.C.: 0401375787

PÁGINA DE APROBACIÓN DEL TRIBUNAL

El tribunal examinador, aprueba el presente trabajo en nombre de la Pontificia Universidad Católica del Ecuador Ibarra:

Galo Hernán Puetate Huera
Firmado digitalmente por Galo Hernán Puetate Huera
Fecha: 2025.08.14 10:42:40 -05'00'

(f):_____

Mgs. Galo Hernán Puetate Huera
C.C.: 0401375787

Álvaro Mauricio Cevallos Ramirez
Firmado digitalmente por Álvaro Mauricio Cevallos Ramirez
Fecha: 2025.08.14 11:48:13 -05'00'

(f):……

Mgs. Alvaro Mauricio Cevallos Ramirez.
C.C.: 1002494019

RICARDO PATRICIO RUIZ QUIRANZA
Firmado digitalmente por RICARDO PATRICIO RUIZ QUIRANZA

(f):……

Mgs. Ruiz Quiranza Ricardo Patricio

C.C.: 1002836524

ACTA DE CESIÓN DE DERECHOS

Yo, Linkin Danny Farinango Pavón, declaro conocer y aceptar la disposición del Art. 165 del Código Orgánico de Economía Social de los Conocimientos, Creatividad e Innovación, que manifiesta textualmente: “Se reconoce facultad de los autores y demás titulares de derechos de disponer de sus derechos o autorizar las utilizaciones de sus obras o prestaciones a título gratuito y oneroso, según las condiciones que determinen. Esta facultad podrá ejercerse mediante licencias libres, abiertas y otros modelos alternativos de licenciamiento o la renuncia”.

Ibarra, julio de 2025

Danny Farinango

Firmado digitalmente por Danny Farinango
Fecha: 2025.08.14 12:35:11 -05'00'

(f):_____

Linkin Danny Farinango Pavón

C.C.: 100429075-3

AUTORIA

Yo, Linkin Danny Farinango Pavón, portador de la cedula de ciudadanía N° 100429075-3, declaro que el presente trabajo de investigación es de total responsabilidad de la autor@, y eximo expresamente a la Pontificia Universidad Católica del Ecuador Ibarra de posibles reclamos o acciones legales.

Ibarra, julio de 2025

Danny Farinango
Firmado digitalmente por Danny Farinango
Fecha: 2025.08.14 12:35:30 -05'00'

(f):_____

Linkin Danny Farinango Pavón

C.C.: 100429075-3

DEDICATORIA

A mis padres, por su amor incondicional, apoyo constante y enseñanzas de vida que me han guiado en cada paso de este camino.

A mi familia, por su paciencia, comprensión y palabras de aliento incluso en los momentos más difíciles.

A Dios, por darme la fuerza y perseverancia para superar cada desafío.

Danny

AGRADECIMIENTOS

Agradezco profundamente a la Pontificia Universidad Católica del Ecuador y a la Escuela de Hábitat, Ingenio y Creatividad por brindarme la formación académica que me ha permitido alcanzar esta meta.

A la empresa Ecualatino, por su colaboración y apertura para aplicar esta propuesta.

A mis compañeros, amigos y docentes, quienes contribuyeron con sus conocimientos, apoyo y consejos a lo largo de mi carrera.

Y, sobre todo, a mi familia, por creer siempre en mí y ser mi mayor inspiración.

Danny

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

La presente investigación aborda la optimización de la gestión de infraestructuras tecnológicas en la nube mediante la aplicación de Infraestructura como Código (IaC) en la empresa ecuatoriana Ecualatino. En el contexto actual de transformación digital, la adopción de soluciones cloud se ha convertido en una estrategia clave para mejorar la competitividad empresarial, reducir costos operativos y escalar servicios de TI. No obstante, la gestión manual de estos entornos presenta retos como errores humanos, falta de estandarización y tiempos prolongados de recuperación. Con el objetivo de superar estas limitaciones, se implementó IaC utilizando herramientas como Terraform y Azure, dentro de una arquitectura modular automatizada que abarca redes, máquinas virtuales y bases de datos. La metodología empleada integró prácticas DevOps y pipelines CI/CD, permitiendo mejorar la trazabilidad de cambios, estandarizar procesos y aumentar la eficiencia operativa. Los resultados obtenidos evidencian que IaC aporta significativamente a la automatización, consistencia y seguridad de la infraestructura tecnológica en la nube. La solución implementada demostró su eficacia durante pruebas de recuperación ante fallos, logrando restaurar el sistema en un promedio de 20 minutos. Además, se validó la capacidad de escalar y replicar entornos de manera ágil y confiable.

Palabras clave: infraestructura como código, automatización, nube, Terraform, Azure, DevOps, CI/CD

ABSTRACT

This research addresses the optimization of cloud-based technological infrastructure management through the application of Infrastructure as Code (IaC) at the Ecuadorian company Ecualatino. In the current context of digital transformation, the adoption of cloud solutions has become a key strategy for improving business competitiveness, reducing operating costs, and scaling IT services. However, manual management of these environments presents challenges such as human error, lack of standardization, and long recovery times. To overcome these limitations, IaC was implemented using tools such as Terraform and Azure, within an automated modular architecture that encompasses networks, virtual machines, and databases. The methodology employed integrated DevOps practices and CI/CD pipelines, allowing for improved change traceability, standardized processes, and increased operational efficiency. The results obtained demonstrate that IaC significantly contributes to the automation, consistency, and security of cloud-based technological infrastructure. The implemented solution demonstrated its effectiveness during disaster recovery tests, managing to restore the system in an average of 20 minutes. In addition, the ability to scale and replicate environments in an agile and reliable manner was validated.

Keywords: Infrastructure as Code, automation, cloud, Terraform, Azure, DevOps, CI/CD

INTRODUCCIÓN

En el actual entorno empresarial, caracterizado por una rápida evolución tecnológica, las organizaciones enfrentan el desafío de mantener infraestructuras de TI ágiles, seguras y escalables que respondan eficientemente a las demandas del mercado. La empresa ecuatoriana Ecualatino, dedicada al desarrollo de software, se encuentra en un proceso de modernización de sus entornos tecnológicos a través de soluciones basadas en la nube.

Ecualatino ofrece aplicaciones web y servicios en la nube para clientes del sector financiero y comercial. Su infraestructura tecnológica actual se basa en máquinas virtuales, bases de datos SQL y servicios de red gestionados de manera manual, lo que genera varios desafíos operativos. Entre las principales limitaciones se encuentran: despliegues lentos y propensos a errores humanos, falta de estandarización en la configuración de entornos, dificultades para replicar ambientes de desarrollo y pruebas, y ausencia de un plan formal de recuperación ante desastres. Estas limitaciones impactan directamente en la capacidad de la empresa para mantener la continuidad de sus servicios, incrementar su competitividad y garantizar la seguridad de los datos de sus clientes. Resolver estos problemas mediante la automatización de la infraestructura no solo permite reducir errores y tiempos de despliegue, sino también asegurar que los entornos sean consistentes, reproducibles y resilientes frente a fallos o incidentes de seguridad.

En este contexto, la Infraestructura como Código (IaC) surge como una alternativa innovadora que permite definir, desplegar y administrar entornos de manera automática mediante archivos de configuración. Para abordar los desafíos presentes en la gestión de entornos en la nube, esta investigación planteó como objetivo general analizar los procesos de despliegue y administración de infraestructura mediante Infraestructura como Código (IaC) en la empresa ecuatoriana Ecualatino, con el fin de identificar oportunidades de mejora y optimizar su implementación.

Los objetivos específicos de este estudio fueron: (1) examinar la infraestructura actual de Ecualatino, evaluando su enfoque en el despliegue y gestión de entornos en la nube; (2) investigar las herramientas y metodologías de IaC disponibles en el mercado y analizar su aplicabilidad en el contexto de la empresa; y (3) comparar las mejores prácticas de despliegue en la nube mediante IaC con los procesos operativos actuales de Ecualatino.

Este estudio constituye una contribución relevante al campo de la gestión de infraestructura tecnológica en entornos cloud, al proporcionar un análisis detallado sobre la implementación de IaC en empresas de desarrollo de software. La investigación no solo mejora la eficiencia operativa de Ecualatino, sino que también ofrece un marco de referencia útil para otras organizaciones que enfrentan retos similares en cuanto a automatización y administración de infraestructura.

Los hallazgos obtenidos evidencian avances en la eficiencia, seguridad y escalabilidad de los entornos tecnológicos mediante el uso de IaC, respaldados por recomendaciones basadas en prácticas exitosas. El trabajo está estructurado en tres capítulos: el primero aborda el estado del arte sobre IaC, incluyendo su evolución, características y aplicaciones; el segundo expone la metodología empleada, detallando las técnicas y herramientas utilizadas; y el tercero presenta los resultados y conclusiones, identificando mejoras en la gestión de la infraestructura cloud y proponiendo estrategias para optimizar su implementación.

CAPÍTULO I
ESTADO DEL ARTE

1.1. Marco teórico

1.1.1. Definición y principios fundamentales de IaC

La Infraestructura como Código (IaC) es una metodología que permite la gestión y aprovisionamiento de infraestructura de TI mediante código, eliminando la necesidad de configuraciones manuales. Según Red Hat(2022), "IaC permite gestionar y preparar la infraestructura a través del código, en lugar de hacerlo mediante procesos manuales" (párr.1). Este enfoque facilita la automatización, la reproducibilidad y la escalabilidad de los entornos tecnológicos.

Los principios fundamentales de IaC incluyen:

- **Automatización**: Reducción de la intervención manual en la configuración de infraestructura.
- **Reproducibilidad**: Garantiza que los entornos sean idénticos cada vez que se despliegan.
- **Escalabilidad**: Permite gestionar grandes volúmenes de infraestructura sin aumentar la complejidad operativa.
- **Control de versiones**: Uso de herramientas como Git para rastrear cambios en la infraestructura.

1.1.2. Evolución de IaC y su impacto en la computación en la nube

La Infraestructura como Código (IaC) surge como respuesta a la necesidad de gestionar infraestructuras cada vez más dinámicas y escalables en la era de la computación en la nube. Anteriormente, la configuración manual de servidores y redes era la norma, lo que a menudo derivaba en inconsistencias y errores humanos. Con el advenimiento de la nube, se hizo imprescindible contar con una solución que permitiera despliegues rápidos, seguros y consistentes. Frente a esta necesidad, IaC se consolidó como una práctica esencial para la gestión eficiente de recursos.

Esta evolución ha generado impactos significativos en la operación de las infraestructuras tecnológicas. Entre los principales beneficios que ha traído IaC destacan:

- **Mayor eficiencia operativa:** La automatización del aprovisionamiento y configuración reduce drásticamente el tiempo de despliegue y minimiza la probabilidad de errores.

- **Integración mejorada con DevOps:** IaC facilita la implementación de prácticas de CI/CD y la ejecución de pruebas automatizadas, lo que optimiza el ciclo de desarrollo y despliegue.
- **Optimización de costos:** La capacidad de ajustar dinámicamente los recursos según la demanda permite un uso más racional y económico de la infraestructura.

Según Chinamanagonda (2019), "IaC revoluciona la manera en que las organizaciones gestionan y despliegan su infraestructura de TI, eliminando procesos manuales propensos a errores y asegurando consistencia y confiabilidad en los entornos" (p. 2037).

1.1.3. IaC vs Configuración Manual: Un Cambio de Paradigma

La gestión de infraestructura tradicional ha estado basada en configuraciones manuales, lo que implica una intervención humana constante y un alto riesgo de errores. En contraste, IaC permite la automatización del aprovisionamiento y configuración de infraestructura, reduciendo los riesgos asociados y mejorando la eficiencia operativa.

Según Red Hat (2022), "IaC permite gestionar y preparar la infraestructura a través del código, en lugar de hacerlo mediante procesos manuales" (párr. 1), facilitando la administración de entornos de manera reproducible y escalable. Esta automatización minimiza errores humanos y tiempos de despliegue, lo que resulta en procesos más eficientes.

Los métodos tradicionales de gestión de infraestructura implicaban configuraciones manuales y scripts personalizados, lo que generaba problemas de escalabilidad y mantenimiento.

En la tabla 1 se puede observar algunas diferencias entre estos métodos:

Tabla 1:
Comparación de Aspectos Clave entre Métodos Tradicionales y IaC.

Aspecto	Métodos Tradicionales	Infraestructura como Código (IaC)
Configuración	Manual, propensa a errores	Automática y reproducible
Escalabilidad	Limitada	Alta, adaptable a demanda
Control de versiones	Difícil de rastrear	Integrado con herramientas como Git
Automatización	Parcial, basada en scripts	Completa, basada en código
Seguridad	Depende de configuraciones manuales	Implementada mediante código

De acuerdo con Microsoft (2023), la infraestructura como código (IaC) elimina la necesidad de configuraciones manuales al garantizar la coherencia en los entornos. Esto se logra a través

de la representación de estados deseados utilizando código estructurado y bien documentado en formatos como JSON.

1.1.4. Tipos de enfoques en IaC para la implementación

IaC se puede implementar mediante dos enfoques principales como se muestra en la figura 1:

Figura 1:
Tipos de enfoques en IaC para la implementación

Fuente: Sentrio (2023). Prácticas de Infraestructura como Código (IaC).

El enfoque declarativo es más común en herramientas como Terraform y AWS CloudFormation, mientras que el imperativo se usa en Ansible y scripts personalizados.
Según Red Hat (2022), el enfoque declarativo establece cómo debería ser el estado final de los sistemas, mientras que el enfoque imperativo determina los pasos concretos necesarios para alcanzar esa configuración.

1.1.5. Aplicación de IaC en diferentes sectores empresariales

IaC ha sido adoptado en diversas industrias debido a su capacidad para optimizar procesos, reducir costos y mejorar la seguridad. Algunos sectores clave incluyen:

- **Finanzas:** Permite la automatización de entornos para el análisis de datos y procesamiento de transacciones. Según **Galarza** (2024), "IaC es esencial en entornos de nube y de alta demanda, permitiendo despliegues rápidos, consistentes y escalables" (párr. 2).
- **Manufactura:** Mejora la administración de recursos tecnológicos y la integración de sistemas industriales.
- **Salud:** Facilita la gestión de infraestructuras críticas, garantizando disponibilidad y cumplimiento normativo.
- **E-commerce y tecnología:** Empresas como Amazon, Netflix y Google han adoptado IaC para manejar sus crecientes demandas de infraestructura.

1.1.6. Metodologías Principales para Implementar IaC

Existen varias metodologías para implementar IaC, cada una con enfoques diferentes:

- **Declarativa:** permite definir el estado deseado de la infraestructura, dejando que las herramientas de gestión de configuración lo implementen automáticamente. Esta metodología asegura coherencia, facilita la reproducción de entornos y minimiza errores humanos, ya que los cambios en la infraestructura se especifican de manera abstracta sin necesidad de detallar cada paso de implementación (Sentrio, 2023).
- **Imperativa:** se basa en la ejecución de comandos específicos para configurar la infraestructura de manera precisa. Este método otorga mayor control sobre cada acción realizada, permitiendo que los administradores de sistemas determinen los pasos exactos de configuración. Aunque proporciona flexibilidad, requiere una gestión meticulosa para evitar errores operacionales (Sentrio, 2023).
- **Basada en módulos:** divide la infraestructura en componentes reutilizables que facilitan la gestión, optimizan los despliegues y mejoran la escalabilidad. Esta metodología permite una administración eficiente al estructurar los elementos del sistema en bloques que pueden integrarse de manera flexible dentro de distintos entornos tecnológicos (Moreno, 2022).
- **GitOps:** es un enfoque que emplea Git como la fuente central de información para la administración de la infraestructura. De acuerdo con Moreno (2022), las estrategias basadas en infraestructura como código (IaC) permiten a las organizaciones organizar sus despliegues de manera optimizada, garantizando la coherencia y la capacidad de escalabilidad en sus sistemas tecnológicos.

1.1.7. Tecnologías y Herramientas de IaC

- **Terraform (HashiCorp):** Terraform, desarrollado por HashiCorp, es una herramienta basada en un enfoque declarativo que facilita la definición de infraestructura mediante archivos de configuración escritos en HCL (HashiCorp Configuration Language). De acuerdo con AWS (2024), esta tecnología permite especificar el estado deseado de la infraestructura, asegurando que sea tanto reproducible como escalable.
- **AWS CloudFormation:** Solución nativa de Amazon Web Services que permite definir infraestructura en archivos JSON o YAML. Facilita la gestión y el aprovisionamiento de recursos en AWS.
- **Azure Resource Manager (ARM):** Plataforma de Microsoft Azure que permite organizar, implementar y gestionar recursos mediante plantillas declarativas.
- **Ansible, Puppet y Chef:** Las herramientas enfocadas en la gestión de configuraciones y la automatización de servidores, como Ansible, facilitan estos procesos mediante

scripts en YAML. Según Galarza (2024), esta tecnología resulta especialmente útil para la administración de configuraciones y la implementación de servicios en entornos computacionales.

Cada herramienta de IaC tiene características particulares en cuanto a facilidad de uso, automatización, soporte multi-nube y gestión del ciclo de vida. En la tabla 2 se presentan las diferencias clave:

Tabla 2:
Comparación de herramientas

Herramienta	Facilidad de Uso	Automatización	Soporte Multi-nube	Gestión del Ciclo de Vida
Terraform (HashiCorp)	Alta	Alta	Sí	Integrada (state management)
AWS CloudFormation	Media	Alta	Limitado a AWS	Automática en AWS
Azure Resource Manager (ARM)	Media	Alta	Limitado a Azure	Integrada en Azure
Ansible	Alta	Alta	Sí	Mediante playbooks
Puppet	Media	Moderada	Sí	Gestión de configuraciones
Chef	Media	Alta	Sí	Gestión de configuraciones

En la tabla 3 se presenta una comparación de aspectos clave relacionados con el uso y gestión de herramientas IaC.

Tabla 3:
Comparación de Características Generales

Herramienta	Costo	Plataforma	Configuración	Automatización
Terraform (HashiCorp)	Gratuita	Multi-nube	Declarativa (HCL)	Alta
AWS CloudFormation	Gratuita	AWS	Declarativa (JSON/YAML)	Alta
Azure Resource Manager (ARM)	Gratuita	Azure	Declarativa (JSON)	Alta
Ansible	Gratuita	Multi-nube	Imperativa (YAML)	Alta
Puppet	Pago	Multi-nube	Declarativa/Imperativa	Moderada
Chef	Pago	Multi-nube	Imperativa (Ruby DSL)	Alta

En la tabla 4 refleja una comparación de características técnicas más específicas como seguridad, escalabilidad y redundancia.

Tabla 4:
Comparación de Aspectos Técnicos Avanzados

Herramienta	Seguridad	Disponibilidad	Escalabilidad	Persistencia	Redundancia
Terraform (HashiCorp)	Alta	Alta	Alta	Sí	Sí
AWS CloudFormation	Alta	Alta	Alta	Sí	Sí
Azure Resource Manager (ARM)	Alta	Alta	Alta	Sí	Sí
Ansible	Media	Alta	Alta	Sí	Sí
Puppet	Alta	Alta	Media	Sí	Sí
Chef	Alta	Alta	Alta	Sí	Sí

1.1.8. Gestión de Entornos en la Nube

Los entornos en la nube se dividen en tres categorías principales**,** cada una con un propósito específico dentro del ciclo de vida del desarrollo de software:

- **Entorno de Desarrollo:** es el espacio destinado a la creación y modificación de aplicaciones antes de su implementación. Según Galarza (2024), este entorno posibilita la realización de ajustes sin impactar la infraestructura de producción, lo que favorece la experimentación y la ejecución de pruebas iniciales.
- **Entorno de Pruebas:** es fundamental para verificar la funcionalidad y estabilidad de las aplicaciones antes de su implementación en producción. Según Moreno (2022), este proceso permite identificar posibles errores en la nube antes del despliegue definitivo, lo que contribuye a garantizar la calidad del software.
- **Entorno de Producción:** es el espacio en el que la aplicación está disponible para los usuarios finales, garantizando altos niveles de disponibilidad, seguridad y rendimiento óptimo. Según Microsoft (2023), la administración de estos entornos en la nube exige un monitoreo continuo y la implementación de estrategias de escalabilidad que aseguren la continuidad del servicio.

1.1.9. Desafíos en la gestión de entornos en la nube

La administración de entornos en la nube presenta varios desafíos clave:

- **Seguridad:** en entornos de computación en la nube es esencial para garantizar la protección de datos y una gestión adecuada de accesos. Según HostDime (2024), su implementación requiere la aplicación de controles de acceso, técnicas de cifrado de información y auditorías periódicas con el fin de minimizar posibles vulnerabilidades.

- **Costos:** La gestión eficiente de los costos en la nube representa un desafío para numerosas empresas. Según Insitech (2025), es fundamental que las organizaciones supervisen el consumo de recursos y adapten su infraestructura para evitar gastos innecesarios.
- **Mantenimiento:** El mantenimiento de infraestructura en la nube requiere el uso de herramientas automatizadas para facilitar su actualización y administración. Según Microsoft (2025), este proceso implica la implementación de estrategias como el monitoreo proactivo, la gestión de parches y la automatización de despliegues, garantizando así la estabilidad y eficiencia operativa.
- **Gobernanza:** La gobernanza en entornos de computación en la nube requiere la implementación de políticas y controles que aseguren el cumplimiento de normativas establecidas. Según Microsoft (2025), este proceso implica la evaluación de riesgos y el desarrollo de directrices para la gestión eficiente de la infraestructura.

1.1.10. Integración de IaC en entornos híbridos y multinube

La **Infraestructura como Código (IaC)** facilita la gestión de entornos en la nube, especialmente en configuraciones híbridas y multinube:

- **Entornos híbridos:** incorporan tanto infraestructura local como servicios en la nube para optimizar la gestión tecnológica. De acuerdo con Galarza (2024), la infraestructura como código (IaC) facilita la administración de estos entornos al automatizar configuraciones y permitir la integración con plataformas locales.
- **Entornos multinube:** emplean varios proveedores de nube con el propósito de fortalecer la resiliencia y mejorar la disponibilidad de los servicios. De acuerdo con Insitech (2025), la administración eficiente de estos entornos demanda herramientas de infraestructura como código (IaC) que faciliten la interoperabilidad entre plataformas como AWS, Azure y Google Cloud.

1.1.11. Pipeline de CI/CD con IaC

Un pipeline de CI/CD (Integración Continua/Entrega Continua) es un flujo de trabajo automatizado que permite la integración de cambios en el código, su prueba y despliegue de manera eficiente y repetitiva. Su propósito es reducir errores y acelerar la entrega de software mediante la automatización de pruebas y compilaciones. Según Kissel (2018), la automatización en CI/CD mejora la eficiencia en proyectos ágiles, reduciendo errores y tiempos de entrega.

La incorporación de la Infraestructura como Código (IaC) en los pipelines de CI/CD facilita la automatización del despliegue de infraestructura, garantizando uniformidad y eficiencia en los entornos de desarrollo. De acuerdo con GitLab (2025), estos procesos permiten optimizar la entrega de software mediante la automatización de pruebas y compilaciones, asegurando una implementación más confiable.

