



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

**Automatización de máquinas
virtuales en RHEL con
Terraform-Azure en entornos
multicloud**

TESIS

Que para obtener el título de
Ingeniero en Telecomunicaciones

P R E S E N T A

Daniel Jiménez Gámez

DIRECTOR DE TESIS

Dr. Luis Francisco García Jiménez



Ciudad Universitaria, Cd. Mx., 2026



**PROTESTA UNIVERSITARIA DE INTEGRIDAD Y
HONESTIDAD ACADÉMICA Y PROFESIONAL
(Titulación con trabajo escrito)**



De conformidad con lo dispuesto en los artículos 87, fracción V, del Estatuto General, 68, primer párrafo, del Reglamento General de Estudios Universitarios y 26, fracción I, y 35 del Reglamento General de Exámenes, me comprometo en todo tiempo a honrar a la institución y a cumplir con los principios establecidos en el Código de Ética de la Universidad Nacional Autónoma de México, especialmente con los de integridad y honestidad académica.

De acuerdo con lo anterior, manifiesto que el trabajo escrito titulado AUTOMATIZACION DE MAQUINAS VIRTUALES EN RHEL CON TERRAFORM-AZURE EN ENTORNOS MULTICLOUD que presenté para obtener el título de INGENIERO EN TELECOMUNICACIONES es original, de mi autoría y lo realicé con el rigor metodológico exigido por mi Entidad Académica, citando las fuentes de ideas, textos, imágenes, gráficos u otro tipo de obras empleadas para su desarrollo.

En consecuencia, acepto que la falta de cumplimiento de las disposiciones reglamentarias y normativas de la Universidad, en particular las ya referidas en el Código de Ética, llevará a la nulidad de los actos de carácter académico administrativo del proceso de titulación.

DANIEL JIMENEZ GAMEZ
Número de cuenta: 319053463

Agradecimientos

A mi asesor

Expreso mi más sincero agradecimiento por su guía, apoyo y acompañamiento durante el desarrollo de este trabajo, así como por sus valiosas observaciones que permitieron fortalecer esta investigación. Asimismo, agradezco su forma de enseñanza, la cual despertó en mí un gran interés por esta área de especialización y me permitió descubrir una vocación profesional dentro de este campo.

A la Universidad Nacional Autónoma de México

Agradezco a la Universidad Nacional Autónoma de México y a la Facultad de Ingeniería por brindarme la formación académica y el entorno necesarios para el desarrollo de este trabajo. Asimismo, agradezco el sentido de pertenencia que me otorgaron, así como a las personas que conocí en este camino y que, sin esperarlo, se convirtieron en una familia.

A mi familia

Agradezco profundamente a mis padres por su apoyo incondicional, su confianza y su constante motivación a lo largo de este proceso. Asimismo, agradezco a mi hermana por su compañía y respaldo durante esta etapa.

A mis amigos

De manera especial, agradezco a Lalo, Leo, Charly, Santiago, Mike, Alfredo, Luis Da, Mary Tere, Pato e Irel, quienes me brindaron su apoyo y compartieron aprendizajes significativos a lo largo de este camino.

Finalmente, agradezco a todas aquellas personas que, de manera directa o indirecta, contribuyeron al desarrollo de esta tesis.