1.1.12. Integración con herramientas de CI/CD

Las herramientas más utilizadas para la integración de IaC en pipelines de CI/CD incluyen:

- **Jenkins:** plataforma de automatización diseñada para facilitar la integración y el despliegue continuo en procesos de desarrollo de software. De acuerdo con Digital.ai (2024), esta herramienta optimiza la creación y administración de pipelines de CI/CD gracias a su ecosistema de complementos y su capacidad para distribuir cargas de compilación de manera eficiente.
- **GitHub Actions:** Permite la automatización de flujos de trabajo directamente en GitHub, facilitando la integración de IaC en proyectos de código abierto y empresariales.
- **GitLab CI/CD:** Ofrece una solución integrada para la automatización de pruebas y despliegues, asegurando la calidad del código antes de su implementación.
- **Azure DevOps:** Plataforma de Microsoft que permite la gestión de pipelines de CI/CD con integración nativa de IaC.
- **AWS CodePipeline:** Servicio de AWS que automatiza la entrega de software mediante integración con herramientas como Terraform y CloudFormation.

1.1.13. Optimización de CI/CD en entornos DevOps

Según IAEME Publication (2022), la optimización de pipelines de CI/CD en entornos DevOps requiere estrategias como paralelización, distribución, containerización y orquestación, lo que permite mejorar la eficiencia y reducir los tiempos de despliegue. Además, la integración de GitOps y la automatización avanzada con Inteligencia Artificial están emergiendo como tendencias clave para mejorar la gestión de infraestructura en la nube.

1.1.14. Validación y pruebas automatizadas de infraestructura

La validación de la infraestructura es una fase fundamental en los pipelines de CI/CD. Según HashiCorp (2025), Terraform permite comprobar las configuraciones antes de su despliegue, lo que asegura que los cambios se implementen de manera segura y puedan ser reproducidos. Las pruebas automatizadas incluyen:

- **Pruebas de sintaxis y validación de código:** Uso de herramientas como Terraform Validate y Checkov.
- **Pruebas de integración:** Validación de la compatibilidad de los cambios con la infraestructura existente.
- **Pruebas de seguridad:** Implementación de escaneos de vulnerabilidades con Open Policy Agent (OPA).

1.1.15. Implementación de prácticas DevOps con IaC

La adopción de Infraestructura como Código (IaC) en DevOps permite la automatización de procesos y la mejora en la gestión de infraestructura. DevOps es una metodología que combina las áreas de desarrollo (Dev) y operaciones (Ops) para acelerar el desarrollo y la entrega de software, asegurando calidad y fiabilidad. Su objetivo principal es la colaboración entre equipos para entregar valor a los clientes de manera constante y eficiente.

Según GitLab (2025), los procesos de integración y entrega continuas contribuyen a la optimización del desarrollo de aplicaciones mediante mecanismos de supervisión y automatización. Además, investigaciones recientes destacan que la implementación de IaC en DevOps incrementa la eficiencia operativa y minimiza errores en la administración de infraestructura (Saxena et al., 2025).

Las principales prácticas DevOps con IaC incluyen:

- **Automatización del aprovisionamiento de infraestructura**: Uso de herramientas como Terraform y Ansible para la gestión de entornos en la nube, permitiendo la creación y modificación de infraestructura de manera declarativa (Tamburri, 2019).
- **Monitoreo y observabilidad**: Implementación de herramientas como Prometheus y Grafana para la supervisión de infraestructura, asegurando la detección temprana de problemas y optimización del rendimiento (Saxena et al., 2025).
- **Gestión de configuración**: Uso de Puppet y Chef para la administración de servidores y aplicaciones, facilitando la estandarización y el control de cambios en entornos distribuidos (Tamburri, 2019).

1.1.16. Seguridad en Infraestructura como Código (IaC)

Principios de seguridad en IaC

La seguridad en Infraestructura como Código (IaC) es fundamental para garantizar la protección de los entornos en la nube y evitar vulnerabilidades. Los principios clave incluyen:

- **Mínimos privilegios:** La administración de accesos debe basarse en el principio de otorgar solo los permisos estrictamente necesarios a cada usuario o servicio. Según

Microsoft (2025), esta estrategia contribuye a reducir la superficie de ataque y a minimizar el impacto de posibles vulnerabilidades en el sistema.

- **Segmentación de red:** La división de entornos y la aplicación de políticas de acceso rigurosas fortalecen la seguridad de la infraestructura. De acuerdo con Suppi Boldrito (2022), la segmentación de red en la gestión de infraestructura como código (IaC) facilita el aislamiento de recursos críticos y disminuye el riesgo de accesos no autorizados.
- **Cifrado:** La implementación de técnicas de cifrado es fundamental para la protección de datos y la prevención de accesos no autorizados. Según AWS (2025), Terraform permite gestionar de manera segura credenciales y datos sensibles mediante mecanismos de cifrado y almacenamiento remoto.

Herramientas de escaneo y auditoría en IaC

Para garantizar la seguridad en IaC, existen herramientas especializadas en escaneo y auditoría:

- **Checkov:** Realiza análisis estático de configuraciones IaC para detectar vulnerabilidades y malas prácticas.
- **Terraform Sentinel:** Permite la aplicación de políticas de seguridad en configuraciones Terraform.
- **Open Policy Agent (OPA):** Facilita la validación de políticas de seguridad en IaC y permiten detectar configuraciones inseguras antes de su implementación, reduciendo riesgos operativos.

Protección de credenciales y secretos

El manejo seguro de credenciales y secretos es crucial en IaC. Algunas herramientas clave incluyen:

- **Vault:** Solución de HashiCorp para la gestión segura de credenciales y secretos.
- **AWS Secrets Manager:** Servicio de AWS para el almacenamiento y recuperación segura de credenciales.
- **HashiCorp Vault:** ofrece cifrado y mecanismos de control de acceso para la gestión segura de credenciales. De acuerdo con Suppi Boldrito (2022), en entornos de infraestructura como código (IaC), es fundamental implementar almacenamiento protegido y restringir el acceso a datos sensibles con el fin de prevenir posibles filtraciones de información.

Cumplimiento normativo y auditorías de seguridad en IaC

Las auditorías de seguridad en entornos de Infraestructura como Código (IaC) son fundamentales para asegurar el cumplimiento de normativas internacionales como ISO 27001, GDPR y NIST. Según Microsoft (2025), estas políticas deben estructurarse conforme a estándares globales con el objetivo de proteger los datos y preservar la integridad de los sistemas.

1.2. Tendencias y Evolución de IaC

La Infraestructura como Código (IaC) ha transformado la administración de infraestructura en la nube al posibilitar la automatización y disminuir la incidencia de errores humanos. Según Paradigma Digital (2022), su evolución ha ido desde la utilización de scripts básicos hasta el desarrollo de herramientas avanzadas como Terraform y Ansible, lo que ha optimizado la gestión de entornos tecnológicos en la nube.

1.2.1. Análisis de investigaciones Recientes sobre IaC

La Infraestructura como Código (IaC) ha sido objeto de múltiples estudios en los últimos años, especialmente en el contexto de la computación en la nube. Estudios recientes han analizado el impacto de IaC en la seguridad y eficiencia operativa. Microsoft (2023) destaca que IaC ha evolucionado para resolver problemas de desfase en entornos de implementación, permitiendo una mayor coherencia y reduciendo errores humanos. Además, IBM (2023) menciona que la automatización con IaC ha sido clave para la adopción de estrategias de DevOps1.

1.2.2. Desafíos en la adopción de la IaC

La adopción de Infraestructura como Código (IaC) presenta diversos retos para las organizaciones, entre ellos la resistencia al cambio, la curva de aprendizaje y la compatibilidad con sistemas heredados. Según Microsoft (2023), la falta de conocimiento sobre IaC y el temor a la automatización pueden generar rechazo en los equipos de TI, lo que dificulta su implementación. De igual manera, IBM (2023) señala que la interoperabilidad entre herramientas sigue siendo un obstáculo, lo que obliga a las empresas a evaluar cuidadosamente sus opciones antes de proceder con la adopción.

Otro aspecto relevante es la curva de aprendizaje, ya que la implementación de IaC requiere que los equipos de TI desarrollen nuevas habilidades en herramientas como Terraform, Ansible y AWS CloudFormation. Según Socioestrategia (2023), la desconfianza en el liderazgo y una comunicación deficiente pueden incrementar la resistencia al cambio dentro de la organización. Además, muchas empresas aún dependen de infraestructuras tradicionales, lo que dificulta la transición hacia IaC. Pacheco-Ruíz et al. (2020) destacan que los procesos de adaptación deben

estar respaldados por una cultura organizacional sólida para minimizar la resistencia y facilitar la adopción de nuevas tecnologías.

1.2.3. Limitaciones técnicas y cómo se están abordando en la actualidad

A pesar de sus beneficios, la Infraestructura como Código (IaC) enfrenta ciertas limitaciones técnicas. Una de ellas es la complejidad en la administración de configuraciones, lo que requiere una planificación minuciosa para prevenir errores. Según Skiller Academy (2025), la automatización de infraestructura con IaC demanda un enfoque estructurado para garantizar la precisión en la configuración.

Otro reto importante es la compatibilidad entre herramientas, ya que no todas las soluciones de IaC se integran fácilmente con las plataformas existentes, lo que puede generar dificultades en la interoperabilidad. AIMojo (2025) destaca que las empresas deben realizar una evaluación detallada de las herramientas disponibles para asegurar una integración eficiente y sin inconvenientes.

CAPÍTULO II

Materiales y Métodos

El presente capítulo expone de manera detallada los materiales y métodos empleados para abordar la problemática derivada de la gestión manual de infraestructuras tecnológicas. En este estudio se enfatiza la importancia de definir claramente tanto las técnicas de recolección de datos como la metodología de desarrollo del sistema, aspectos fundamentales para garantizar la robustez y la reproducibilidad de los resultados. La elección cuidadosa de estos métodos incluye la implementación de herramientas de Infraestructura como Código (IaC), integradas en un pipeline de CI/CD, y la aplicación de metodologías ágiles.

Este enfoque metodológico facilita la detección temprana y la corrección oportuna de errores en la configuración de entornos en la nube, al mismo tiempo que optimiza los procesos de despliegue, asegurando una administración más segura, escalable y eficiente. En consecuencia, la sistematización de los materiales y métodos presentados se erige como un elemento esencial para demostrar la viabilidad e impacto de la transformación digital en el ámbito de la administración tecnológica.

La investigación se clasifica como aplicada, ya que se orienta a resolver una problemática real relacionada con la administración de infraestructuras en la nube. En este estudio se plantea la optimización de los procesos de despliegue y gestión de configuraciones mediante la implementación de IaC utilizando Terraform y Azure. Este enfoque automatiza la creación y el mantenimiento de entornos tecnológicos, mejorando significativamente la eficiencia, seguridad y escalabilidad operativa.

2.1. Alcance de la Investigación

En este contexto, se delimita el alcance del estudio a la evaluación del despliegue en la nube utilizando IaC para la empresa Ecualatino. En particular, se desarrolla un modelo de automatización centrado únicamente en la provisión y gestión de la base de datos mediante Terraform y Azure, lo que permite validar su efectividad en ese componente específico de la infraestructura tecnológica.

La investigación abarca los siguientes aspectos:

- Definición de requisitos y configuración automatizada de la base de datos en la nube.
- Comparación entre métodos tradicionales y enfoques basados en IaC para medir la eficiencia del despliegue de bases de datos.
- Implementación de un pipeline de CI/CD enfocado en la infraestructura de base de datos, asegurando la integración continua y una administración eficiente del entorno.

El estudio no aborda la automatización de otros componentes de la infraestructura tecnológica**,** ni la implementación en proveedores de nube distintos a Azure, ni el desarrollo de arquitecturas híbridas o multinube.

De esta manera, los materiales y métodos descritos proporcionan una base sólida para evaluar el impacto de la automatización en la gestión de bases de datos en la nube.

2.2. Metodología de desarrollo

La metodología aplicada en este proyecto se basa en un enfoque incremental y sistemático para la implementación de infraestructura como código (IaC) utilizando Terraform en la plataforma Azure, orientado a automatizar el despliegue y gestión de entornos tecnológicos. Se adoptó una combinación de buenas prácticas en DevOps, junto con herramientas modernas como Azure CLI **y** Terraform**,** para garantizar la trazabilidad, escalabilidad y reproducibilidad de los resultados obtenidos.

A continuación, se describen los pasos ejecutados durante el desarrollo del proyecto:

2.2.1. Planificación y Diseño del Entorno

2.2.1.1. Definición de Requisitos

En esta fase, se identificaron los requisitos específicos del proyecto en ECUALATINO, tales como los recursos necesarios, la configuración de red, las consideraciones de seguridad y las necesidades de escalabilidad. Fue fundamental comprender las metas del entorno cloud para garantizar que la infraestructura desarrollada sea eficiente y capaz de soportar la carga prevista. Se consideraron aspectos como:

- Recursos de cómputo (máquinas virtuales, contenedores)
- Almacenamiento y bases de datos
- Políticas de seguridad (firewalls, permisos)
- Requerimientos de escalabilidad y disponibilidad

2.2.1.2. Diseño de la arquitectura base en Azure

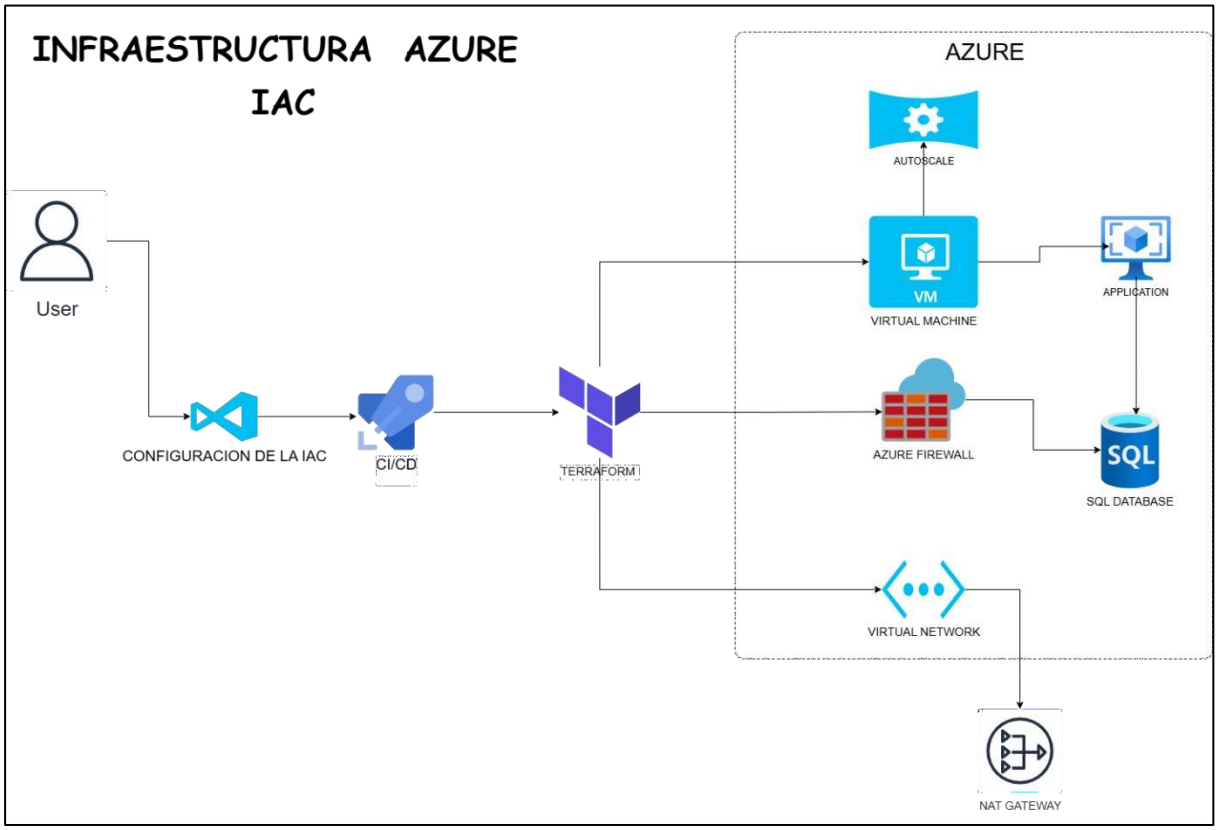
Previo a la implementación técnica de la infraestructura, se realizó el diseño de la arquitectura base que serviría como soporte para el despliegue de los recursos necesarios en la nube de Azure.

Este diseño contempló aspectos clave como:

- **Región geográfica** donde se ubicarían los recursos (por ejemplo, East US o West Europe).
- **Red virtual (VNet)** que proporcionaría conectividad interna entre los servicios.

- **Subredes** para segmentar los distintos tipos de recursos (por ejemplo, front-end, back-end, base de datos).
- **Grupo de recursos (Resource Group)** para organizar los elementos relacionados.
- **Máquinas virtuales**, bases de datos u otros servicios necesarios, según los requerimientos del proyecto.
- **Reglas de seguridad (NSG)** para controlar el acceso a la red.
- **Almacenamiento y monitoreo**, si fueran necesarios.

Figura 2:
Infraestructura Azure IaC



La figura 2 muestra la arquitectura de infraestructura implementada en Microsoft Azure utilizando el enfoque de Infraestructura como Código (IaC) mediante la herramienta Terraform. Este diseño busca automatizar y estandarizar el aprovisionamiento de los recursos necesarios para el despliegue de aplicaciones, asegurando escalabilidad, seguridad y eficiencia operativa.

La arquitectura se compone de los siguientes elementos principales:

- **Dev Console / Configuración IaC**: El proceso inicia desde el entorno de desarrollo, donde se configura el código de IaC utilizando Terraform. Este código contiene definiciones declarativas para todos los recursos necesarios en la nube.

- **Terraform**: Actúa como motor de automatización, leyendo los archivos de configuración para crear y gestionar la infraestructura en Azure. Es responsable de orquestar la creación de redes, máquinas.

2.2.2. Selección de Herramientas de IaC

Para automatizar y gestionar la infraestructura en la nube, se optó por utilizar Terraform como la herramienta principal de Infraestructura como Código (IaC), dada su compatibilidad multiplataforma y su amplia adopción en la industria. Puesto que el entorno de Ecualatino se desplegó principalmente en Microsoft Azure, se eligió el proveedor hashicorp/azurerm, que permite definir recursos y configuraciones específicas de Azure mediante scripts declarativos. La versión seleccionada fue 3.0, la cual es compatible con las funciones y recursos necesarios para el despliegue.

La elección de esta herramienta y proveedor fue motivada por su estabilidad, soporte y la integración sencilla con Azure, facilitando así el despliegue automatizado y controlado de la infraestructura en la nube de Ecualatino. Además, Terraform permitió gestionar la infraestructura de manera reproducible, versiones controladas y con un proceso que reduce errores humanos durante el despliegue y actualización de recursos.

2.2.3. Instalación y configuración del entorno de desarrollo

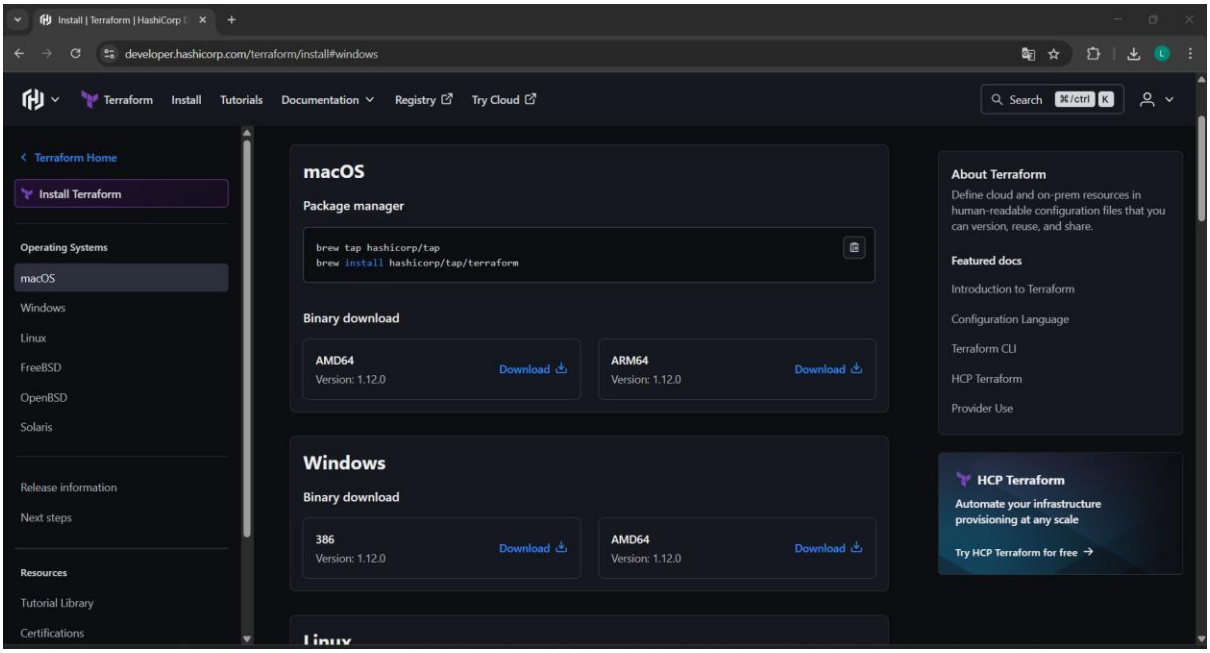
Con el objetivo de gestionar la infraestructura como código, se procedió a la instalación y configuración de las herramientas necesarias. Entre estas, se utilizó **Terraform**, una herramienta desarrollada por HashiCorp que permite definir y aprovisionar infraestructura mediante archivos de configuración.Instalación de **Terraform**.

2.2.3.1. Instalación de Terraform

Para llevar a cabo la instalación de Terraform en un sistema operativo Windows, se siguieron los pasos descritos a continuación:

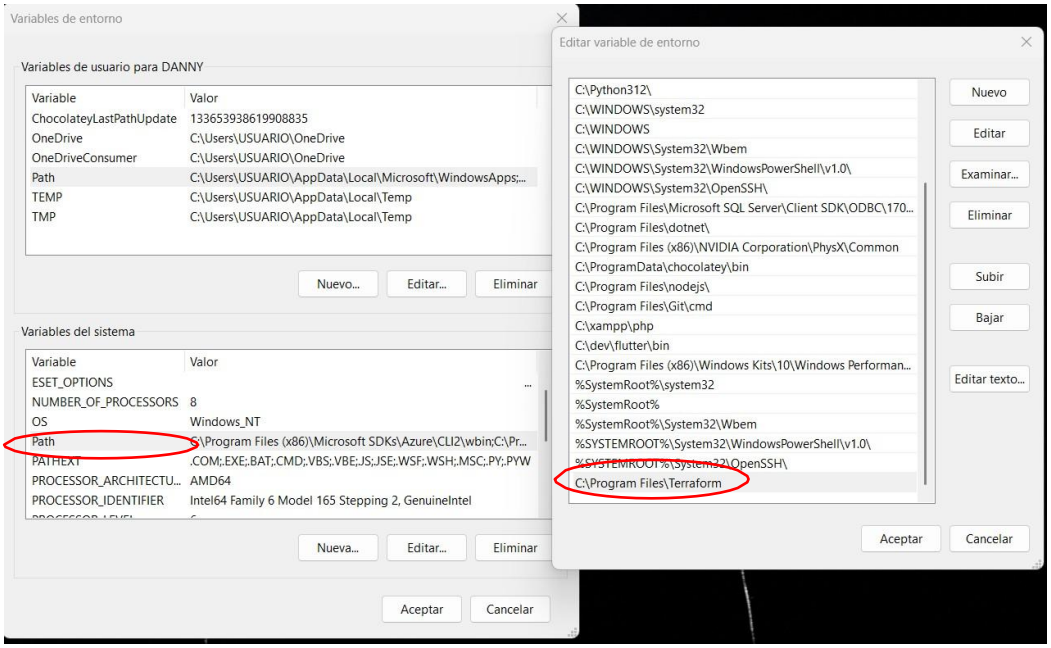
A. Se accedió al sitio oficial de Terraform en el siguiente enlace (Ver Figura 3).: https://developer.hashicorp.com/terraform/install#windows

Figura 3:
Instalacion de Terraform



B. Se identificó la versión del sistema operativo, confirmando que se trataba de una arquitectura de 64 bits. En función de ello, se descargó el binario correspondiente a la versión **AMD64**.
C. El archivo descargado fue un comprimido en formato `.zip`, el cual se descomprimió en una carpeta designada para herramientas de desarrollo (por ejemplo, `C:\Program Files\Terraform`).
D. Posteriormente, se agregó la ruta de la carpeta que contenía el ejecutable `terraform.exe` a la variable de entorno **PATH** del sistema. Esto permitió ejecutar Terraform desde cualquier terminal de comandos (Ver Figura 4).

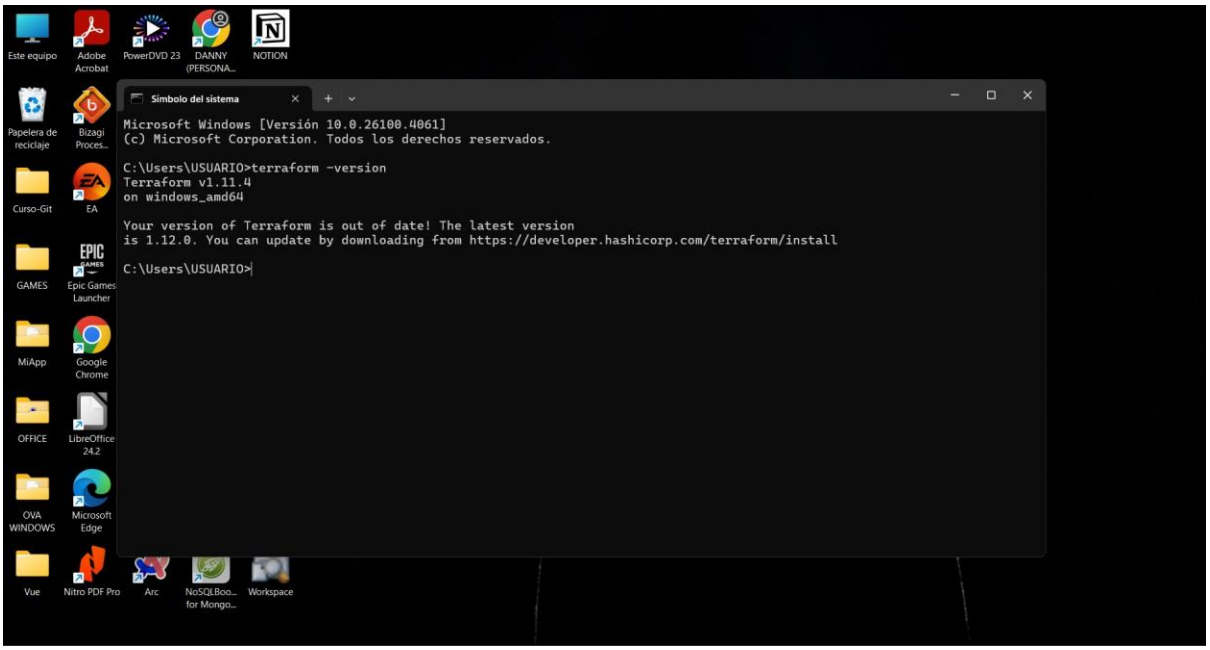
Figura 4:
Variables de entorno



E. Para verificar la instalación, se abrió una ventana del símbolo del sistema (cmd) y se ejecutó el siguiente comando (Ver figura 5):

terraform -version

Figura 5:
Versión de Terraform

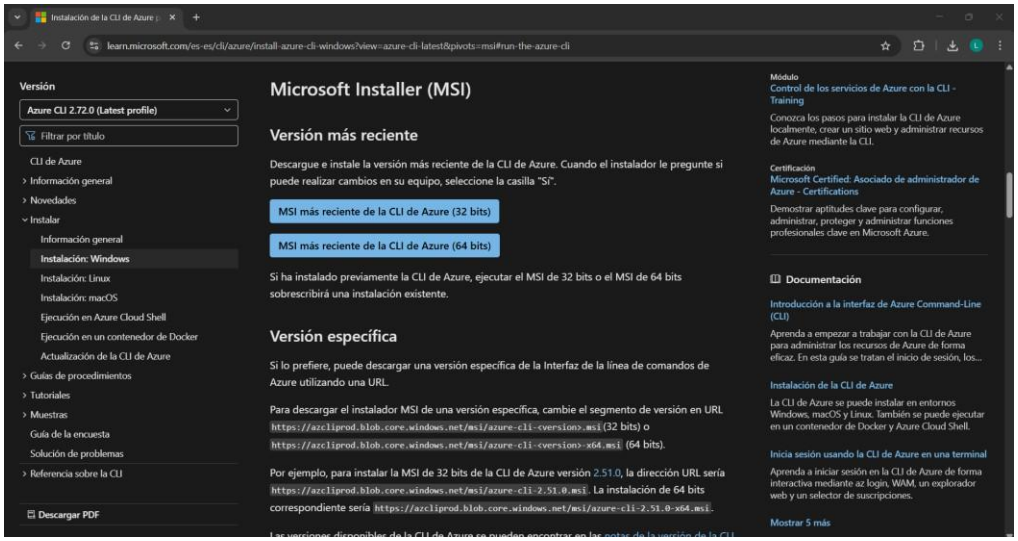


2.2.3.2. Configuración de Azure CLI para autenticarse con la cuenta de Azure.

Para interactuar con los servicios de Azure desde la línea de comandos y facilitar la integración con Terraform, se procedió a la instalación y configuración de **Azure CLI** (Command-Line Interface).

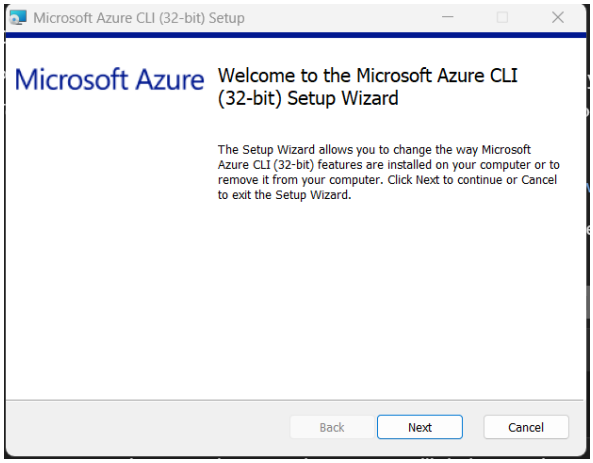
A. Se descargó el instalador oficial desde la página de Microsoft (Ver figura 6): https://learn.microsoft.com/es-es/cli/azure/install-azure-cli-windows?view=azure-cli-latest&pivots=msi#run-the-azure-cli

Figura 6:
Instalador Azure CLI



B. Una vez descargado, se ejecutó el instalador y se completó el proceso de instalación mediante el asistente(Ver figura 7).

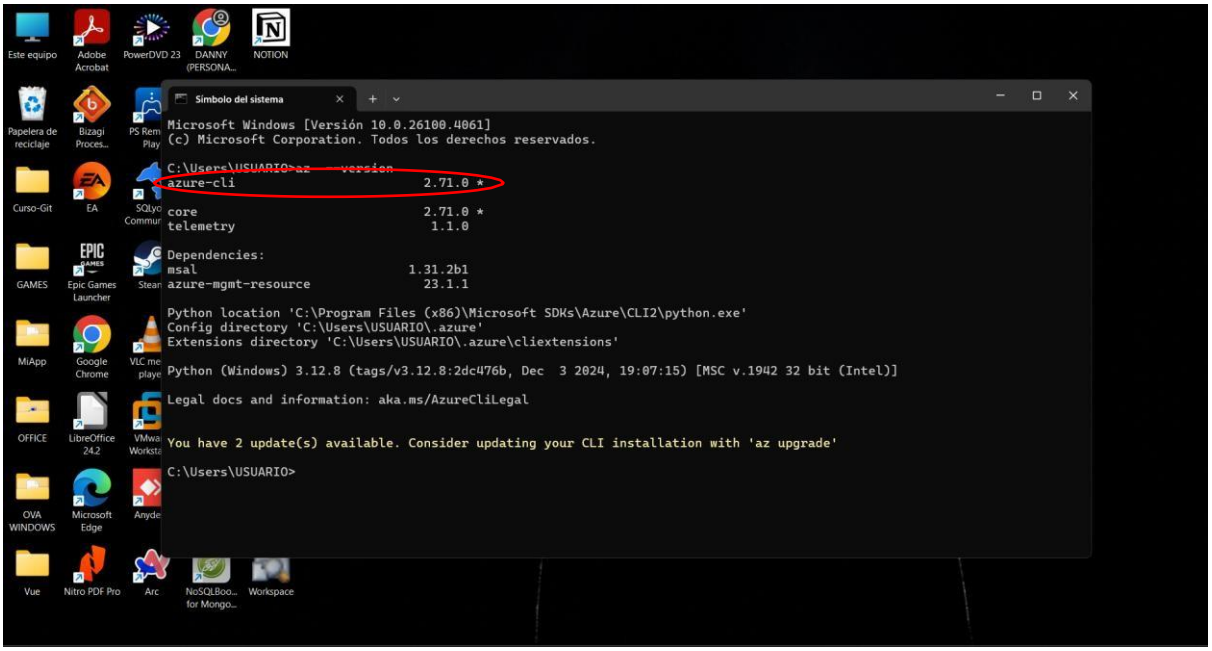
Figura 7:
Setup Azure CLI



C. Finalizada la instalación, se validó su funcionamiento ejecutando el siguiente comando en la terminal:

```
az --version
```

Figura 8:
Versión Azure CLI



D. Para autenticarse con la cuenta de Azure, se utilizó el comando:

az login

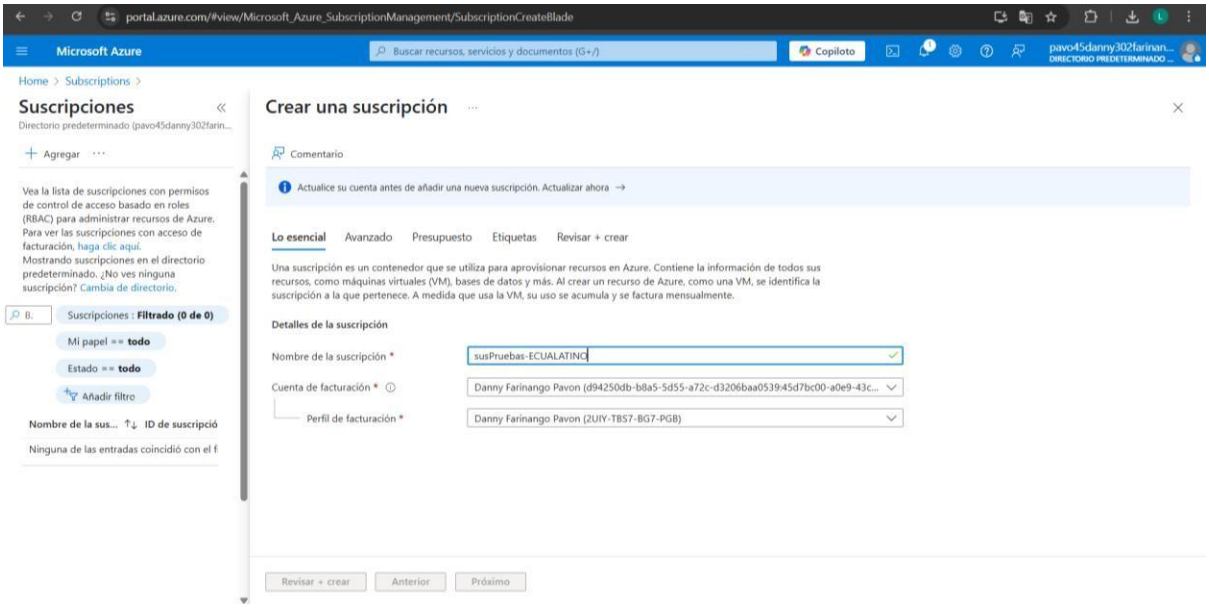
Al ejecutarlo, se abrió una ventana del navegador que solicitó las credenciales de acceso. Una vez autenticado, la terminal mostró la información asociada a la cuenta, incluyendo las suscripciones disponibles.

2.2.3.3. Creación de una suscripción de Azure y validación de permisos para el uso de recursos.

Como parte de la preparación del entorno, se requirió una **suscripción activa de Azure** para poder aprovisionar y gestionar recursos a través de Terraform.

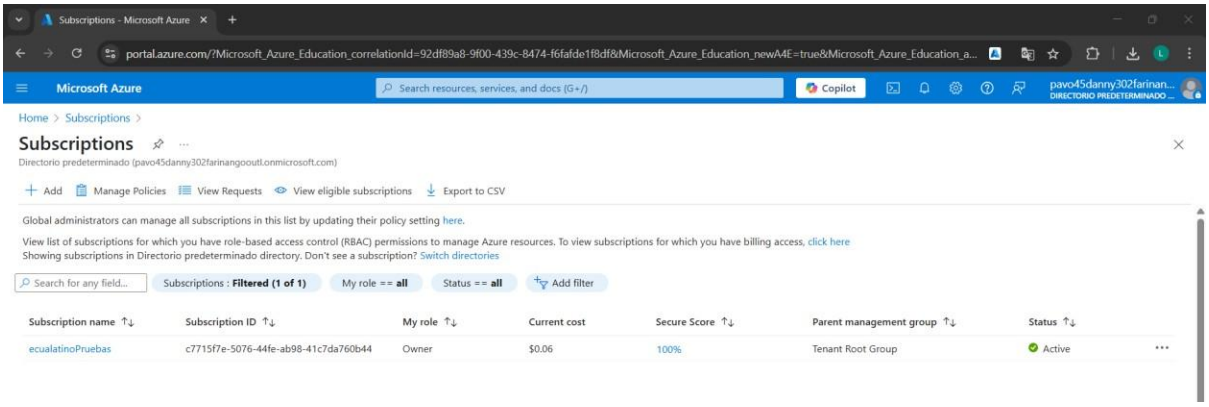
A. Se accedió al portal de Azure (https://portal.azure.com) y se procedió a crear una nueva suscripción bajo el modelo de prueba gratuita (Free Trial) o utilizando una cuenta previamente habilitada con permisos administrativos(Ver Figura 9).

Figura 9:
Suscripción de Azure



B. Una vez creada la suscripción, se validó que la cuenta tuviera los permisos necesarios para desplegar recursos, tales como **Owner** o **Contributor**, dentro del directorio y grupo de recursos correspondiente (Ver Figura 10).

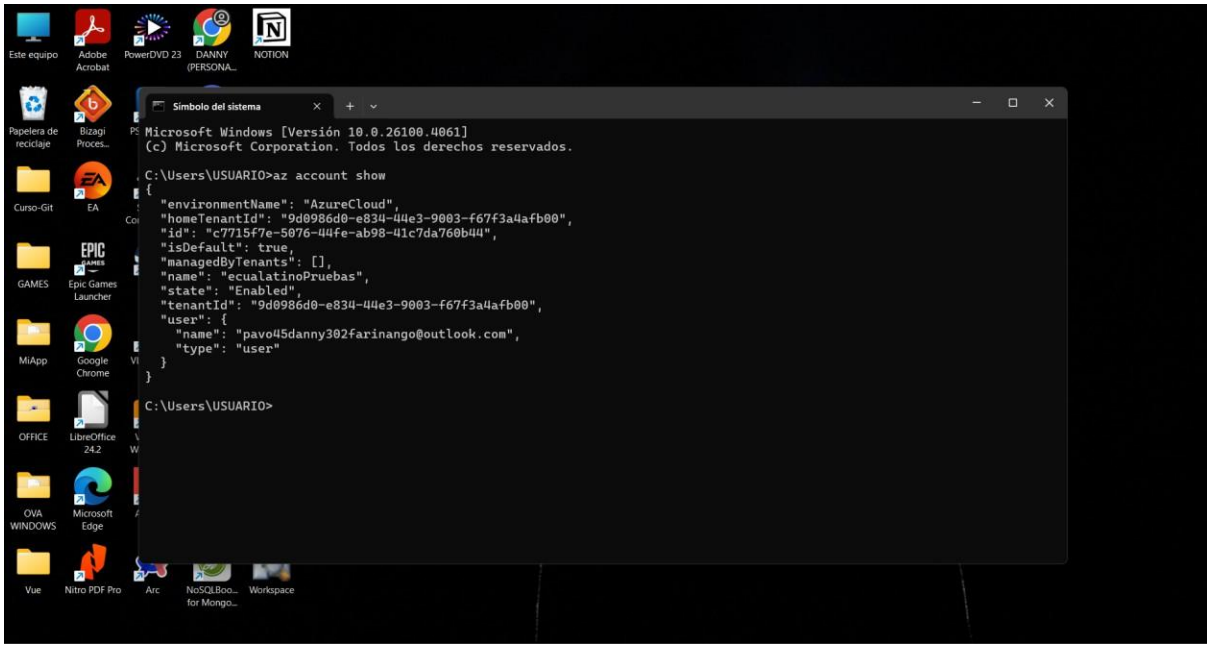
Figura 10:
Suscripción Creada



C. La suscripción activa se verificó con el siguiente comando de Azure CLI:

```
az account show
```

Figura 11:
Suscripción Activa

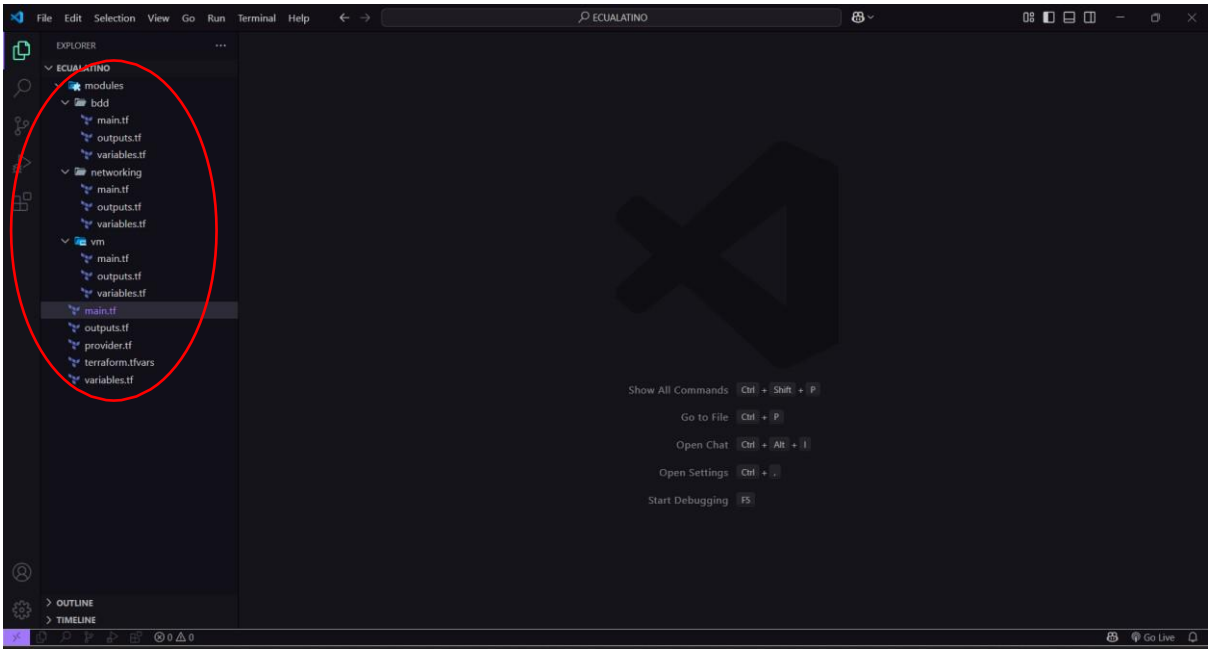


Este comando permitió obtener el subscriptionId que posteriormente fue utilizado en la configuración de Terraform para conectarse con Azure.

2.2.3.4. Creación de archivos de configuración con Terraform

Como parte fundamental del desarrollo del proyecto, se procedió a la creación de los archivos de configuración en lenguaje HCL (HashiCorp Configuration Language), organizando la infraestructura mediante el enfoque modular que ofrece Terraform. Esta estructuración permitió gestionar de forma reutilizable, ordenada y escalable los recursos desplegados en Azure.

Figura 12:
Creación de archivos Terraform



El objetivo principal de esta fase fue automatizar la creación de recursos en la nube tales como grupos de recursos, redes virtuales, subredes, máquinas virtuales y bases de datos SQL, mediante scripts declarativos. Para ello, se estructuró el proyecto con una carpeta raíz y una subcarpeta denominada modules, que a su vez contiene submódulos como networking, vm y bdd (base de datos).

Los recursos definidos incluyeron:

azurerm_virtual_network y azurerm_subnet: Se definió una red virtual principal junto con subredes específicas para separar los componentes de la aplicación y la base de datos. Esto permitió una segmentación lógica del tráfico interno en Azure, facilitando la organización y la seguridad de los servicios.

azurerm_network_security_group (NSG): Se crearon grupos de seguridad para restringir y controlar el acceso a las subredes. Las reglas permitieron únicamente tráfico necesario (por ejemplo, RDP, HTTP/HTTPS o SQL desde rangos IP definidos), minimizando los vectores de ataque.

azurerm_public_ip y azurerm_network_interface: Se aprovisionó una IP pública estática para acceder a la máquina virtual desde el exterior y se configuró una interfaz de red que enlaza la VM con la red virtual, integrando la configuración de IP y seguridad.

azurerm_windows_virtual_machine: Se implementó una máquina virtual con Windows Server 2022, configurada con discos Premium para alto rendimiento. Esta VM fue utilizada como servidor principal para desplegar aplicaciones y servicios.

azurerm_virtual_machine_extension: Se utilizó una extensión personalizada para instalar y configurar automáticamente el servidor web IIS mediante un script en PowerShell ejecutado al momento del despliegue.

azurerm_mssql_server y azurerm_mssql_database: Se desplegó un servidor de base de datos SQL Server en Azure, junto con una base de datos de tipo General Purpose Serverless, optimizada para uso intermitente y ahorro de costos.

azurerm_mssql_firewall_rule y azurerm_mssql_virtual_network_rule: Se configuraron reglas de firewall para permitir conexiones desde servicios de Azure y desde la subnet donde se encontraba la VM, garantizando el acceso seguro al servidor SQL.

azurerm_private_endpoint: Se implementó un endpoint privado para conectar de forma segura a la base de datos, evitando el tráfico por internet público y mejorando la seguridad general de la arquitectura.

2.2.4. Escritura del Código IaC

2.2.4.1. Configuración del Proveedor de Infraestructura – Archivo provider.tf

En este archivo se definió el proveedor de infraestructura requerido para la implementación del entorno en la nube. Se especificó el uso del proveedor oficial de Azure para Terraform (azurerm), el cual permite interactuar con los recursos ofrecidos por Azure a través del lenguaje HCL.