Acrónimos

1. **API:** *Application Programming Interface*, Interfaz de Programación de Aplicaciones.
2. **CI:** *Continuous Integration*, Integración Continua.
3. **CIDR:** *Classless Inter-Domain Routing*, Enrutamiento Entre Dominios sin Clases.
4. **CD:** *Continuous Delivery/Deployment*, Entrega/Implementación Continua.
5. **CLI:** *Command Line Interface*, Interfaz de Línea de Comandos.
6. **IaC:** *Infrastructure as Code*, Infraestructura como Código.
7. **IoT:** *Internet of Things*, Internet de las Cosas.
8. **IP:** *Internet Protocol*, Protocolo de Internet.
9. **IaaS:** *Infrastructure as a Service*, Infraestructura como Servicio.
10. **CA:** *Certificate Authority*, Unidad Certificadora.
11. **CAPEX:** *Capital Expenditure*, Gasto Capital.
12. **CPU:** *Central Processing Unit*, Unidad Central de Procesamiento.
13. **DCHP:** *Dynamic Host Configuration Protocol*, Protocolo de Configuración Dinámica de Host.
14. **EDGE:** *Enhanced Data rates for GSM of Evolution*, Procesamiento Perimetral.
15. **HCL:** *HashiCorp Configuration Language*, Lenguaje de configuración de HashiCorp.
16. **HTTPS:** *Hypertext Transfer Protocol Secure*, Protocolo de Transferencia de Hipertexto Seguro.
17. **KVM:** *Kernel-based Virtual Machine*, Máquina Virtual Basada en Kernel.
18. **libvirt:** *LIBrary VIRTualization*, Biblioteca de Virtualización.
19. **OPEX:** *Operating Expenses*, Gastos Operativos.
20. **PaaS:** *Platform as a Service*, Plataforma como Servicio.
21. **QEMU:** *Quick Emulator*, Emulador Rápido.

- 22. **RAM:** *Random Access Memory*, Memoria de Acceso Aleatorio.
- 23. **RHEL:** *Red Hat Enterprise Linux*.
- 24. **SaaS:** *Software as a Service*, Software como Servicio.
- 25. **SELinux:** *Security-Enhanced Linux*, Módulo de Seguridad del Kernel de Linux.
- 26. **SKU:** *Stock Keeping Unit*, Unidad de Mantenimiento de Existenciasahora .
- 27. **SO:** *Operating System*, Sistema Operativo.
- 28. **SSH:** *Secure Shell*.
- 29. **TI:** *information technologies*, Tecnologías de la Información.
- 30. **V-CPU:** *Virtual Central Processing Unit* , Unidad Central de Procesamiento Virtual.
- 31. **VM:** *Virtual Machine*, Máquina Virtual.
- 32. **VMM:** *Virtual Machine Monitor*, Monitor de la Máquina Virtual.

Índice general

Resumen	1
1. Introducción	2
1.1. Definición del problema	3
1.2. Alcance de la tesis	3
1.3. Objetivo	4
1.4. Hipótesis	4
1.5. Justificación	4
1.6. Metodología de trabajo	5
1.7. Estructura de la tesis	5
2. Marco conceptual	6
2.1. Virtualización	6
2.1.1. Hipervisor	6
2.1.2. Tipos de Hipervisor	7
2.1.3. KVM, QEMU y libvirt en entorno Linux	7
2.2. Máquinas Virtuales	8
2.3. Cómputo en la nube	9
2.3.1. Interacción entre virtualización y cómputo en la nube	10
2.4. Red Hat Enterprise Linux (<i>RHEL</i>)	11
2.5. Infraestructura como Código	11
2.5.1. Terraform	12
2.5.2. Microsoft Azure	14
2.6. Automatización de despliegues	14
3. Entorno base en RHEL y OpenStack	16
3.1. Referencia del entorno base	16
3.2. Proceso manual de creación de <i>VM</i>	18
3.3. Limitaciones identificadas en el entorno	19
4. Automatización de la creación de máquinas virtuales	21
4.1. Diseño del proceso automatizado	21
4.2. Desarrollo del <i>script</i> en <i>Terraform</i>	22
4.2.1. Definición de recursos	22
4.2.2. Variables y archivos de configuración	23
4.3. Despliegue automatizado en <i>RHEL/OpenStack</i>	25
4.4. Visualización y validación en <i>Horizon</i>	27