El bloque terraform estableció como dependencia principal el proveedor azurerm, restringido a versiones compatibles con la serie 3.0, lo que garantizó estabilidad y compatibilidad con las funcionalidades modernas sin incorporar versiones experimentales o inestables. El código utilizado se muestra en la Figura 13.

Figura 13:
Código del Proveedor

```
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~>3.0"
    }
  }
```

```
}
```

Posteriormente, se configuró el bloque provider, que habilitó las características predeterminadas de azurerm necesarias para interactuar con los servicios de Azure. La configuración features {} se dejó vacía, ya que no se requerían ajustes adicionales en esta etapa. El código utilizado se muestra en la Figura 14.

Figura 14:
Código Configuracion del Proveedor

```
provider "azurerm" {
  features {}
}
```

Esta configuración fue esencial para inicializar correctamente el entorno de Terraform y autenticar la conexión con la cuenta de Azure, permitiendo así el aprovisionamiento automatizado de la infraestructura descrita en los siguientes archivos de configuración.

2.2.4.2. Declaración de Variables Parametrizadas – Archivo variables.tf

En este archivo se declararon todas las variables de entrada necesarias para parametrizar el despliegue de la infraestructura. Este enfoque permitió desacoplar la definición de los recursos de los valores concretos, facilitando la reutilización y el cambio de configuraciones sin necesidad de modificar directamente el código fuente.

- **Parámetros generales**

Se definieron variables clave como el nombre del grupo de recursos (resource_group_name), la ubicación geográfica de Azure (location) y el entorno de despliegue (environment). Estas variables establecieron las bases comunes para todos los recursos, el código utilizado para la declaración de variables generales se presenta en la Figura 15.

Figura 15:
Código de Variables Generales

```
variable "resource_group_name" {
  description = "Nombre del grupo de recursos"
  type        = string
  default     = "ecualatino-rg"
}

variable "location" {
  description = "Ubicación de Azure para los recursos"
  type        = string
  default     = "East US"
}
```

```
variable "environment" {
  description = "Entorno (dev, test, prod)"
  type        = string
  default     = "dev"
}
```

- **Parámetros para la máquina virtual**

Para la creación de la máquina virtual Windows Server 2022, se especificaron variables relacionadas con su nombre (vm_name), tamaño (vm_size) y credenciales de acceso (admin_username y admin_password). La contraseña se marcó como sensible para garantizar que no fuera expuesta en los planes ni en los logs de Terraform, el código utilizado para la declaración de variables para la máquina virtual se presenta en la Figura 16.

Figura 16:
Código de Variables Para la Máquina Virtual

```
# Variables para VM
variable "vm_name" {
  description = "Nombre de la máquina virtual"
  type        = string
  default     = "ecualatino-vm"
}

variable "vm_size" {
  description = "Tamaño de la máquina virtual"
  type        = string
  default     = "Standard_D2s_v3"
}

variable "admin_username" {
  description = "Nombre de usuario administrador"
  type        = string
  default     = "adminuser"
}

variable "admin_password" {
  description = "Contraseña de administrador"
  type        = string
  sensitive   = true
}
```

- **Parámetros para la base de datos SQL**

Asimismo, se parametrizó el despliegue del servidor SQL y la base de datos. Se utilizaron variables para el nombre del servidor (sql_server_name), el nombre de la base de datos (database_name), así como las credenciales del administrador SQL (sql_admin_username y

sql_admin_password). También en este caso, la contraseña fue tratada como información sensible, el código utilizado para la declaración de variables para la base de datos SQL se presenta en la Figura 17.

Figura 17:
Código de Variables para la Base Datos SQL

```
# Variables para SQL
variable "sql_server_name" {
  description = "Nombre del servidor SQL"
  type        = string
  default     = "ecualatino-sql"
}

variable "database_name" {
  description = "Nombre de la base de datos"
  type        = string
  default     = "ecualatino-db"
}

variable "sql_admin_username" {
  description = "Nombre de usuario administrador de SQL"
  type        = string
  default     = "sqladmin"
}

variable "sql_admin_password" {
  description = "Contraseña de administrador de SQL"
  type        = string
  sensitive   = true
}
```

Estas variables proporcionaron flexibilidad y escalabilidad al proyecto, permitiendo adaptar la infraestructura a diferentes entornos (desarrollo, pruebas o producción) mediante el uso de distintos archivos de valores (.tfvars) o argumentos en línea.

2.2.4.3. Definición de Parámetros de Configuración – Archivo. tfvars

Durante el desarrollo de la infraestructura como código con Terraform, se creó un archivo denominado .tfvars, el cual se utilizó para definir variables clave necesarias para la provisión de recursos en la nube. Este archivo permitió separar la lógica del despliegue de los valores específicos de configuración, facilitando la reutilización y el mantenimiento del código.

El archivo contenía las siguientes variables:

- **resource_group_name:** Se definió el nombre del grupo de recursos como "ecualatino-rg", el cual agrupó todos los recursos implementados en Azure.

- **location:** Se especificó la región "East US 2" como el lugar de despliegue, seleccionada por su disponibilidad y características técnicas.
- **environment:** Se indicó que el entorno de despliegue era de desarrollo ("dev"), lo cual ayudó a diferenciarlo de otros entornos como pruebas o producción.

El código utilizado para la declaración de variables de entrada para la creación de recursos de la infraestructura se presenta en la Figura 18.

Figura 18:
Código de variables generales de entrada (. tfvars)

```
resource_group_name = "ecualatino-rg"
location            = "East US 2"
environment         = "dev"
```

Configuración de la máquina virtual (VM)

Figura 19:
Código de variables de entrada de la maquina virtual (.tfvars)

```
# VM
vm_name             = "ecualatino-vm"
vm_size             = "Standard_D2s_v3"
admin_username      = "adminuser"
admin_password      = "TuContraseñaSegura123!"  # Cambia esto por una
contraseña segura
```

- **vm_name:** Se estableció el nombre de la máquina virtual como "ecualatino-vm".
- **vm_size:** Se utilizó el tamaño "Standard_D2s_v3", adecuado para entornos de desarrollo debido a su balance entre rendimiento y costo.
- **admin_username y admin_password:** Se definieron las credenciales de acceso administrativo a la máquina virtual. Por seguridad, se utilizó una contraseña robusta y única.

El fragmento de código mostrado en la Figura 19 corresponde a la declaración de las variables de entrada necesarias para la implementación de la máquina virtual mediante Terraform.

Configuración de la base de datos SQL

Figura 20:
Código de variables de entrada para la base de datos (.tfvars).

```
# SQL
sql_server_name     = "ecualatino-sql"
database_name       = "AppDB"
sql_admin_username  = "sqladmin"
sql_admin_password  = "OtraContraseñaSegura456!"  # Cambia esto por una
contraseña segura
```

- **sql_server_name:** Se nombró el servidor SQL como "ecualatino-sql".
- **database_name:** Se creó una base de datos con el nombre "AppDB".
- **sql_admin_username y sql_admin_password:** Se configuraron las credenciales del administrador de la base de datos. Al igual que en el caso de la VM, se utilizaron contraseñas seguras que fueron almacenadas y gestionadas con las debidas precauciones.

El fragmento de código mostrado en la Figura 20 corresponde a la declaración de las variables de entrada necesarias para la implementación de la base de datos mediante Terraform.

2.2.4.4. Definición de salidas en Terraform – Archivo outputs.tf

Como parte de la automatización de la infraestructura mediante Terraform, se implementó un archivo denominado outputs.tf en el directorio raíz del proyecto. Este archivo tuvo como propósito definir las salidas (outputs) relevantes del despliegue, permitiendo visualizar de forma inmediata los valores clave tras la ejecución del plan de infraestructura, sin necesidad de inspeccionar manualmente los recursos creados desde el portal de Azure o a través de comandos adicionales por línea de comandos.

Salidas asociadas a la máquina virtual

Se configuraron dos salidas principales relacionadas con la máquina virtual desplegada mediante el módulo vm. Estas salidas resultan esenciales para habilitar el acceso a la máquina virtual y facilitar su integración con otros componentes del sistema. El código correspondiente se presenta en la Figura 21.

Figura 21:
Código de salidas asociadas a la máquina virtual (outputs.tf)

```
output "vm_public_ip" {
  description = "Dirección IP pública de la máquina virtual"
  value       = module.vm.vm_public_ip
}

output "vm_private_ip" {
  description = "Dirección IP privada de la máquina virtual"
  value       = module.vm.vm_private_ip
}
```

- **vm_public_ip**: Representó la dirección IP pública asignada a la máquina virtual, necesaria para el acceso remoto desde internet.
- **vm_private_ip**: Correspondió a la dirección IP privada dentro de la red virtual, útil para la comunicación interna entre servicios en la misma red.

Estas salidas facilitaron el acceso y la integración con otros componentes del sistema, como bases de datos o herramientas de monitoreo, así como la configuración de conexiones SSH o RDP durante las fases de pruebas y despliegue.

Salidas relacionadas con el servidor SQL y la base de datos

También se definieron salidas relevantes para el componente de base de datos, específicamente aquellas generadas por el módulo bdd, el cual contenía la lógica de aprovisionamiento del servidor SQL y la base de datos. El código correspondiente se presenta en la Figura 22.

Figura 22:
Código de salidas asociadas a la base de datos (outputs.tf)

```
output "sql_server_fqdn" {
  description = "Nombre de dominio completo del servidor SQL"
  value       = module.bdd.sql_server_fqdn
}

output "sql_database_name" {
  description = "Nombre de la base de datos SQL"
  value       = module.bdd.database_name
}

output "sql_connection_string" {
  description = "Cadena de conexión a la base de datos SQL"
  value       = module.bdd.connection_string
  sensitive   = true
}
```

- **sql_server_fqdn**: Proporcionó el nombre de dominio completo (FQDN) del servidor SQL, esencial para establecer conexiones externas.
- **sql_database_name**: Reflejó el nombre de la base de datos creada, utilizado en consultas o configuraciones de conexión.
- **sql_connection_string**: Contuvo la cadena de conexión completa a la base de datos. Esta salida se marcó como **sensible**, lo cual ocultó su contenido en la salida estándar de Terraform, resguardando información crítica como nombres de usuario y contraseñas.

Importancia de las salidas en la automatización

La definición adecuada de estas salidas fue fundamental para facilitar la integración con otras aplicaciones o servicios que requerían conectividad con la infraestructura desplegada. Además, se consolidó como una buena práctica en proyectos con Terraform, ya que mejoró la trazabilidad, redujo errores manuales y reforzó el enfoque DevOps mediante la reutilización de valores generados dinámicamente.

2.2.4.5. Estructura principal del despliegue – Archivo main.tf

El archivo main.tf, ubicado en la raíz del proyecto, definió la estructura general del despliegue de la infraestructura, adoptando una arquitectura modular orientada a la reutilización y mantenibilidad del código. Esta arquitectura se basó en tres componentes clave: **red**, **máquina virtual** y **base de datos**, cada uno encapsulado en módulos independientes. Esta separación de responsabilidades permitió una mayor organización, claridad y escalabilidad del proyecto Terraform.

Creación del grupo de recursos

Antes de invocar los módulos, se implementó un grupo de recursos principal (azurerm_resource_group.rg), el cual sirvió como contenedor lógico para todos los recursos desplegados en Azure. Su creación se parametrizó utilizando variables definidas previamente en el archivo .tfvars, permitiendo su reutilización y adaptación a distintos entornos (por ejemplo, desarrollo, pruebas o producción). El código correspondiente se muestra en la Figura 23**.**

Figura 23:
Creación del grupo de recursos principal (main.tf).

```
# Crear el grupo de recursos principal
resource "azurerm_resource_group" "rg" {
  name     = var.resource_group_name
  location = var.location
  tags = {
    environment = var.environment
  }
}
```

Módulo network

El primer módulo invocado fue network, encargado de construir la infraestructura de red. Este módulo creó una red virtual (VNet) con dos subredes: una para la aplicación (donde se desplegó la máquina virtual) y otra para la base de datos. Además, configuró los grupos de seguridad de red (NSG) con reglas específicas para proteger el tráfico entrante y saliente.

Este módulo recibió como parámetros el nombre del grupo de recursos, la ubicación y el entorno, y como salidas proporcionó los identificadores (IDs) de las subredes creadas, los cuales fueron reutilizados en los siguientes módulos. La implementación se presenta en la Figura 24**.**

Figura 24:
Invocación del módulo de red (main.tf).

```
# Módulo de network
```

```
module "network" {
  source              = "./modules/network"
  resource_group_name = azurerm_resource_group.rg.name
  location            = var.location
  environment         = var.environment
}
```

Módulo de máquina virtual (vm)

El módulo vm fue responsable de aprovisionar una máquina virtual con sistema operativo Windows Server 2022, configurada para hospedar aplicaciones web mediante la instalación automática de IIS (Internet Information Services).

Este módulo recibió las credenciales administrativas, el tamaño de la máquina, y los identificadores de las subredes generadas por el módulo de red, permitiendo así su integración en la infraestructura ya definida. También gestionó la asignación de una dirección IP pública para el acceso externo y una IP privada para comunicación interna. El código correspondiente se muestra en la Figura 25**.**

Figura 25:
Invocación del módulo de máquina virtual (main.tf).

```
# Módulo de máquina virtual
module "vm" {
  source              = "./modules/vm"
  resource_group_name = azurerm_resource_group.rg.name
  location            = var.location
  vm_name             = var.vm_name
  vm_size             = var.vm_size
  admin_username      = var.admin_username
  admin_password      = var.admin_password
  subnet_id           = module.networking.app_subnet_id  # Usar la subnet
creada en el módulo de networking
  environment         = var.environment
  bdd_subnet_id       = module.networking.db_subnet_id    # Pasar la subnet de
la base de datos
}
```

Módulo de base de datos (bdd)

Finalmente, se invocó el módulo bdd, encargado de la creación de un **servidor SQL en Azure**, junto con una base de datos denominada "AppDB". Este módulo también configuró reglas de red específicas para permitir la conectividad segura desde la máquina virtual a través de las subredes definidas previamente, utilizando **endpoints privados**.

Al igual que los módulos anteriores, bdd recibió variables parametrizadas y los identificadores de las subredes para establecer correctamente la comunicación entre la base de datos y la aplicación. La implementación se presenta en la Figura 26**.**

Figura 26:
Invocación del módulo de base de datos (main.tf).

```
# Módulo de base de datos
module "bdd" {
  source              = "./modules/bdd"
  resource_group_name = azurerm_resource_group.rg.name
  location            = var.location
  sql_server_name     = var.sql_server_name
  database_name       = var.database_name
  admin_username      = var.sql_admin_username
  admin_password      = var.sql_admin_password
  subnet_id           = module.networking.db_subnet_id  # Usar la subnet de BD
  vm_subnet_id        = module.networking.app_subnet_id # Subnet de la VM para
la regla de firewall
  environment         = var.environment
}
```

El archivo main.tf permitió orquestar de manera ordenada todos los componentes de la infraestructura, integrando correctamente la red, la máquina virtual y la base de datos mediante el uso de módulos reutilizables. Este enfoque modular no solo facilitó el despliegue automatizado, sino que también incrementó la mantenibilidad y la trazabilidad del proyecto a lo largo de su ciclo de vida.

2.2.4.6. Módulo network: Creación de la red virtual

a. Archivo main.tf

El módulo network fue responsable de implementar la infraestructura de red básica del entorno desplegado en Azure. Este incluyó la creación de una red virtual (VNet)**,** dos subredes segmentadas (una para aplicación y otra para base de datos), así como grupos de seguridad de red (NSG) y sus respectivas asociaciones. Su diseño modular permitió establecer un entorno seguro, escalable y reutilizable para los recursos de aplicación y datos.

Creación de la red virtual

Se aprovisionó una red virtual (azurerm_virtual_network.vnet) con un espacio de direcciones amplio (10.0.0.0/16), que sirvió como base para todas las subredes y recursos internos. Esta red fue creada a partir de variables de entrada definidas en el archivo variables.tf, lo que permitió adaptabilidad entre distintos entornos. La implementación de esta red virtual se muestra en la Figura 27**.**

Figura 27:
Creación de red virtual main-vnet.

```
# Red Virtual
resource "azurerm_virtual_network" "vnet" {
  name                = "main-vnet"
  location            = var.location
  resource_group_name = var.resource_group_name
  address_space       = ["10.0.0.0/16"]

  tags = {
    environment = var.environment
  }
}
```

Definición de subredes

Se establecieron dos subredes específicas, cada una con propósitos distintos La implementación de estas dos subredes se muestra en la Figura 28**.**

- **app_subnet**: destinada a alojar la máquina virtual de la aplicación, con espacio de direcciones 10.0.1.0/24.
- **db_subnet**: diseñada para alojar servicios de base de datos, con rango 10.0.2.0/24. Esta subred incluyó la habilitación de service_endpoints para Microsoft.Sql, lo que permitió comunicaciones seguras entre los recursos internos y Azure SQL Database.

Figura 28:
Código para creación de subredes.

```
resource "azurerm_subnet" "app_subnet" {
  name                 = "app-subnet"
  resource_group_name  = var.resource_group_name
  virtual_network_name = azurerm_virtual_network.vnet.name
  address_prefixes     = ["10.0.1.0/24"]
}

# Subnet para base de datos
resource "azurerm_subnet" "db_subnet" {
  name                 = "db-subnet"
  resource_group_name  = var.resource_group_name
  virtual_network_name = azurerm_virtual_network.vnet.name
  address_prefixes     = ["10.0.2.0/24"]
  service_endpoints    = ["Microsoft.Sql"]  # Importante para conectar con
Azure SQL
}
```

Grupos de seguridad de red (NSG)

Se implementaron dos NSG diferenciados, con reglas específicas para proteger el acceso a los recursos, así como se muestra en la Figura 29**.**

NSG para la subred de aplicación (`app_nsg`)

Contenía dos reglas:

- **Allow-RDP**: permitió el acceso remoto mediante RDP (puerto 3389) exclusivamente desde una dirección IP especificada por el usuario mediante la variable admin_ip_address.
- **Allow-HTTP/HTTPS**: permitió tráfico entrante HTTP y HTTPS (puertos 80 y 443) desde cualquier origen, permitiendo así el acceso público a aplicaciones web hospedadas.

Figura 29:
Implementación del NSG para la aplicación.

```
# NSG para subnet de aplicación
resource "azurerm_network_security_group" "app_nsg" {
  name                = "app-nsg"
  location            = var.location
  resource_group_name = var.resource_group_name

  # Regla para RDP - limitar a una IP específica de administración
  security_rule {
    name                       = "Allow-RDP"
    priority                   = 100
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "3389"
    source_address_prefix      = var.admin_ip_address # Variable para IP de
administración
    destination_address_prefix = "*"
  }

  # Regla para HTTP/HTTPS
  security_rule {
    name                       = "Allow-HTTP"
    priority                   = 110
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_ranges    = ["80", "443"]
    source_address_prefix      = "*"
    destination_address_prefix = "*"
  }
```

```
  tags = {
    environment = var.environment
  }
}
```

NSG para la subred de base de datos (`db_nsg`)

Este NSG incluyó una única regla que permitía tráfico entrante al puerto 1433 (SQL) solo desde la subred de aplicación (10.0.1.0/24), aplicando el principio de mínimo privilegio para proteger el acceso a la base de datos. La implementación de esta regla se muestra en la Figura 30.

Figura 30:
NSG para la subred de base de datos.

```
# NSG para subnet de base de datos
resource "azurerm_network_security_group" "db_nsg" {
  name                = "db-nsg"
  location            = var.location
  resource_group_name = var.resource_group_name

  # Permitir tráfico SQL solo desde la subnet de aplicación
  security_rule {
    name                       = "Allow-SQL-From-App"
    priority                   = 100
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "1433"
    source_address_prefix      = "10.0.1.0/24"  # La subnet de la aplicación
    destination_address_prefix = "*"
  }

  tags = {
    environment = var.environment
  }
}
```

Asociaciones NSG a subredes

Cada NSG fue vinculado a su correspondiente subred utilizando el recurso azurerm_subnet_network_security_group_association. Esta acción aplicó de forma efectiva las reglas definidas, garantizando un comportamiento de red seguro y controlado, así como se muestra en la Figura 31.

Figura 31:
Asociación de NSG a subredes.

```
# Asociar NSG a la subnet de aplicación
resource "azurerm_subnet_network_security_group_association"
"app_nsg_association" {
  subnet_id                 = azurerm_subnet.app_subnet.id
  network_security_group_id = azurerm_network_security_group.app_nsg.id
}

# Asociar NSG a la subnet de base de datos
resource "azurerm_subnet_network_security_group_association"
"db_nsg_association" {
  subnet_id                 = azurerm_subnet.db_subnet.id
  network_security_group_id = azurerm_network_security_group.db_nsg.id
}
```

b. Archivo variables.tf del módulo network

En este archivo se declararon las variables de entrada requeridas por el módulo, así como se muestra en la Figura 32.

Estas incluyeron:

- **resource_group_name**: nombre del grupo de recursos donde se desplegaron los componentes de red.
- **location**: región de Azure en la que se aprovisionaron los recursos.
- **environment**: etiqueta descriptiva del entorno (dev, test, prod).
- **admin_ip_address**: dirección IP pública o rango desde donde se permitió el acceso remoto vía RDP a la máquina virtual.

Figura 32:
Variables del módulo network

```
variable "resource_group_name" {
  description = "Nombre del grupo de recursos"
  type        = string
}

variable "location" {
  description = "Ubicación de Azure para los recursos"
  type        = string
}

variable "environment" {
  description = "Entorno (dev, test, prod)"
  type        = string
  default     = "dev"
}

variable "admin_ip_address" {
  description = "Dirección IP desde donde se permitirá acceso RDP"
```

```
  type        = string
  default     = "0.0.0.0/0"  # Cambia esto a tu IP específica
}
```

Esta configuración mejoró la portabilidad del módulo y permitió mayor control de acceso al entorno de administración, promoviendo prácticas seguras desde la etapa de despliegue.

c. Archivo outputs.tf del módulo network

Se definieron salidas clave del módulo para permitir su reutilización y conexión con otros módulos, el código utilizado se muestra en la Figura 33.

- **vnet_id** y **vnet_name**: identificador y nombre de la red virtual principal, necesarios para vinculación de recursos.
- **app_subnet_id** y **db_subnet_id**: IDs de las subredes de aplicación y base de datos, respectivamente. Estas salidas fueron críticas para instanciar máquinas virtuales o bases de datos en subredes específicas.
- **app_nsg_id** y **db_nsg_id**: IDs de los grupos de seguridad de red para la aplicación y la base de datos, útiles si se necesitaban aplicar reglas o inspeccionar seguridad desde otros módulos o herramientas.

Figura 33:
Outputs del módulo network

```
output "vnet_id" {
  description = "ID de la red virtual"
  value       = azurerm_virtual_network.vnet.id
}

output "vnet_name" {
  description = "Nombre de la red virtual"
  value       = azurerm_virtual_network.vnet.name
}

output "app_subnet_id" {
  description = "ID de la subnet de aplicación"
  value       = azurerm_subnet.app_subnet.id
}

output "db_subnet_id" {
  description = "ID de la subnet de base de datos"
  value       = azurerm_subnet.db_subnet.id
}

output "app_nsg_id" {
  description = "ID del NSG de aplicación"
  value       = azurerm_network_security_group.app_nsg.id
```

```
}

output "db_nsg_id" {
  description = "ID del NSG de base de datos"
  value       = azurerm_network_security_group.db_nsg.id
}
```

Estas salidas facilitaron la interoperabilidad entre módulos**,** asegurando una integración fluida y automatizada dentro del entorno definido por Terraform.

2.2.4.7. Módulo VM: Creación de Máquina Virtual en Azure

Este módulo tuvo como objetivo automatizar el aprovisionamiento de una máquina virtual con sistema operativo **Windows Server 2022** en **Microsoft Azure**. Para ello, se definieron y desplegaron recursos relacionados como una dirección IP pública, una interfaz de red, un disco del sistema operativo y una extensión para la instalación automática del servidor web IIS. Todo se implementó siguiendo buenas prácticas de Infraestructura como Código (IaC)**,** permitiendo entornos reproducibles, seguros y mantenibles.

a. Archivo main.tf (módulo de vm)

En este archivo se configuró una dirección IP pública para la máquina virtual (VM), la cual fue necesaria para permitir el acceso externo a la misma. Esta IP se definió con asignación estática y utilizando el SKU “Standard”, más adecuado para entornos productivos por su mayor resiliencia. El código correspondiente se presenta en la Figura 34**.**

Figura 34:
Asignación de IP pública estática para la VM

```
# IP Pública para la VM
resource "azurerm_public_ip" "vm_pip" {
  name                = "${var.vm_name}-pip"
  location            = var.location
  resource_group_name = var.resource_group_name
  allocation_method   = "Static"
  sku                 = "Standard"  # Mejor para entornos de producción

  tags = {
    environment = var.environment
  }
}
```

Posteriormente, se creó la interfaz de red (NIC) asociada a la VM, donde se enlazó la IP pública definida previamente, y se configuró la asignación dinámica de la IP privada dentro de una subred específica. El fragmento de código mostrado en la Figura 35 refleja esta configuración.

Figura 35:
Creación y configuración de la interfaz de red para la VM.

```
# Interfaz de red para la VM
resource "azurerm_network_interface" "vm_nic" {
  name                = "${var.vm_name}-nic"
  location            = var.location
  resource_group_name = var.resource_group_name

  ip_configuration {
    name                          = "${var.vm_name}-ipconfig"
    subnet_id                     = var.subnet_id  # Usar la subnet pasada
como parámetro
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id          = azurerm_public_ip.vm_pip.id
  }
}
```

A continuación, se aprovisionó la máquina virtual como tal, utilizando la imagen de Windows Server 2022 Datacenter. Se especificaron sus parámetros clave como nombre, tamaño, credenciales administrativas, disco OS y la NIC asociada. También se optó por un disco premium (Premium_LRS) para garantizar mejor rendimiento. El código correspondiente se presenta en la Figura 36.

Figura 36:
Aprovisionamiento de la máquina virtual Windows Server 2022.

```
# Máquina virtual Windows Server 2022
resource "azurerm_windows_virtual_machine" "vm" {
  name                = var.vm_name
  location            = var.location
  resource_group_name = var.resource_group_name
  size                = var.vm_size
  admin_username      = var.admin_username
  admin_password      = var.admin_password
  network_interface_ids = [azurerm_network_interface.vm_nic.id]

  os_disk {
    name                 = "${var.vm_name}-osdisk"
    caching              = "ReadWrite"
    storage_account_type = "Premium_LRS"  # Mejor rendimiento para producción
  }

  source_image_reference {
    publisher = "MicrosoftWindowsServer"
    offer     = "WindowsServer"
    sku       = "2022-Datacenter"
    version   = "latest"
  }
```

```
  tags = {
    environment = var.environment
  }
}
```

Finalmente, se utilizó una extensión de script personalizado (CustomScriptExtension) para automatizar la instalación del servidor web IIS en la VM. Esta extensión ejecutó un script en PowerShell descargado desde un repositorio remoto de GitHub. El fragmento de código mostrado en la Figura 37 corresponde a esta configuración adicional.

Figura 37:
Configuración de la extensión para instalar IIS script personalizado.

```
# Extensión de script personalizado para configurar IIS (opcional)

resource "azurerm_virtual_machine_extension" "iis_extension" {
  name                 = "${var.vm_name}-iis-extension"
  virtual_machine_id   = azurerm_windows_virtual_machine.vm.id
  publisher            = "Microsoft.Compute"
  type                 = "CustomScriptExtension"
  type_handler_version = "1.10"

 settings = jsonencode({
    fileUris = [
      "https://raw.githubusercontent.com/LINK-
DANNY/scriptSetup/main/setup.ps1"
    ]
    commandToExecute = "powershell -ExecutionPolicy Unrestricted -File
setup.ps1 -gitToken '${var.git_pat}'"
  })

  tags = {
    environment = var.environment
  }
}
```

b. Archivo variables.tf (módulo de vm)

Este archivo definió los parámetros de entrada requeridos por el módulo, los cuales permitieron su reutilización en diferentes entornos. El código correspondiente se presenta en la Figura 38.

Figura 38:
Variables del módulo de vm

```
variable "resource_group_name" {
  description = "Nombre del grupo de recursos"
```

```
  type        = string
}

variable "location" {
  description = "Ubicación de Azure para los recursos"
  type        = string
}

variable "vm_name" {
  description = "Nombre de la máquina virtual"
  type        = string
}

variable "vm_size" {
  description = "Tamaño de la máquina virtual"
  type        = string
  default     = "Standard_D2s_v3"
}

variable "admin_username" {
  description = "Nombre de usuario administrador"
  type        = string
}

variable "admin_password" {
  description = "Contraseña de administrador"
  type        = string
  sensitive   = true
}

variable "subnet_id" {
  description = "ID de la subnet donde se desplegará la VM"
  type        = string
}

variable "environment" {
  description = "Entorno (dev, test, prod)"
  type        = string
  default     = "dev"
}

variable "bdd_subnet_id" {
  description = "ID de la subnet de la base de datos"
  type        = string
}
```

- resource_group_name, location: información de despliegue general.

- vm_name, vm_size: configuración de nombre y tamaño de la VM.
- admin_username, admin_password: credenciales de administrador.
- subnet_id: ID de la subnet donde se desplegó la VM.
- bdd_subnet_id: ID de la subnet de base de datos, en caso de necesitar interacción entre ambos entornos.
- environment: utilizado para etiquetas y segmentación lógica del entorno.

c. Archivo outputs.tf

Este archivo expuso los valores de salida relevantes del módulo, útiles para otros módulos o para validaciones posteriores. El código correspondiente se presenta en la Figura 39.

Figura 39:
Salidas del módulo de vm

```
output "vm_id" {
  description = "ID de la máquina virtual"
  value       = azurerm_windows_virtual_machine.vm.id
}

output "vm_name" {
  description = "Nombre de la máquina virtual"
  value       = azurerm_windows_virtual_machine.vm.name
}

output "vm_public_ip" {
  description = "Dirección IP pública de la máquina virtual"
  value       = azurerm_public_ip.vm_pip.ip_address
}

output "vm_private_ip" {
  description = "Dirección IP privada de la máquina virtual"
  value       = azurerm_network_interface.vm_nic.private_ip_address
}

output "vm_nic_id" {
  description = "ID de la interfaz de red de la máquina virtual"
  value       = azurerm_network_interface.vm_nic.id
}
```

- **vm_id**, **vm_name**: identificador y nombre de la VM.
- **vm_public_ip**: IP pública asignada, clave para RDP o acceso web.
- **vm_private_ip**: IP interna dentro de la VNet.
- **vm_nic_id**: identificador de la interfaz de red, útil para diagnósticos o futuras asociaciones.

Este módulo fue diseñado siguiendo buenas prácticas de infraestructura en la nube, como la segmentación por subredes**,** la definición explícita de recursos y el uso de etiquetas. La automatización de la instalación de IIS agilizó la preparación de entornos de prueba web, reduciendo tiempos y errores en el despliegue.

2.2.4.8. Módulo bdd: Aprovisionamiento de Base de Datos SQL en Azure

Este módulo se encarga de automatizar el aprovisionamiento de un entorno seguro y optimizado para una base de datos SQL en Azure. Incluye la creación del servidor SQL, la base de datos, reglas de acceso (firewall y red virtual), así como la configuración de un endpoint privado. Se aplican buenas prácticas de seguridad, integración en red y eficiencia en costos mediante recursos *serverless*.

a. Archivo main.tf

1. Servidor SQL (azurerm_mssql_server.sql_server)

Se crea un servidor SQL en Azure con las siguientes características:

- Versión 12.0 (actualmente la única disponible para Azure SQL).
- Autenticación mediante nombre de usuario y contraseña, definidos como variables.
- Requiere como mínimo el uso de TLS 1.2 para garantizar conexiones seguras.
- Se etiquetan los recursos según el entorno (dev, test, prod).

El código correspondiente se presenta en la Figura 40.

Figura 40:
Creación del servidor SQL en Azure.

```
# Servidor SQL
resource "azurerm_mssql_server" "sql_server" {
  name                         = var.sql_server_name
  resource_group_name          = var.resource_group_name
  location                     = var.location
  version                      = "12.0"
  administrator_login          = var.admin_username
  administrator_login_password = var.admin_password
  minimum_tls_version          = "1.2"  # Requiere TLS 1.2 mínimo por
seguridad

  tags = {
    environment = var.environment
  }
}
```

***2.* Base de Datos SQL (azurerm_mssql_database.sql_db)**

Se implementa una base de datos SQL en modalidad *Serverless*, ideal para entornos con cargas intermitentes. Sus características principales son:

- SKU GP_S_Gen5_2: 2 vCores, modalidad *General Purpose*.
- Tamaño máximo de 32 GB.
- Capacidad mínima de 0.5 vCores.
- Auto-pausa habilitada tras 60 minutos de inactividad.

El fragmento de código mostrado en la Figura 41 refleja esta implementación.

Figura 41:
Implementación de la base de datos SQL en modalidad Serverless.

```
# Base de datos SQL
resource "azurerm_mssql_database" "sql_db" {
  name           = var.database_name
  server_id      = azurerm_mssql_server.sql_server.id
  collation      = "SQL_Latin1_General_CP1_CI_AS"
  sku_name       = "GP_S_Gen5_2"  # General Purpose, Serverless, 2 vCores - 
mejor para prod
  max_size_gb    = 32

  # Configuración serverless para optimizar costos
  auto_pause_delay_in_minutes = 60
  min_capacity                = 0.5

  tags = {
    environment = var.environment
  }
}
```

3. Regla de Firewall (azurerm_mssql_firewall_rule.allow_azure_services)

Se habilita el acceso desde todos los servicios de Azure utilizando la IP especial `0.0.0.0`, lo cual facilita la comunicación entre recursos dentro de la misma suscripción. El código correspondiente se presenta en la Figura 42.

Figura 42:
Regla de firewall para permitir acceso desde servicios de Azure.

```
# Regla de firewall para permitir acceso desde Azure
resource "azurerm_mssql_firewall_rule" "allow_azure_services" {
  name             = "AllowAzureServices"
  server_id        = azurerm_mssql_server.sql_server.id
  start_ip_address = "0.0.0.0"
  end_ip_address   = "0.0.0.0"
}
```

4. Regla de Red Virtual (azurerm_mssql_virtual_network_rule.allow_vm_subnet)

Permite el acceso a la base de datos únicamente desde la subred de la máquina virtual, brindando un control de acceso basado en red. El código utilizado se muestra en la Figura 43.

Figura 43:
Regla de red virtual que permite el acceso desde la subred de la VM.

```
# Regla para permitir acceso desde la subnet de la VM
resource "azurerm_mssql_virtual_network_rule" "allow_vm_subnet" {
  name      = "allow-vm-subnet"
  server_id = azurerm_mssql_server.sql_server.id
  subnet_id = var.vm_subnet_id  # Usar la subnet de la VM pasada como
parámetro
}
```

5. Endpoint Privado (azurerm_private_endpoint.sql_pe)

Se implementó un endpoint privado para permitir el acceso interno y seguro a la base de datos dentro de la red virtual. Este se asoció a la subred correspondiente y utilizó conexión automática al servidor SQL. El fragmento de código correspondiente se presenta en la Figura 44.

Figura 44:
Configuración del endpoint privado para acceso seguro a SQL Server.

```
# Endpoint privado para acceso seguro a la base de datos
resource "azurerm_private_endpoint" "sql_pe" {
  name                = "${var.sql_server_name}-endpoint"
  location            = var.location
  resource_group_name = var.resource_group_name
  subnet_id           = var.subnet_id  # Usar la subnet de base de datos

  private_service_connection {
    name                           = "${var.sql_server_name}-
privateserviceconnection"
    private_connection_resource_id = azurerm_mssql_server.sql_server.id
    subresource_names              = ["sqlServer"]
    is_manual_connection           = false
  }

  tags = {
    environment = var.environment
  }
}
```

b. Archivo variables.tf

Este módulo recibe parámetros fundamentales como:

- Datos generales de despliegue (resource_group_name, location, environment).
- Configuración del servidor y base de datos (sql_server_name, database_name, admin_username, admin_password).
- Identificadores de subredes necesarias para definir las reglas de red y endpoints (subnet_id para base de datos y vm_subnet_id para acceso desde la VM).

La definición de estas variables se muestra en la Figura 45.

Figura 45:
Definición de variables necesarias para el despliegue del servidor y base de datos SQL.

```
variable "resource_group_name" {
  description = "Nombre del grupo de recursos"
  type        = string
}

variable "location" {
  description = "Ubicación de Azure para los recursos"
  type        = string
}

variable "sql_server_name" {
  description = "Nombre del servidor SQL"
  type        = string
}

variable "database_name" {
  description = "Nombre de la base de datos"
  type        = string
}

variable "admin_username" {
  description = "Nombre de usuario administrador de SQL"
  type        = string
}

variable "admin_password" {
  description = "Contraseña de administrador de SQL"
  type        = string
  sensitive   = true
}

variable "subnet_id" {
  description = "ID de la subnet donde se desplegará el endpoint privado de
SQL"
  type        = string
}

variable "vm_subnet_id" {
  description = "ID de la subnet donde está la VM"
  type        = string
}

variable "environment" {
  description = "Entorno (dev, test, prod)"
  type        = string
```

```
  default     = "dev"
}
```

c. Archivo outputs.tf

Se definieron varias salidas útiles para otros módulos dependientes o para pruebas y documentación, tales como identificadores, nombres y cadena de conexión con cifrado activado. Estas salidas se detallan en la Figura 46.

- **sql_server_id, sql_server_name, sql_server_fqdn**: detalles básicos del servidor SQL, incluyendo su FQDN (utilizable en cadenas de conexión).
- **database_id, database_name**: identificadores y nombre de la base de datos.
- **connection_string**: cadena de conexión SQL completa con encriptación habilitada. Marcada como sensible para evitar su exposición en la consola de Terraform.
- **private_endpoint_ip**: dirección IP privada del endpoint, usada para diagnóstico o configuración adicional dentro de la VNet.

Figura 46:
Definición de salidas para reutilización e integración del módulo SQL.

```
output "sql_server_id" {
  description = "ID del servidor SQL"
  value       = azurerm_mssql_server.sql_server.id
}

output "sql_server_name" {
  description = "Nombre del servidor SQL"
  value       = azurerm_mssql_server.sql_server.name
}

output "sql_server_fqdn" {
  description = "Nombre de dominio completo del servidor SQL"
  value       = azurerm_mssql_server.sql_server.fully_qualified_domain_name
}

output "database_id" {
  description = "ID de la base de datos SQL"
  value       = azurerm_mssql_database.sql_db.id
}

output "database_name" {
  description = "Nombre de la base de datos SQL"
  value       = azurerm_mssql_database.sql_db.name
}

output "connection_string" {
  description = "Cadena de conexión a la base de datos SQL"
```

```
  value       =
"Server=${azurerm_mssql_server.sql_server.fully_qualified_domain_name};Databas
e=${azurerm_mssql_database.sql_db.name};User
ID=${var.admin_username};Password=${var.admin_password};Encrypt=True;"
  sensitive   = true  # Marca como sensible para que no se muestre en los logs
}

output "private_endpoint_ip" {
  description = "IP privada del endpoint de la base de datos"
  value       =
azurerm_private_endpoint.sql_pe.private_service_connection[0].private_ip_addre
ss
}
```

Este módulo está diseñado bajo principios de seguridad, modularidad y eficiencia en la nube. Se evita el acceso público innecesario mediante el uso de endpoints privados, se segmentan correctamente las subredes y se utiliza la modalidad serverless para optimizar costos en entornos de desarrollo o baja carga. Además, permite una integración fluida con otros módulos, como el de máquinas virtuales.

Para realizar la validación del entorno se desplegó una máquina virtual con Windows Server 2022 utilizando la infraestructura definida mediante Terraform, y se instaló IIS. Esta solución permitió validar la correcta provisión de la infraestructura, la conectividad entre los servicios, y la posibilidad de desplegar aplicaciones que utilicen bases de datos SQL. Asimismo, se comprobó que, incluso ante fallos controlados, el sistema es capaz de recuperar el servicio con mínima intervención manual gracias al diseño de alta disponibilidad. En particular, se espera que el entorno sea capaz de: detectar automáticamente el fallo, promover la base secundaria como nueva principal, restaurar una base secundaria a partir de esta nueva principal, restablecer la geo-replicación de forma automatizada y sincronizar la nueva topología con el estado definido en Terraform. En el siguiente capítulo se documentan las pruebas realizadas, los fallos inducidos y los resultados obtenidos en cuanto a tiempos de restauración y comportamiento del entorno.

CÁPITULO III
RESULTADOS Y DISCUSIÓN

En este capítulo se presentan las pruebas prácticas realizadas para verificar el comportamiento del entorno ante escenarios de fallos, así como los tiempos de restauración mediante despliegues automatizados. El objetivo fue evaluar la resiliencia de la infraestructura implementada, validando que los recursos definidos mediante IaC (Infrastructure as Code) puedan ser reproducidos de forma confiable tras un incidente.

Para ello, se realizaron pruebas que incluyeron la eliminación controlada de ciertos recursos (como la máquina virtual o la base de datos), y se observó la capacidad del sistema para recuperar el entorno mediante una nueva ejecución de los scripts de despliegue. Adicionalmente, se registraron los tiempos de provisión y restauración, y se verificó que la aplicación continuara funcionando correctamente tras la recuperación. Para llevar esta prueba se siguieron los siguientes puntos

3.1. Actividades realizadas para ejecutar las pruebas.

3.1.1. Inicialización y Validación del Proyecto Terraform

El primer paso consistió en inicializar el proyecto mediante el comando terraform init, el cual descarga los proveedores necesarios y configura el entorno local. Posteriormente, se ejecutó terraform validate para comprobar que la sintaxis de los archivos .tf fuese correcta.

Para visualizar los cambios que se aplicarían, se utilizó terraform plan, generando así un archivo de plan (tfplan) que posteriormente fue aplicado mediante el comando terraform apply -auto-approve tfplan, lo cual permitió el despliegue completo de la infraestructura definida.

Estos pasos fueron automatizados a través del script por lotes deploy.bat, que contiene las siguientes instrucciones. El código utilizado se muestra en la Figura 47.

Figura 47:
Script deploy.bat

```
@echo off
echo Iniciando despliegue de infraestructura...

:: Inicializa Terraform
terraform init

:: Valida sintaxis (opcional pero útil)
terraform validate

:: Planifica y guarda el plan
terraform plan -out=tfplan

:: Aplica el plan automáticamente
```

```
terraform apply -auto-approve tfplan

echo Despliegue completado.
pause
```

Resultado del Despliegue Inicial

La ejecución del script anterior resultó en la creación automática de los recursos definidos en los archivos de configuración Terraform, entre los que se incluyen:

- Un grupo de recursos (Resource Group).
- Una instancia de **Azure SQL Database** con geo-replicación configurada.
- Una máquina virtual Windows Server para pruebas de despliegue de la aplicación.
- Elementos de red asociados (vNet, NSG, IP pública, etc.).

La Figura 48 muestra la consola cmd durante la ejecución exitosa del despliegue. En la Figura 49 se presenta la vista del portal de Azure, en la cual se verifica la existencia de todos los recursos desplegados, validando así el éxito del proceso.

Figura 48:
Ejecución del script deploy.bat

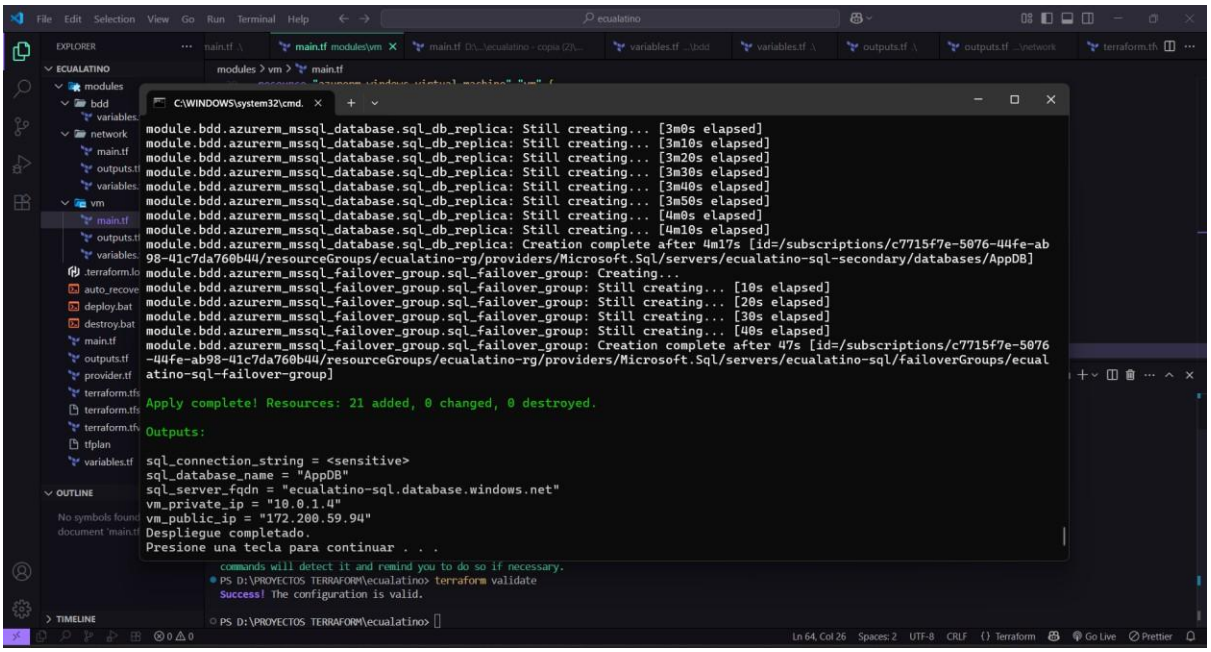
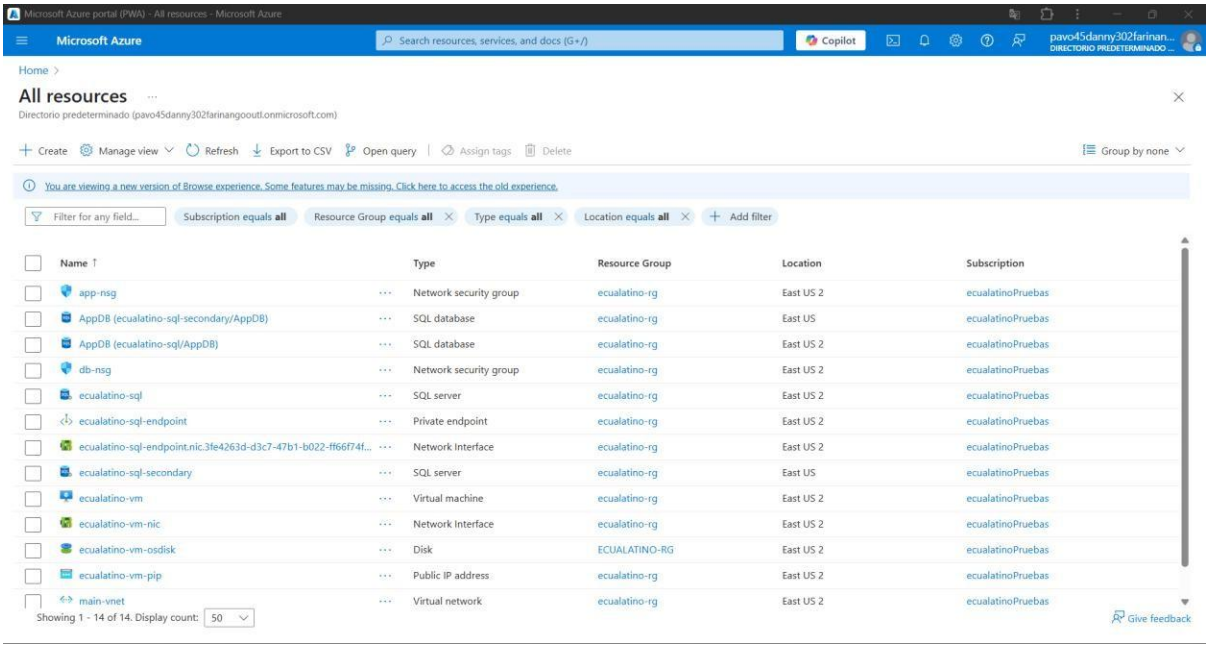


Figura 49:
Visualización de los recursos desplegados en Azure Portal.