5. Despliegue automatizado en Microsoft Azure	28
5.1. Configuración del proveedor <i>Azure</i>	28
5.2. Adaptación del código <i>Terraform</i>	29
5.3. Creación automatizada de máquinas virtuales	31
5.4. Configuración de red y acceso	34
5.5. Validación del despliegue	35
6. Análisis y resultados	37
6.1. Pruebas de rendimiento y escalabilidad	37
6.1.1. Prueba de paralelismo	37
6.1.1.1. Prueba de paralelismo en <i>OpenStack</i>	42
6.1.1.2. Prueba de paralelismo en <i>Azure Portal</i>	46
6.1.2. Prueba de destrucción de infraestructura	49
6.1.2.1. Prueba de destrucción en <i>OpenStack</i>	49
6.1.2.2. Prueba de destrucción en <i>Azure Portal</i>	52
6.1.3. Prueba de actualización de infraestructura	53
6.1.3.1. Prueba de actualización en <i>OpenStack</i>	53
6.1.3.2. Prueba de actualización en <i>Azure Portal</i>	57
6.1.4. Validación del archivo de estado	61
6.1.4.1. Prueba de validación del archivo de estado en <i>OpenStack</i>	61
6.1.4.2. Prueba de validación del archivo de estado en <i>Azure Portal</i>	64
6.1.5. Prueba de <i>rollback</i> mediante control de versiones	67
6.1.5.1. Prueba de <i>rollback</i> en <i>OpenStack</i>	67
6.1.5.2. Prueba de <i>rollback</i> en <i>Azure Portal</i>	72
6.1.6. Comparación entre <i>OpenStack</i> y <i>Microsoft Azure Portal</i>	74
6.2. Análisis de ventajas de la automatización	76
6.3. Limitaciones identificadas en la implementación	77
7. Conclusiones y trabajo futuro	78
7.1. Conclusiones	78
7.2. Trabajo futuro	79
Apéndice	80
A. Código <i>Terraform</i> para <i>OpenStack</i>	80
A.1. Archivo principal <i>main.tf</i>	80
B. Código <i>Terraform</i> para <i>Microsoft Azure</i>	82
B.1. Archivo principal <i>main.tf</i>	82
C. Configuración adicional del nodo <i>Compute</i>	86
C.1. Archivo <i>/etc/libvirt/qemu.conf</i>	86
C.2. Archivo <i>/etc/neutron/neutron.conf</i>	86
D. Configuración adicional del nodo <i>Controller</i>	87
D.1. Archivo <i>/etc/hosts</i>	87
Bibliografía	88

Índice de figuras

2.1. Arquitectura de <i>KVM</i> y <i>QEMU</i>	8
3.1. Módulos básicos de <i>OpenStack</i>	16
3.2. Interfaz de <i>Horizon</i> : pantalla de inicio de sesión.	19
4.1. Vista de los proyectos disponibles en <i>OpenStack Horizon</i>	22
4.2. Registro del <i>flavor m1.tiny</i> dentro del panel de administración de <i>OpenStack Horizon</i>	23
4.3. Ejecución de los comandos <code>terraform init</code> , <code>terraform plan</code> y <code>terraform apply</code> durante el despliegue automatizado de la instancia en <i>OpenStack</i>	26
4.4. Visualización en <i>Horizon</i> de la instancia desplegada mediante <i>Terraform</i>	27
5.1. Sección de suscripciones en el portal de <i>Microsoft Azure</i>	29
5.2. Consulta de compatibilidad de tamaños de máquina virtual en la región <i>southcentralus</i>	32
5.3. Consulta de <i>SKU</i> de imágenes Ubuntu disponibles en la región <i>southcentralus</i> , donde se identifica la opción <i>22_04-lts-arm64</i>	32
5.4. Ejecución del comando <code>terraform init -upgrade</code> y primera parte del comando <code>terraform plan</code>	33
5.5. Segunda parte de la ejecución del comando <code>terraform plan</code>	33
5.6. Primera parte de la ejecución del comando <code>terraform apply</code>	34
5.7. Segunda parte de la ejecución del comando <code>terraform apply</code>	34
5.8. Máquina virtual desplegada en <i>Microsoft Azure</i>	35
6.1. Configuración inicial de memoria en los nodos <i>controller</i> y <i>compute</i>	38
6.2. Incremento de memoria <i>RAM</i> en los nodos <i>controller</i> y <i>compute</i>	39
6.3. Estadísticas del hipervisor en <i>OpenStack</i>	40
6.4. Creación del <i>flavor m1.nanotiny</i> en <i>OpenStack</i>	40
6.5. Configuración inicial de conexiones en <i>MariaDB</i>	41
6.6. Configuración de <i>MariaDB</i> después de incrementar el parámetro <code>max_connections</code>	42
6.7. Tiempo de despliegue en función del nivel de paralelismo en <i>OpenStack</i>	43
6.8. Estado de las máquinas virtuales después de las pruebas de paralelismo en <i>OpenStack</i>	44
6.9. Error <code>NoValidHost</code> durante las pruebas de paralelismo en <i>OpenStack</i>	45
6.10. Cuotas y estadísticas del hipervisor durante las pruebas de paralelismo en <i>OpenStack</i>	45
6.11. Tiempo de despliegue en función del nivel de paralelismo en <i>Azure</i>	47
6.12. Errores de cuota regional durante el despliegue de máquinas virtuales en <i>Azure</i>	48
6.13. Errores adicionales de aprovisionamiento ocasionados por límites regionales de <i>vCPU</i> en <i>Azure</i>	48
6.14. Finalización parcial del despliegue debido a restricciones de cuota en <i>Azure</i>	49
6.15. Máquinas virtuales creadas exitosamente en <i>Azure</i>	49