3.1.2. Destrucción Controlada de Infraestructura

Para fines de pruebas y simulaciones posteriores, también se creó un segundo script por lotes llamado **`destroy.bat`**, el cual automatiza la eliminación de toda la infraestructura de forma controlada. Su contenido es el siguiente que se presenta en la figura 50:

Figura 50:
Script detroy.bat

```
@echo off
echo Destruyendo infraestructura...

:: Inicializa Terraform por si acaso
terraform init

:: Destruye todo sin pedir confirmación
terraform destroy -auto-approve

echo Infraestructura destruida.
pause
```

Este script permite liberar los recursos en Azure y evitar costos innecesarios durante fases de prueba o simulación de recuperación. La ejecución de `terraform destroy` elimina todos los componentes definidos previamente sin intervención manual, asegurando un entorno limpio y controlado para cada experimento.

La ejecución del script resultó en la eliminación completa de todos los recursos creados en Azure. Esto se evidencia en la Figura 51, donde se muestra el mensaje final de éxito generado por la consola de cmd al concluir el proceso. Además, la Figura 52 presenta la vista del portal de Azure luego de la destrucción, en la cual se puede comprobar que ya no existen recursos activos dentro del grupo de recursos originalmente creado.

Figura 51:
Mensaje de éxito tras ejecutar el script destroy.bat.

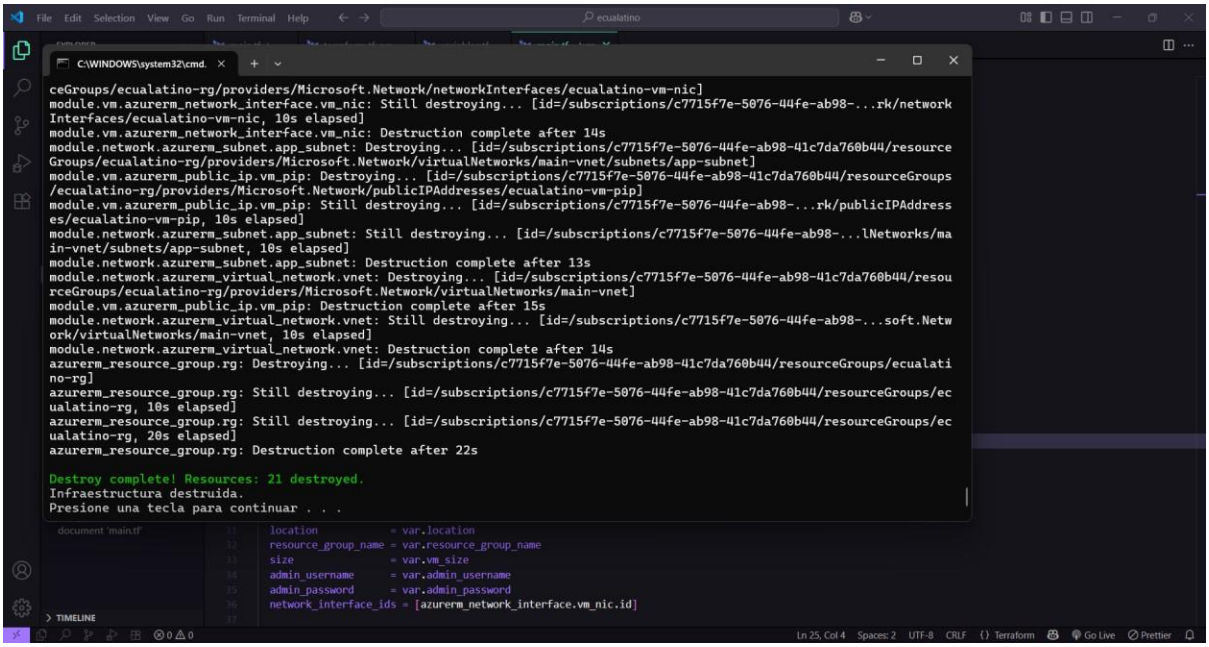
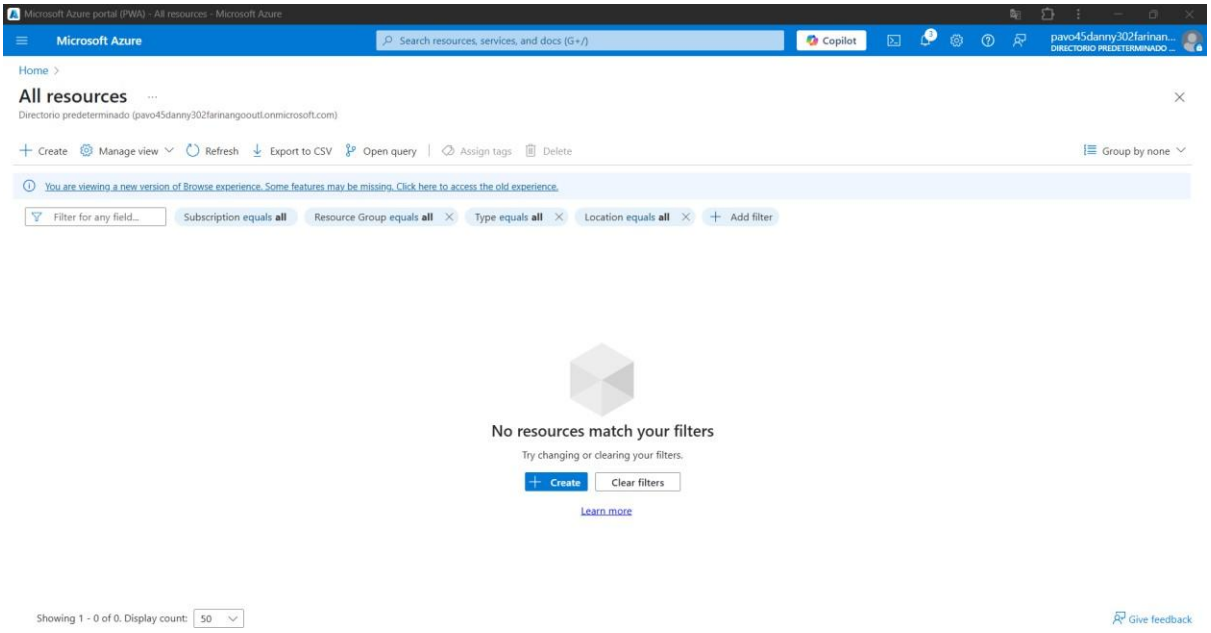


Figura 52:
Vista del portal de Azure sin recursos tras la eliminación.



3.1.3. Despliegue de la Aplicación Web y Configuración del Entorno

Para validar el funcionamiento de la infraestructura, se desplegó manualmente una aplicación web de prueba llamada insuPro, la cual fue alojada en la máquina virtual creada previamente mediante Terraform. Esta aplicación permite simular un entorno real de funcionamiento conectado a una base de datos SQL en Azure.

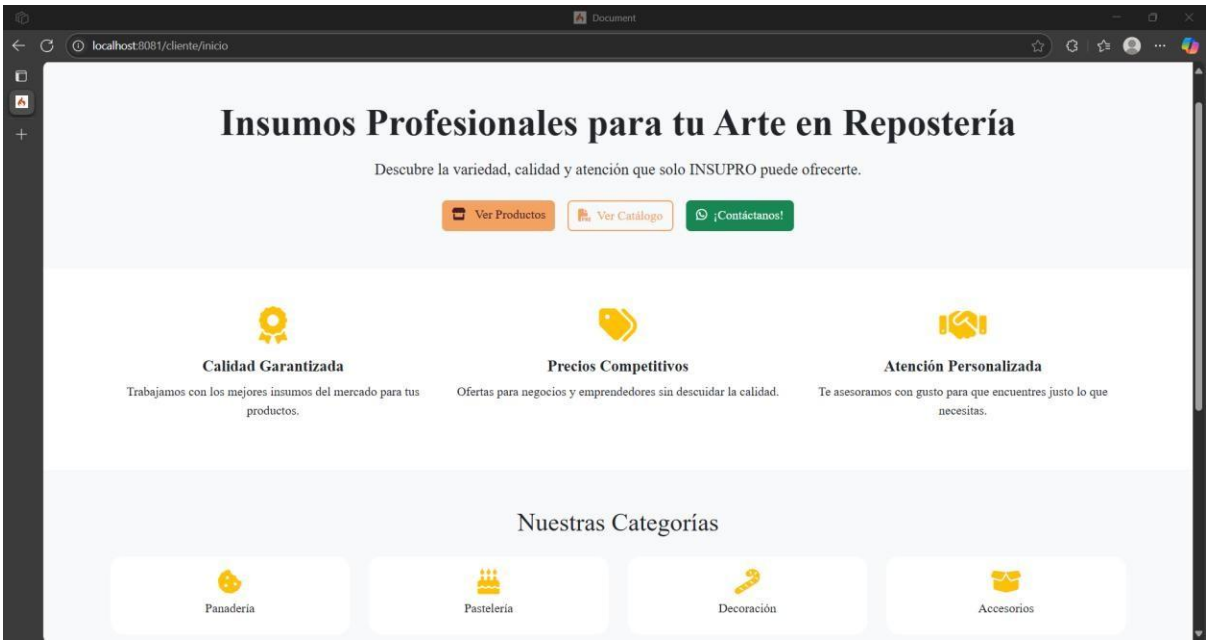
Configuración Manual del Entorno

El entorno fue configurado completamente de forma manual, accediendo directamente a la máquina virtual y realizando paso a paso la instalación y preparación del servidor web. Los pasos ejecutados fueron los siguientes:

- Ingreso remoto a la máquina virtual mediante el protocolo **RDP**.
- Instalación de **IIS (Servidor Web)** desde las características de Windows.
- Descarga e instalación de **PHP** desde su sitio oficial.
- Configuración de **FastCGI** en IIS para permitir la ejecución de archivos .php.
- Copia directa del proyecto **insuPro** a la carpeta de publicación web, configurada para el puerto **8081**.
- Verificación del correcto funcionamiento desde el navegador.

La Figura 53 muestra la ventana del navegador accediendo localmente a la aplicación, lo cual confirma que el entorno fue configurado exitosamente.

Figura 53:
Aplicación insuPro desplegada y funcionando en la máquina virtual

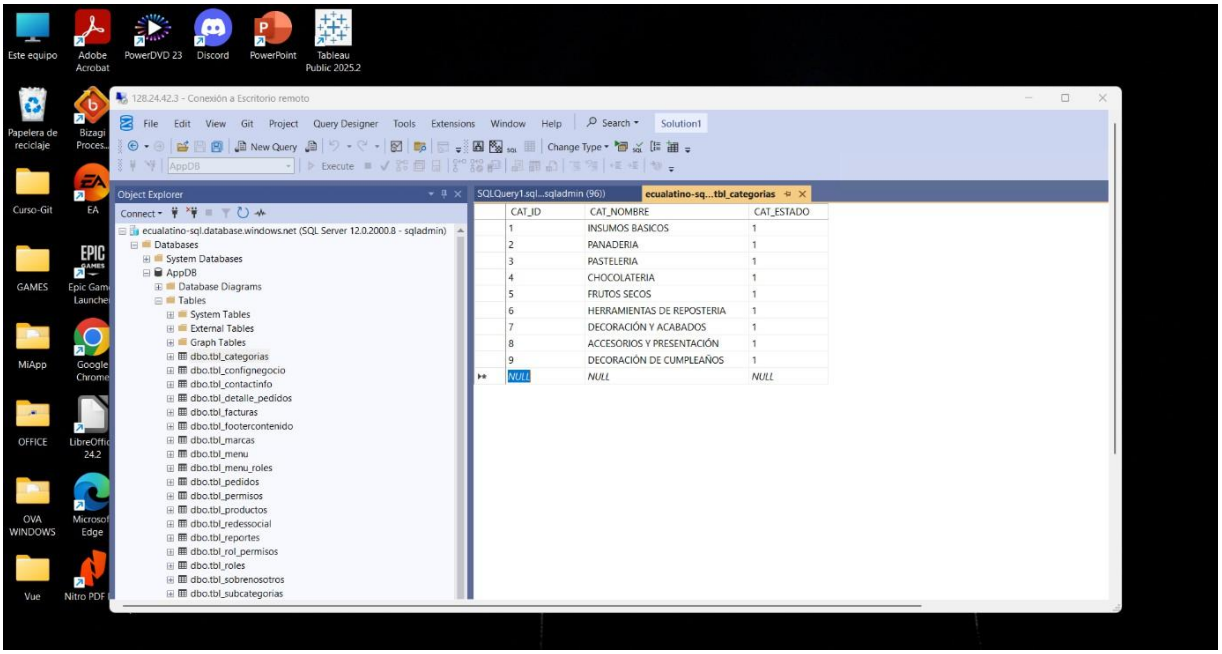


Subida Manual de la Base de Datos SQL

Para garantizar el funcionamiento completo de la aplicación, también se realizó la **subida manual de la base de datos** al servidor SQL de Azure. Esta base ya estaba previamente preparada y fue cargada directamente al entorno, permitiendo así que la aplicación pudiera conectarse correctamente desde el primer momento. La Figura 54 muestra la ventana de la base de datos lo cual confirma que el entorno fue configurado exitosamente.

La conexión con la base de datos fue configurada en el archivo correspondiente de la aplicación PHP, permitiendo establecer la comunicación con el servidor SQL en la nube.

Figura 54:
Base de Datos SQL SERVER



3.1.4. Simulación de Falla: Eliminación de la Base de Datos Principal

Con el objetivo de validar los mecanismos de recuperación y continuidad operativa implementados mediante redundancia geográfica, se diseñó una prueba controlada de fallo sobre la base de datos principal del sistema.

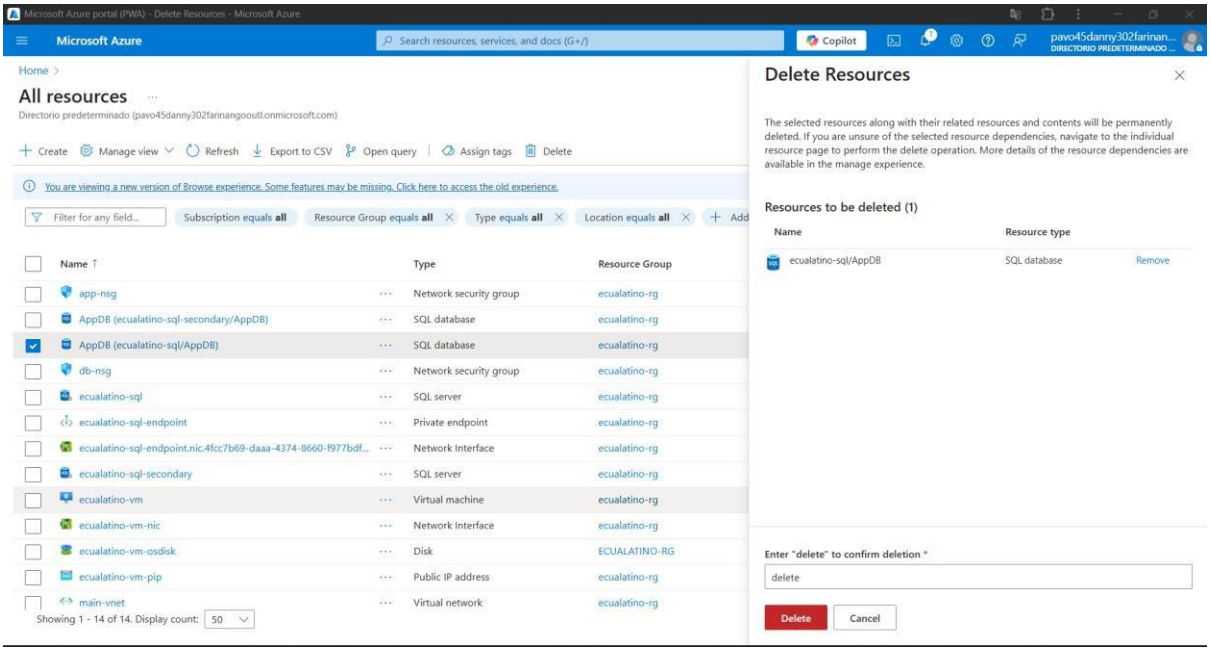
Simulación de la Falla

La eliminación de la base de datos principal se realizó manualmente desde el portal de Azure, eliminando el recurso AppDB alojado en el servidor ecualatino-sql. Esta acción simuló una caída total y repentina del servicio de base de datos principal.

Como consecuencia, la aplicación web dejó de funcionar correctamente al perder la conexión con la base principal. Esta situación permitió validar los mecanismos de detección y recuperación automática implementados en el entorno.

La Figura 55 muestra la interfaz del portal Azure con la base de datos AppDB eliminada y la ausencia del recurso en el servidor principal.

Figura 55:
Eliminación de la base principal



Script de Recuperación Automática

Para automatizar la recuperación y minimizar tiempos de indisponibilidad, se desarrolló un script batch denominado `auto_recovery_ecualatino.bat`, cuyo objetivo es detectar la ausencia de la base de datos principal, promover la secundaria a primaria, recrear la geo-replicación y sincronizar el estado con Terraform.

El script realiza las siguientes acciones principales:

1. Verifica la existencia y estado de las bases de datos en los servidores principal y secundario.
2. En caso de ausencia de la base principal, promueve la base secundaria como primaria.
3. Crea una nueva base principal a partir de una copia de la base secundaria.
4. Elimina la base secundaria para recrear la geo-replicación.
5. Recrea la geo-replicación entre los servidores.
6. Sincroniza la infraestructura mediante Terraform para reflejar los cambios.

El usuario puede elegir si desea proceder con la recuperación automática o cancelarla.

A continuación, se presenta el contenido completo del script en la figura 56.

Figura 56:
Script de recuperación automática

```
@echo off
```

```
REM auto_recovery_ecualatino.bat - Script específico para recuperacion de
Ecualatino
setlocal enabledelayedexpansion
cd /d "%~dp0"

REM --- Capturar y mostrar hora de inicio ---
for /f "tokens=1-2 delims= " %%a in ('echo %time%') do set START_TIME=%%a %%b

echo.
echo ====================================
echo  Ecualatino Database Recovery Tool
echo Iniciado a las: %START_TIME%
echo ====================================
echo.

REM Variables específicas de la configuracion
set RESOURCE_GROUP_NAME=ecualatino-rg
set PRIMARY_SERVER_NAME=ecualatino-sql
set SECONDARY_SERVER_NAME=ecualatino-sql-secondary
set DATABASE_NAME=AppDB
set "TERRAFORM_DIR=D:\PROYECTOS TERRAFORM\ecualatino"

REM Obtener subscription ID
for /f "tokens=*" %%a in ('az account show --query id -o tsv') do set
SUBSCRIPTION_ID=%%a

echo.
echo  Verificando estado actual de las bases de datos...

REM Funcion para verificar si existe una base de datos
call :check_database_exists %PRIMARY_SERVER_NAME% %DATABASE_NAME%
%RESOURCE_GROUP_NAME% PRIMARY_EXISTS
call :check_database_exists %SECONDARY_SERVER_NAME% %DATABASE_NAME%
%RESOURCE_GROUP_NAME% SECONDARY_EXISTS

REM Funcion para obtener el estado de la base de datos
call :get_database_status %PRIMARY_SERVER_NAME% %DATABASE_NAME%
%RESOURCE_GROUP_NAME% PRIMARY_STATUS
call :get_database_status %SECONDARY_SERVER_NAME% %DATABASE_NAME%
%RESOURCE_GROUP_NAME% SECONDARY_STATUS

echo.
echo  Estado actual:
if defined PRIMARY_EXISTS (
    echo    Base Principal ^(%PRIMARY_SERVER_NAME%^): %PRIMARY_EXISTS%
    echo      Estado: !PRIMARY_STATUS!
) else (
    echo    Base Principal ^(%PRIMARY_SERVER_NAME%^):  NO EXISTE
```

```
)

if defined SECONDARY_EXISTS (
    echo    Base Secundaria ^(%SECONDARY_SERVER_NAME%^): %SECONDARY_EXISTS%
    echo      Estado: !SECONDARY_STATUS!
) else (
    echo    Base Secundaria ^(%SECONDARY_SERVER_NAME%^): NO EXISTE
)

echo.
echo  Analizando situacion...

REM Determinar accion necesaria
if not defined PRIMARY_EXISTS if defined SECONDARY_EXISTS (
    REM CASO 1: Principal eliminada, secundaria existe
    echo.
    echo  SITUACIoN DE RECUPERACIoN DETECTADA
    echo  Escenario: Base principal eliminada pero secundaria existe con datos
    echo.
    echo ¿Que va a pasar?
    echo 1. La base secundaria sera promovida a independiente
    echo 2. Se creara una nueva base principal copiando los datos del backup
    echo 3. Se recreara la geo-replica
    echo 4. Terraform sera sincronizado con el nuevo estado
    echo.

    set /p REPLY="¿Deseas proceder con la recuperacion automatica? (y/N): "
    if /i "!REPLY!"=="y" (
        echo.
        set "RECOVERY_START=%TIME%"
        echo  Iniciando proceso de recuperacion a las %RECOVERY_START%...

        REM Paso 1: Promover secundaria a independiente
        echo.
        echo 1 Promoviendo base secundaria a independiente...
        az sql db replica set-primary --name %DATABASE_NAME% --server
%SECONDARY_SERVER_NAME% --resource-group %RESOURCE_GROUP_NAME%
        if !errorlevel! neq 0 (
            echo   promoviendo base secundaria
            goto :error_exit
        )
        echo Base secundaria promovida exitosamente

        REM Paso 2: Crear nueva base principal
        echo.
        echo 2 Creando nueva base principal desde backup...
        echo     Copiando %DATABASE_NAME% desde %SECONDARY_SERVER_NAME% a
%PRIMARY_SERVER_NAME%
```

```
        az sql db copy --dest-name %DATABASE_NAME% --dest-server
%PRIMARY_SERVER_NAME% --name %DATABASE_NAME% --server %SECONDARY_SERVER_NAME%
--resource-group %RESOURCE_GROUP_NAME%
        if !errorlevel! neq 0 (
            echo    copiando base de datos
            goto :error_exit
        )
        echo  Comando de copia ejecutado

        REM Paso 3: Esperar que la copia termine
        echo.
        echo 3 Esperando que la copia termine...
        set WAIT_COUNT=0
        :wait_loop
         call :get_database_status %PRIMARY_SERVER_NAME% %DATABASE_NAME%
%RESOURCE_GROUP_NAME% CURRENT_STATUS
        if "!CURRENT_STATUS!"=="Online" (
            echo  Base principal está en línea y lista
            goto :continue_recovery
        )
        set /a WAIT_COUNT+=1
        echo  Estado actual: !CURRENT_STATUS! - Esperando...
^(!WAIT_COUNT!/20^)
        if !WAIT_COUNT! geq 20 (
            echo  La copia está tomando más tiempo del esperado
            set /p CONTINUE_REPLY="¿Continuar esperando? (y/N): "
            if /i not "!CONTINUE_REPLY!"=="y" goto :continue_recovery
            set WAIT_COUNT=0
        )
        timeout /t 30 /nobreak >nul
        goto :wait_loop

        :continue_recovery
        REM Paso 4: Eliminar base secundaria actual
        echo.
        echo 4 Preparando para recrear geo-replica...
        echo     Eliminando base secundaria actual para recrear como replica
        az sql db delete --name %DATABASE_NAME% --server
%SECONDARY_SERVER_NAME% --resource-group %RESOURCE_GROUP_NAME% --yes
        if !errorlevel! neq 0 (
            echo  Eliminando base secundaria, continuando...
        ) else (
            echo  Base secundaria eliminada
        )

        echo  Esperando 30 segundos para que Azure procese...
        timeout /t 30 /nobreak >nul
```

```
        REM Paso 5: Recrear geo-replica
        echo.
        echo  5 Recreando geo-replica...
        az sql db replica create --name %DATABASE_NAME% --server
%PRIMARY_SERVER_NAME% --resource-group %RESOURCE_GROUP_NAME% --partner-server
%SECONDARY_SERVER_NAME% --partner-resource-group %RESOURCE_GROUP_NAME%
        if !errorlevel! neq 0 (
            echo    Recreando geo-replica
            goto :error_exit
        )
        echo Geo-replica recreada

        REM Paso 6: Sincronizar con Terraform
        echo.
        echo 6 Sincronizando estado con Terraform...
        cd /d %TERRAFORM_DIR%
        echo     Aplicando configuracion...
        terraform init
        terraform apply -auto-approve

    REM Capturar hora de finalizacion de la recuperación
    for /f "tokens=1-2 delims= " %%a in ('echo %time%') do set
RECOVERY_END=%%a %%b
    echo.
    echo 🕒 Recuperación iniciada a las !RECOVERY_START! y finalizada a las
!RECOVERY_END!
    echo.
    pause

    ) else (
        echo ⏸ Recuperacion cancelada por el usuario
        goto :end
    )

) else if defined PRIMARY_EXISTS if defined SECONDARY_EXISTS (
    REM CASO 2: Estado normal
    echo  Estado normal - Ambas bases de datos existen
    echo  Ejecutando terraform apply para mantener sincronizacion...
    cd /d %TERRAFORM_DIR%
    terraform apply -auto-approve

) else if defined PRIMARY_EXISTS if not defined SECONDARY_EXISTS (
    REM CASO 3: Solo existe principal
    echo  Solo existe la principal - Recreando secundaria
    echo  Ejecutando terraform apply para recrear la geo-replica...
    cd /d %TERRAFORM_DIR%
    terraform apply -auto-approve
```

```
) else (
    REM CASO 4: Ninguna existe
    echo  Ninguna base existe - Ejecutando creacion inicial
    echo  Ejecutando terraform apply para crear toda la infraestructura...
    cd /d %TERRAFORM_DIR%
    terraform apply -auto-approve
)

REM Verificacion final
echo.
echo  Verificacion final del estado:
call :check_database_exists %PRIMARY_SERVER_NAME% %DATABASE_NAME%
%RESOURCE_GROUP_NAME% FINAL_PRIMARY
call :check_database_exists %SECONDARY_SERVER_NAME% %DATABASE_NAME%
%RESOURCE_GROUP_NAME% FINAL_SECONDARY
call :get_database_status %PRIMARY_SERVER_NAME% %DATABASE_NAME%
%RESOURCE_GROUP_NAME% FINAL_PRIMARY_STATUS
call :get_database_status %SECONDARY_SERVER_NAME% %DATABASE_NAME%
%RESOURCE_GROUP_NAME% FINAL_SECONDARY_STATUS

if defined FINAL_PRIMARY (
    echo    Base Principal: !FINAL_PRIMARY!
    echo      Estado: !FINAL_PRIMARY_STATUS!
) else (
    echo    Base Principal:  NO EXISTE
)

if defined FINAL_SECONDARY (
    echo    Base Secundaria: !FINAL_SECONDARY!
    echo       Estado: !FINAL_SECONDARY_STATUS!
) else (
    echo    Base Secundaria:  NO EXISTE
)

if defined FINAL_PRIMARY if defined FINAL_SECONDARY (
    echo.
    echo  Configuracion completa y funcional
    echo  Ecualatino Database está listo para usar
    echo.
    echo  Endpoints de conexion:
    echo    Principal: %PRIMARY_SERVER_NAME%.database.windows.net
    echo   Secundaria: %SECONDARY_SERVER_NAME%.database.windows.net
    echo    Base de datos: %DATABASE_NAME%
) else (
    echo.
    echo  Revisar configuracion manualmente
    echo Ejecuta: terraform plan para ver el estado actual
)
```

```
REM --- Capturar y mostrar hora de finalizacion ---
for /f "tokens=1-2 delims= " %%a in ('echo %time%') do set END_TIME=%%a %%b
echo.
echo ====================================
echo Proceso completado - Ecualatino Database Recovery Tool
echo Hora de finalizacion: !END_TIME!
echo.
echo Presiona ENTER para cerrar...
pause >nul
exit /b 0

:check_database_exists
set server_name=%1
set db_name=%2
set rg_name=%3
set return_var=%4
for /f "tokens=*" %%a in ('az sql db show --name %db_name% --server
%server_name% --resource-group %rg_name% --query "name" -o tsv 2^>nul') do (
    if not "%%a"=="" set %return_var%=%%a
)
goto :eof

:get_database_status
set server_name=%1
set db_name=%2
set rg_name=%3
set return_var=%4
for /f "tokens=*" %%a in ('az sql db show --name %db_name% --server
%server_name% --resource-group %rg_name% --query "status" -o tsv 2^>nul') do (
    if not "%%a"=="" (
        set %return_var%=%%a
    ) else (
        set %return_var%=Not Found
    )
)
goto :eof

:error_exit
echo.
echo  Se produjo un error durante la ejecucion
echo  Revisa los mensajes anteriores para más detalles
pause
exit /b 1
```

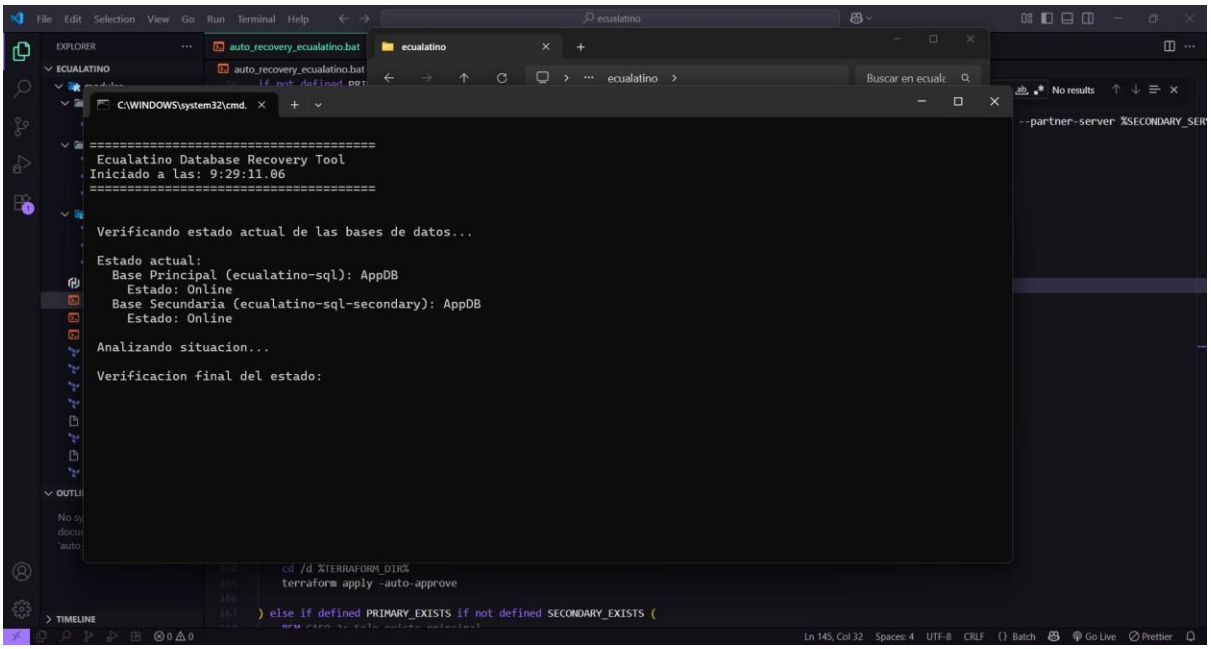
Este script puede ejecutarse desde la consola de Windows con los permisos adecuados y las herramientas CLI necesarias previamente instaladas (Azure CLI, Terraform).

Para una mejor comprensión del funcionamiento del script, en el Anexo 1 se presenta el paso a paso detallado del proceso de recuperación automatizado, incluyendo capturas de pantalla y una explicación de cada una de las acciones ejecutadas.

Resultado de la Simulación

Una vez ejecutado el script, se logró una **recuperación completa del servicio**, manteniendo la continuidad de datos y replicación. La base de datos fue reconstruida exitosamente y volvió a estar disponible en ambas regiones (primaria y secundaria), como se muestra en la Figura 57.

Figura 57:
Estado final de las bases de datos tras la recuperación automática



3.2. Resultados Obtenidos

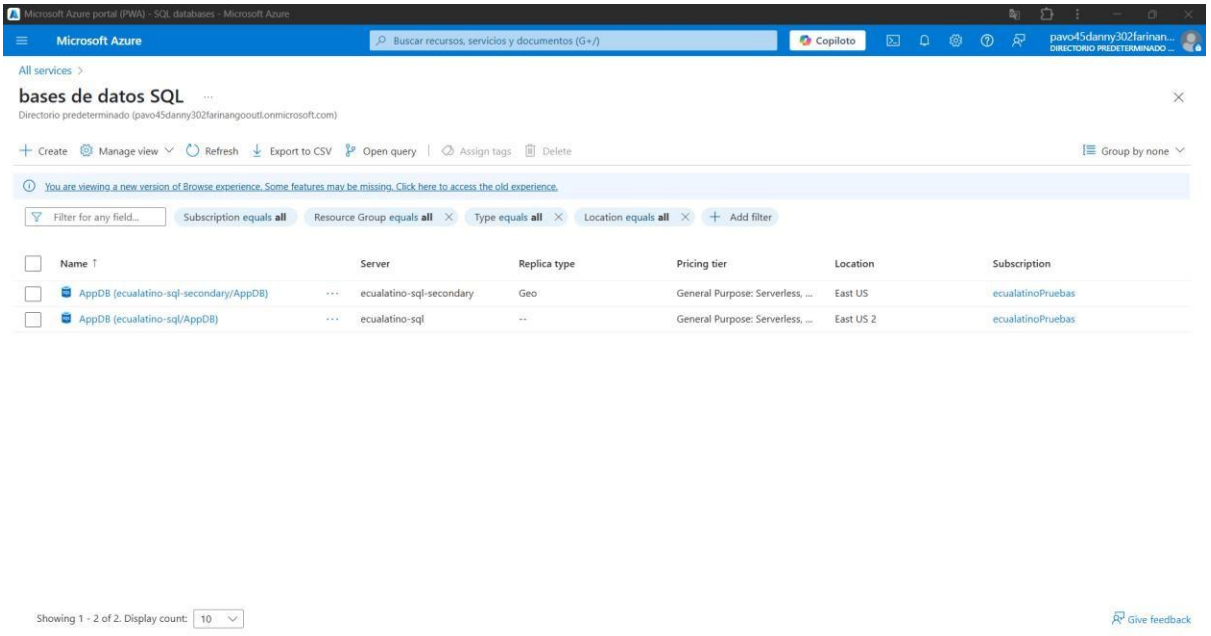
Como resultado de la ejecución del procedimiento de recuperación automatizada, se logró restaurar exitosamente la infraestructura crítica del sistema **Ecualatino**. A continuación, se detallan los principales logros alcanzados:

- **Infraestructura recuperada exitosamente:** La base de datos principal fue recreada a partir de la réplica secundaria y su estado pasó a estar operativo sin errores.
- **Geo-replicación restaurada sin intervención manual adicional:** Una vez recreada la base principal, se volvió a establecer la geo-replica con la región secundaria de forma automática, tal como estaba definido en la configuración original.
- **La aplicación siguió disponible (o fue restablecida rápidamente):** La aplicación web se mantuvo operativa durante el proceso o fue restablecida en un corto tiempo tras la recuperación, sin requerir intervenciones adicionales sobre el servidor web.

- **Validaciones finales satisfactorias:** Se realizaron conexiones exitosas a la base de datos recuperada, así como consultas desde la aplicación para comprobar la integridad de los datos.

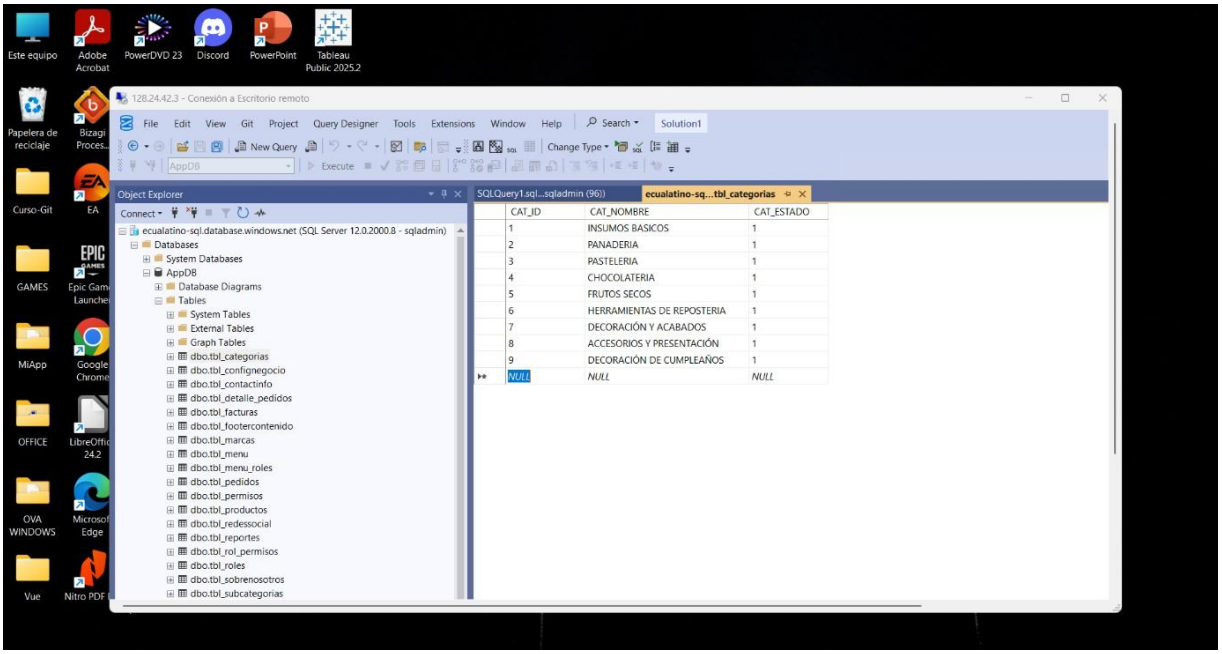
La Figura 58 muestra el estado final de las bases de datos en el portal de Azure, donde se observa que ambas instancias (principal y secundaria) se encuentran **en línea** y correctamente replicadas.

Figura 58:
Estado final de las bases de datos en Azure tras la recuperación automatizada.



Por otro lado, la Figura 59 muestra una conexión establecida a la base de datos AppDB desde SQL Server Management Studio (SSMS), validando el acceso, integridad de datos y disponibilidad del sistema.

Figura 59:
Conexión exitosa a la base de datos AppDB desde SSMS tras el proceso de recuperación.



El sistema demostró ser resiliente ante fallas críticas y capaz de recuperar la infraestructura y los datos sin intervención humana directa. La automatización mediante IaC representa una herramienta poderosa para garantizar la continuidad operativa y preparar escenarios ante desastres.

CONCLUSIONES

La implementación de Infraestructura como Código (IaC) en la empresa Ecualatino permitió validar la viabilidad de automatizar el despliegue y recuperación de entornos tecnológicos en la nube, mejorando significativamente la eficiencia, la trazabilidad y la disponibilidad de los servicios. Durante el proceso de pruebas, el sistema demostró su capacidad de recuperación ante fallos simulados, logrando restaurar el servicio en un tiempo promedio de 20 minutos. Este resultado representa una mejora notable frente a procedimientos tradicionales que dependen de intervención manual.

- **Valor del enfoque IaC:** El uso de Terraform facilitó un aprovisionamiento reproducible y auditable de la infraestructura. Esta práctica asegura coherencia en los entornos y permite recrear configuraciones con exactitud, incluso después de eventos críticos.
- **Continuidad operativa garantizada:** La capacidad de restaurar el sistema con rapidez y sin intervención directa reduce significativamente los tiempos de inactividad, lo cual es crucial para mantener la competitividad de la empresa y la confianza del cliente.

Durante el desarrollo, surgieron inconvenientes relevantes que fueron resueltos con enfoques alternativos. Uno de los principales fue que, al cambiar parámetros sensibles como el usuario o contraseña de la máquina virtual o base de datos, Terraform tendía a destruir y recrear automáticamente dichos recursos. Para evitar esta pérdida de configuraciones o tiempos innecesarios de aprovisionamiento, se optó por asegurar previamente la estabilidad de las credenciales antes de aplicar cambios, minimizando así la recreación no deseada de recursos.

Otro desafío importante fue la gestión de la base de datos protegida con `lifecycle { prevent_destroy = true }`. Esta configuración evitaba su eliminación accidental, pero impedía destruir toda la infraestructura de manera limpia cuando era necesario. La solución fue realizar el cambio manualmente (estableciendo `prevent_destroy = false`) antes de proceder con el `terraform destroy`, ya que Terraform no permitía automatizar ese ajuste de forma sencilla por incompatibilidades en su ciclo de ejecución.

También se presentaron errores relacionados con las restricciones regionales de Azure**,** como la no disponibilidad de ciertos tamaños de máquinas virtuales o tipos de bases de datos en determinadas regiones. Estos fueron resueltos probando configuraciones alternativas y ajustando la región o los recursos según la compatibilidad del proveedor.

Finalmente, para asegurar una recuperación fiable ante fallos, se creó un script adicional para automatizar backups**,** el cual fue probado exitosamente antes de realizar cualquier eliminación

de infraestructura. Gracias a este mecanismo, los datos de la base de datos se conservaron intactos incluso cuando los entornos fueron destruidos y aprovisionados nuevamente.

Esta investigación no solo aporta soluciones prácticas a los desafíos tecnológicos actuales de Ecualatino, sino que también ofrece un marco de referencia aplicable a otras organizaciones que busquen optimizar sus operaciones en la nube. En un contexto empresarial altamente competitivo, el uso de IaC representa una ventaja estratégica al permitir mayor agilidad, seguridad y control en la gestión de infraestructura.

El trabajo realizado demuestra que la automatización mediante IaC no es únicamente una mejora técnica, sino una inversión que impacta positivamente en la continuidad del negocio, la reducción de costos y la capacidad de escalar de forma ordenada. Así, la adopción de esta metodología posiciona a las organizaciones frente a un futuro más resiliente y eficiente.

RECOMENDACIONES

Si bien los resultados fueron positivos, se identificaron áreas clave donde se podría fortalecer aún más la solución:

- **Incorporar monitoreo automatizado de servicios:** Se recomienda implementar herramientas de monitoreo continuo para bases de datos, redes y máquinas virtuales, que permitan detectar proactivamente fallos o caídas de rendimiento.
- **Integrar alertas automáticas:** La implementación de notificaciones vía correo electrónico, Microsoft Teams u otros canales puede mejorar la capacidad de respuesta inmediata ante incidentes o cambios en el estado del entorno.
- **Fortalecer los scripts de automatización:** Es recomendable optimizar el script en PowerShell utilizado para el proceso de recuperación, añadiendo validaciones más robustas, manejo detallado de errores y generación de registros (logs) que faciliten el análisis posterior.
- **Fomentar la documentación y capacitación continua:** Documentar detalladamente el proceso de despliegue y recuperación, así como capacitar al equipo técnico en el uso de IaC y herramientas relacionadas, contribuirá a una adopción más eficiente y sostenible.