6.16. Tiempo de destrucción de infraestructura en función del nivel de paralelismo en <i>OpenStack</i>	51
6.17. Tiempo de destrucción en función del nivel de paralelismo en <i>Azure</i>	53
6.18. Creación del <i>flavor</i> <code>m1.medium</code> en <i>OpenStack</i>	54
6.19. Creación del <i>flavor</i> <code>m1.big</code> en <i>OpenStack</i>	54
6.20. Generación de evidencias previas a la actualización de infraestructura.	55
6.21. Verificación de <i>flavors</i> actuales.	56
6.22. Generación de evidencias posteriores a la actualización de infraestructura.	57
6.23. Creación inicial de máquinas virtuales en <i>Azure</i>	57
6.24. Verificación de tamaños de máquina virtual disponibles en <i>Azure</i>	59
6.25. Restricciones regionales durante la actualización de infraestructura en <i>Azure</i>	59
6.26. Detección de <i>drift</i> mediante <code>terraform plan</code> en <i>Azure</i>	61
6.27. Comparación de evidencias antes y después de modificar la máquina virtual <code>vm-terraform-1</code>	62
6.28. Comparación de evidencias antes y después de realizar <i>resize</i> sobre la máquina virtual <code>vm-terraform-2</code>	62
6.29. Comparación de evidencias antes y después de realizar <i>resize</i> sobre la máquina virtual <code>vm-terraform-3</code>	63
6.30. Detección de diferencias entre la infraestructura real y el archivo <code>terraform.tfstate</code> mediante <code>terraform plan</code>	64
6.31. Proceso de reconciliación del archivo <code>terraform.tfstate</code> en <i>Azure</i>	66
6.32. Proceso de reconciliación del archivo <code>terraform.tfstate</code> en <i>Azure</i>	66
6.33. Ejecución de <code>terraform apply -auto-approve</code> durante el proceso de <i>rollback</i>	68
6.34. Validación final del estado de la infraestructura mediante <code>terraform plan</code>	68
6.35. Eventos registrados durante las operaciones de actualización y <i>rollback</i>	69
6.36. Estado final de las máquinas virtuales después del proceso de <i>rollback</i>	70
6.37. Historial de migraciones y operaciones de <i>resize</i> sobre <code>vm-terraform-2</code>	70
6.38. Eventos registrados para la máquina virtual <code>vm-terraform-2</code>	71
6.39. Detalle del error generado durante el proceso de <i>resize</i>	71
6.40. Detalle de una operación exitosa de <i>resize</i> sobre <code>vm-terraform-2</code>	72
6.41. Proceso de restauración de infraestructura mediante <i>Terraform</i> en <i>Azure</i>	73
6.42. Comparación antes y después del proceso de <i>rollback</i> en <i>Azure</i>	74