REFERENCIAS BIBLIOGRAFICAS

AWS. (2025). Uso de Terraform como herramienta de IaC para Nube de AWS. https://docs.aws.amazon.com/es_es/prescriptive-guidance/latest/choose-iac-tool/terraform.html

Check Point Software. (2025). Seguridad de la infraestructura como código (IaC). https://www.checkpoint.com/es/cyber-hub/cloud-security/what-is-infrastructure-as-code-iac/infrastructure-as-code-iac-security/

Chinamanagonda, S. (2019). Automating infrastructure with infrastructure as code (IaC). International Journal of Science and Research (IJSR), 8(11), 2037–2045. https://www.ijsr.net/archive/v8i11/SR24829170834.pdf

Chinamanagonda, S. (2019). Automating infrastructure with infrastructure as code (IaC). SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4986767

Digital.ai. (2024). Creación y gestión de pipelines de CI/CD con Jenkins. https://digital.ai/es/catalyst-blog/cicd-pipeline-jenkins

Estrategia Digital. (2023). Casos de éxito de empresas que han implementado estrategias innovadoras en su sector. https://estrategiadigital.org/innovacion-tecnologica/casos-de-exito-de-empresas-que-han-implementado-estrategias-innovadoras-en-su-sector/

Galarza, P. (2024). IaC (Infraestructura como Código) aplicado en proyectos. https://paulogalarza.com/iac-infraestructura-como-codigo-aplicado-en-proyectos/

GitLab. (2025). ¿Qué es un pipeline de CI/CD?. https://about.gitlab.com/es/topics/ci-cd/cicd-pipeline/

HashiCorp. (s.f.). Terraform: Automate Infrastructure on Any Cloud. https://developer.hashicorp.com/terraform

HostDime. (2024). Seguridad en la nube: riesgos, desafíos y soluciones. https://blog.hostdime.com.co/seguridad-en-la-nube-riesgos-desafios-y-soluciones/

IAEME Publication. (2022). *Optimizing Continuous Integration and Continuous Deployment Pipelines in DevOps Environments*. International Journal of Computer Engineering and Technology, 13(3), 95-101. https://www.academia.edu/114061716/OPTIMIZING_CONTINUOUS_INTEGRATION_AND_CONTINUOUS_DEPLOYMENT_PIPELINES_IN_DEVOPS_ENVIRONMENTS

Insitech. (2025). 6 desafíos y soluciones de gestión de múltiples nubes. https://go.insitech.com.mx/6-desafios-y-soluciones-de-gestion-de-multiples-nubes/

IBM. (2023). Infraestructura como código. https://www.ibm.com/es-es/topics/infrastructure-as-code

IT Trends. (2020). Factores que están impulsando la adopción de inteligencia artificial. https://www.ittrends.es/inteligencia-artificial/2020/06/factores-que-estan-impulsando-la-adopcion-de-inteligencia-artificial

Kissel, J. (2018). Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management. IEEE. https://www.academia.edu/39802146/Continuous_Integration_and_Continuous_Delivery_Pipeline_Automation_for_Agile_Software_Project_Management

Macías Sánchez, M., Tamayo Maggi, M., & Cerda Paredes, M. (2019). Resistencia al cambio en las organizaciones: propuesta para minimizarlo. Universidad de las Fuerzas Armadas ESPE. https://www.palermo.edu/economicas/cbrs/pdf/pbr19/PBR_19_02.pdf

Microsoft. (2023). ¿Qué es la infraestructura como código?. Microsoft Learn. https://learn.microsoft.com/es-es/devops/deliver/what-is-infrastructure-as-code

Microsoft. (2025). ¿Qué es la infraestructura como código?. Microsoft Learn. https://learn.microsoft.com/es-es/azure/cloud-adoption-framework/organize/cloud-governance

Microsoft. (2025). DevSecOps para infraestructura como código (IaC). https://learn.microsoft.com/es-es/azure/architecture/solution-ideas/articles/devsecops-infrastructure-as-code

Moreno, J. (2022). Infraestructura como código. Universitat Oberta de Catalunya. https://openaccess.uoc.edu/bitstream/10609/145707/8/moreno_juanTFG0622memoria.pdf

Pacheco-Ruíz, C., Rojas-Martínez, C., Niebles-Nuñez, W., & Hernández-Palma, H. (2020). Desarrollo integral de procesos de adaptación al cambio en pequeñas y medianas empresas. SciELO. https://www.scielo.cl/scielo.php?script=sci_arttext&pid=S0718-07642020000500089

Paradigma Digital. (2022). La era de la automatización IaC: la infraestructura como código. https://www.paradigmadigital.com/techbiz/iac-infraestructura-como-codigo/

Pérez Castro, A. M. (2015). Análisis comparativo de software para levantar una infraestructura como servicio en cloud computing e implementación de una nube privada. Universidad Técnica del

Norte.https://repositorio.utn.edu.ec/bitstream/123456789/5343/3/04%20ISC%20358%20ARTICULO.pdf

Saxena, A., Singh, S., Prakash, S., Yang, T., & Rathore, R. (2025). *DevOps Automation Pipeline Deployment with IaC (Infrastructure as Code)*. IEEE Silchar Subsection Conference. https://arxiv.org/abs/2503.16038

Sentrio. (2023). Prácticas de infraestructura como código (IaC). https://sentrio.io/blog/practicas-de-infraestructura-como-codigo-iac/

Skiller Academy. (2025). Superar limitaciones técnicas con IA: guía completa. https://skiller.education/superar-limitaciones-tecnicas-con-ia/

Socioestrategia. (2023). Afrontando la resistencia al cambio organizacional para un crecimiento personal y profesional. https://socioestrategia.com/afrontando-la-resistencia-al-cambio-organizacion

Suppi Boldrito, R. (2022). Infraestructura como código (IaC): Monitorización y seguridad. Universitat Oberta de Catalunya. https://openaccess.uoc.edu/bitstream/10609/151318/1/InfraestructuraComoCodigoIaCMonitorizacionSeguridad.pdf

Tacuri Pajuña, F. M. (2023). Estrategias de arquitectura de solución escalables con aprovisionamiento de infraestructura automática (IaC). Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/bitstream/123456789/26430/1/MSQ702.pdf

Tamburri, D. A. (2019). *Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry*. IEEE International Conference on Software Maintenance and Evolution. https://www.academia.edu/67912701/Adoption_Support_and_Challenges_of_Infrastructure_as_Code_Insights_from_Industry

TNECom Canarias. (2025). Infraestructura como Código (IaC): Automatización segura en la gestión de recursos. https://www.tnecomcanarias.es/blog/infraestructura-como-c%C3%B3digo-iac-automatizaci%C3%B3n-segura-en-la-gesti%C3%B3n-de-recursos

ANEXOS

Anexo 1: Explicación paso a paso del script auto_recovery_ecualatino.bat

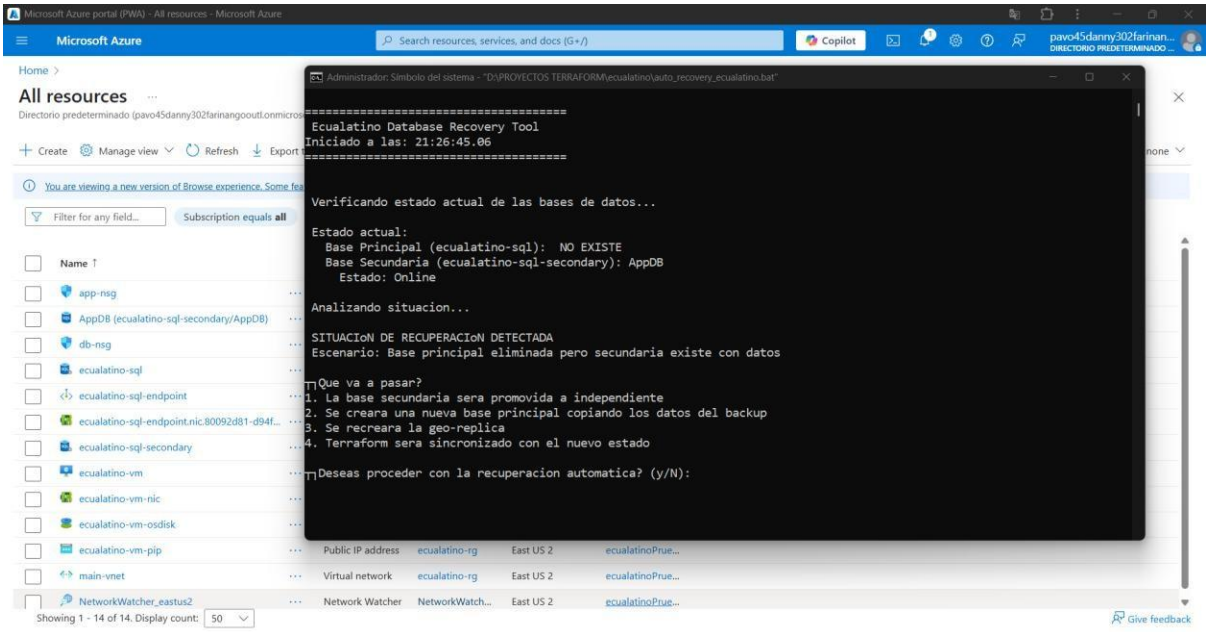
El script auto_recovery_ecualatino.bat fue ejecutado desde la máquina administrativa con permisos adecuados sobre los recursos de Azure. A continuación, se detalla el proceso automatizado que se llevó a cabo:

1. Verificación del estado inicial de las bases de datos

El script consulta el estado de la base de datos primaria y secundaria mediante comandos az sql db show. Esto permite determinar si existe una situación anómala, como la ausencia de la base principal.

La Figura 60 muestra la verificación del estado inicial antes de iniciar el proceso.

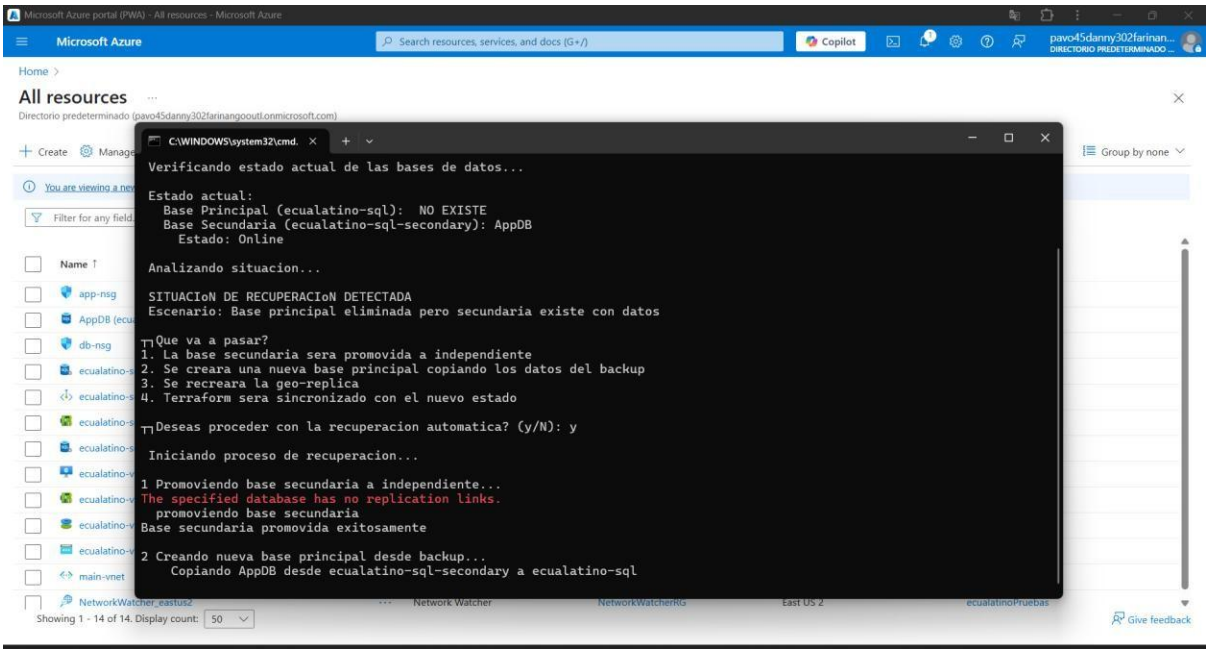
Figura 60:
Estado inicial de las bases de datos antes del proceso de recuperación



2. Análisis de situación y decisión automática

Si se detecta que la base principal no existe, pero la secundaria está disponible, se activa el flujo de recuperación. El script solicita una confirmación al usuario para continuar, asi como se muestra en la figura 61.

Figura 61:
Análisis de situación y decisión automática

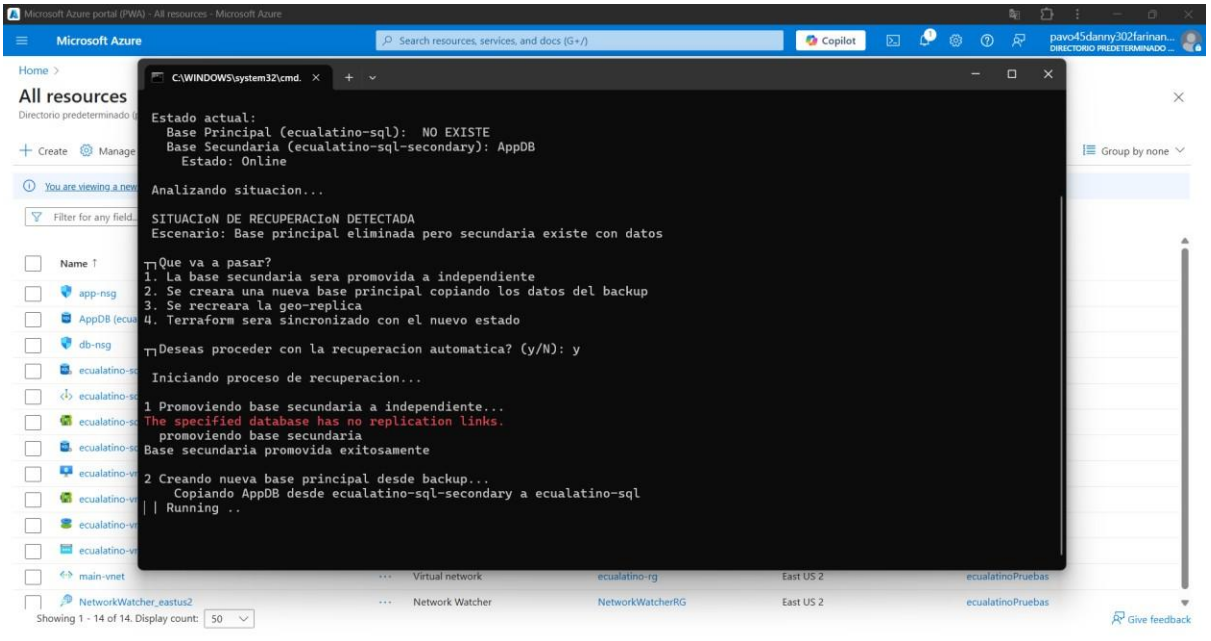


3. Promoción de la base secundaria a independiente

Utilizando el comando az sql db replica set-primary que se presentó el código anterior en la figura 56, se promueve la base secundaria a una base autónoma. Esto garantiza que los datos se mantengan sin pérdida.

La Figura 62 presenta la ejecución del comando de promoción y su confirmación.

Figura 62:
Promoción de la base secundaria a principal

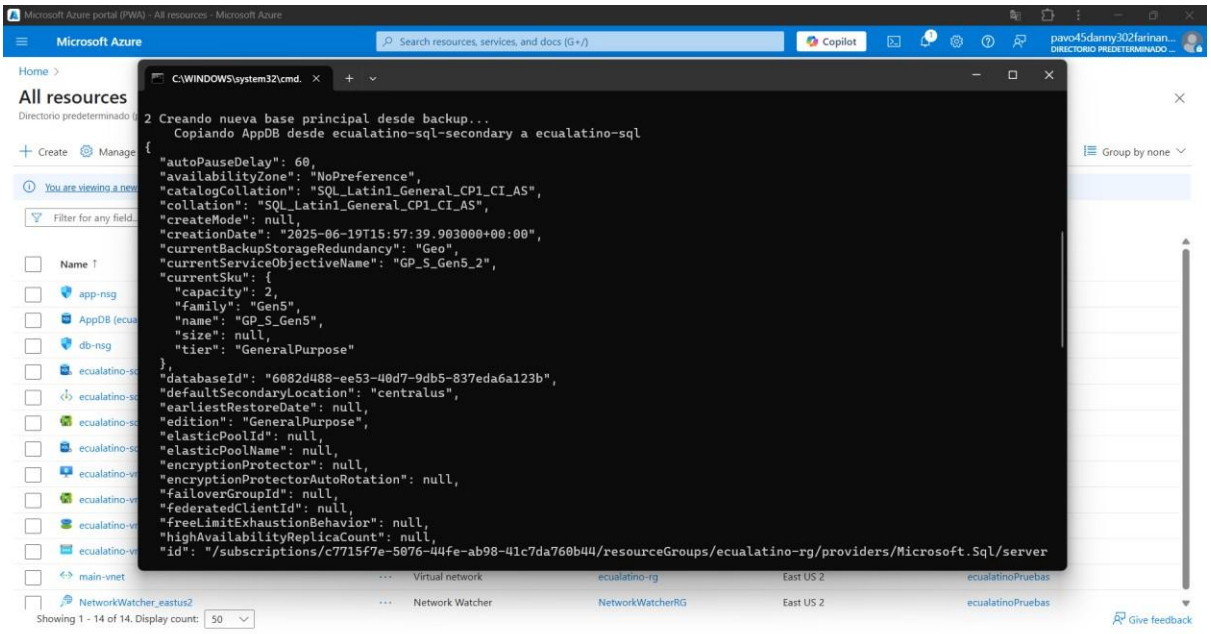


4. Copia de la base promovida a una nueva instancia primaria

Una vez promovida, se realiza una copia de esta base para restaurar la arquitectura original. Este paso se realiza mediante az sql db copy.

En la Figura 63 se observa la ejecución de la copia hacia el servidor primario.

Figura 63:
Copia de la base promovida hacia el servidor primario.

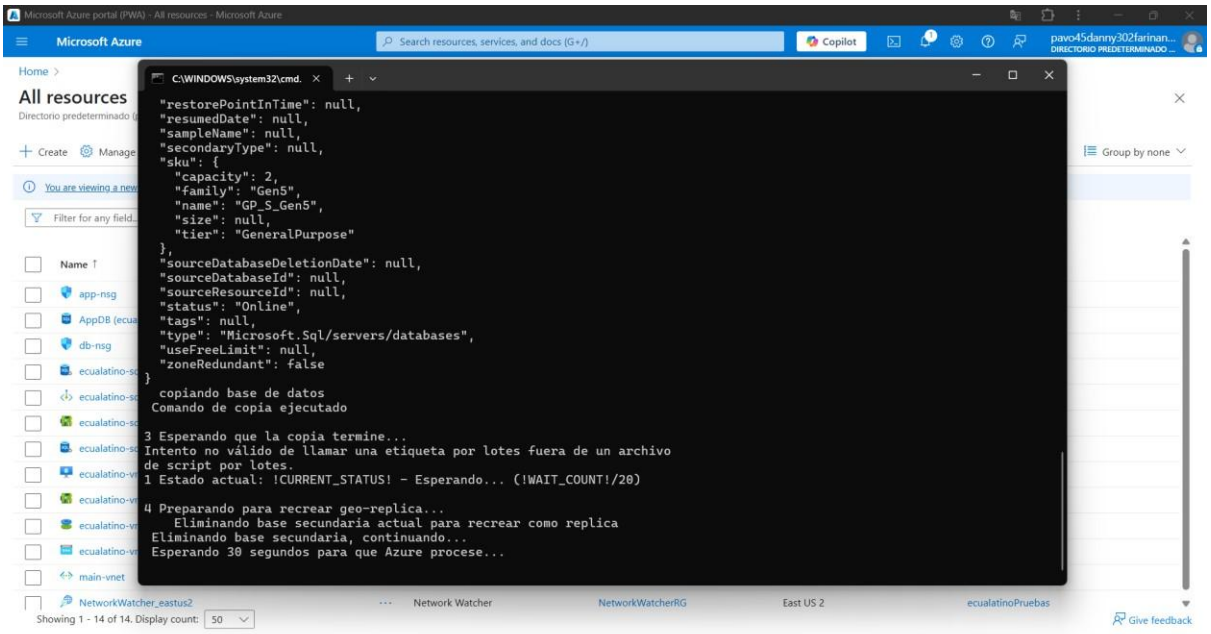


5. Recreación de la geo-replica

Después de validar que la nueva base primaria se encuentra “Online”, se elimina la anterior base secundaria y se recrea la geo-replica con az sql db replica create, restableciendo la alta disponibilidad.

La Figura 64 ilustra este paso de sincronización entre servidores.

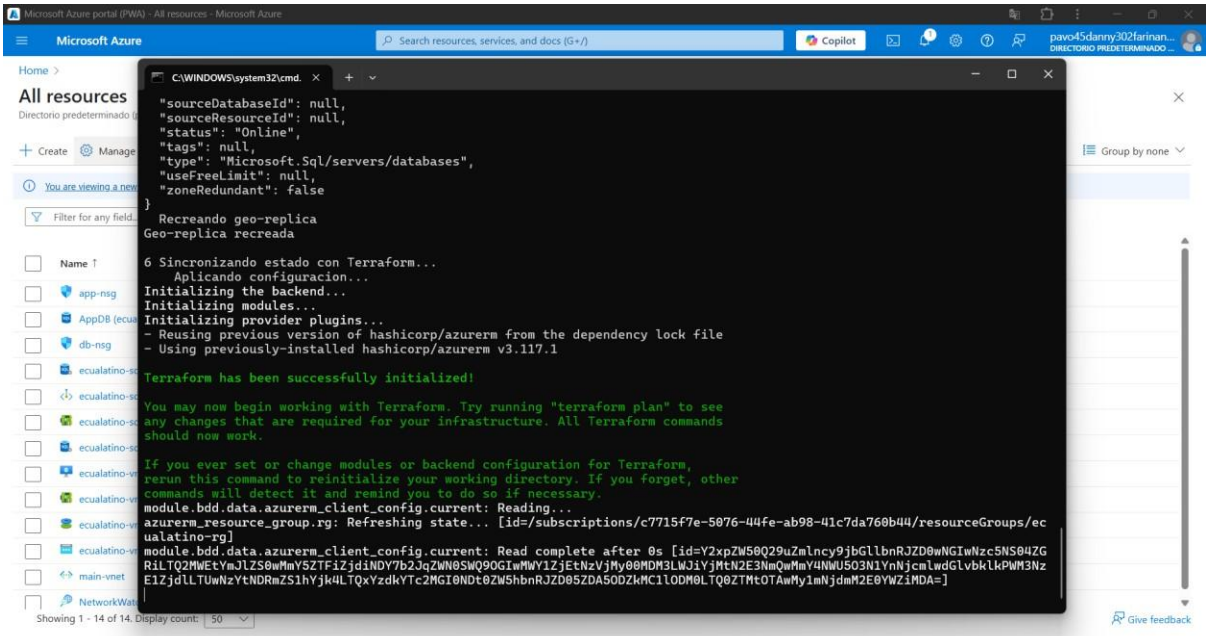
Figura 64:
Recreación de la geo-replica entre servidores



6. Sincronización del estado con Terraform

Finalmente, se ejecuta terraform apply para actualizar el estado de la infraestructura y asegurar que la configuración refleje el nuevo escenario post-recuperación asi como se muestra en la figura 65.

Figura 65:
Estado Terraform



Anexo 2. Carta de recepción de la empresa

Ecualatino S.A.

Ibarra, Ecuador

14/08/21025

Por medio de la presente, la empresa **Ecualatino S.A.**, dedicada al desarrollo de software y soluciones tecnológicas, ha revisado y acepta el **Trabajo de Titulación "Automatización de Procesos de Despliegue y Gestión de Entornos en la Nube mediante Infraestructura como Código (IaC) en la Empresa Ecualatino"**, elaborado por el estudiante **Danny Farinango**, el cual cumple con todos los aspectos detallados y objetivos para la obtención del título de **Ingeniero en Tecnologías de la Información** en la **Pontificia Universidad Católica del Ecuador, Sede Ibarra**.

Dicho proyecto tiene como objetivo optimizar la gestión de la infraestructura tecnológica de nuestra empresa mediante la implementación de **Infrastructure as Code (IaC)**, lo que permitirá mejorar la eficiencia operativa, reducir errores humanos y garantizar la continuidad del negocio a través de la automatización de procesos de TI. En calidad de representante de Ecualatino S.A., confirmo la Aceptación del Proyecto y de acuerdo con el desarrollo de **Infrastructure as Code (IaC)**,

Sin otro particular, extendemos nuestro respaldo al proyecto y quedamos atentos a cualquier consulta adicional.

Firma: Galo Hernán Puetate Huera — Firmado digitalmente por Galo Hernán Puetate Huera. Fecha: 2025.08.14 10:31:50 -05'00'

040137578-7

Dirección: Av. Teodoro Gómez Esquina 7-42 y Calixto Miranda
Teléfono: 062605673 - 0999458210
Email: gerencia@ecualatino.com