Índice de tablas

2.1. Comparación general de <i>CAPEX</i> y <i>OPEX</i> en nube privada, pública e híbrida. . . .	10
6.1. Resumen estadístico de las pruebas de paralelismo en <i>OpenStack</i>	43
6.2. Resumen estadístico de las pruebas de paralelismo en <i>Azure</i>	46
6.3. Resumen estadístico de las pruebas de destrucción de infraestructura en <i>OpenStack</i>	50
6.4. Resumen estadístico de las pruebas de destrucción de infraestructura en <i>Azure</i>	52
6.5. Comparación de pruebas de creación entre <i>OpenStack</i> y <i>Microsoft Azure</i> en segundos. . . .	75
6.6. Comparación de pruebas de destrucción entre <i>OpenStack</i> y <i>Microsoft Azure</i> en segundos.	76

Resumen

Cloud Computing es un modelo de cómputo que ofrece recursos tecnológicos bajo demanda mediante paradigmas como *Infrastructure as a Service (IaaS)*, *Platform as a Service (PaaS)* y *Software as a Service (SaaS)*. En particular, el modelo *IaaS* provee máquinas virtuales, redes, almacenamiento y capacidad de procesamiento, permitiendo a las organizaciones desplegar infraestructura sin adquirir *hardware* propio.

En entornos empresariales y educativos de gran escala, la creación manual de máquinas virtuales (VM) puede generar inconsistencias debido a errores humanos y dificultades para replicar configuraciones de manera uniforme. A medida que la infraestructura crece, estos procesos se vuelven poco eficientes y limitan la agilidad operativa. Por lo que en esta tesis se desarrolla un mecanismo automatizado y estandarizado para la creación de VM en un entorno de nube privada basada en la plataforma *OpenStack*, y a su vez en una nube pública basada en *Microsoft Azure*, donde ambas gestionan recursos virtualizados (*IaaS*). Para ello se emplea *Terraform* [1], una herramienta que permite definir Infraestructura como Código (*IaC*) mediante archivos declarativos y reproducibles. Aunque existen diversas herramientas de automatización, se seleccionó *Terraform* por su enfoque declarativo y su compatibilidad con múltiples proveedores.

El mecanismo desarrollado automatiza, mediante *scripts* de *Terraform*, el despliegue de máquinas virtuales, garantizando la consistencia y reproducibilidad en la infraestructura. La investigación incluye una fase experimental donde se evalúa el comportamiento del sistema ante pruebas de estrés mediante el uso de concurrencia, actualizaciones, destrucción y *rollback*. Los resultados obtenidos demuestran que el uso de Infraestructura como Código no solo reduce la variabilidad entre despliegues y optimiza el mantenimiento, sino que también ofrece un modelo cuantificable para la gestión de recursos en entornos de crecimiento dinámico. Finalmente, se evidencia que este enfoque permite una portabilidad efectiva entre nubes privadas y públicas con ajustes mínimos, mitigando la dependencia de proveedores y fortaleciendo la resiliencia operativa.

Capítulo 1

Introducción

El cómputo en la nube ha modificado de manera significativa la forma en que las organizaciones acceden y administran sus recursos tecnológicos. Actualmente, es posible disponer de infraestructura, plataformas y aplicaciones bajo demanda sin la necesidad de contar con *hardware* propio, lo que permite mayor flexibilidad, escalabilidad y optimización de costos. El uso extendido de este modelo ha dado lugar a la clasificación de distintos tipos de nube según su forma de implementación:

- El primer modelo se conoce como nube pública, en la cual los recursos son proporcionados por un proveedor externo y se accede a ellos a través de Internet. En este modelo, el *hardware*, el *software* base y la infraestructura que soporta las máquinas virtuales son propiedad del proveedor, quien también se encarga de su mantenimiento, actualización y disponibilidad. Entre sus principales ventajas se encuentran el modelo de pago por uso, la alta escalabilidad bajo demanda y la reducción de inversión inicial en infraestructura física. Este modelo resulta adecuado cuando se requiere rapidez de implementación y elasticidad en los recursos.
- El segundo modelo se conoce como nube privada. Este modelo puede estar ubicado en un centro de datos propio o en instalaciones dedicadas; sin embargo, su infraestructura no es compartida con terceros. En este esquema, la organización conserva el control total sobre el *hardware*, el *software* y las políticas de seguridad. Esto permite un mayor nivel de personalización, cumplimiento normativo y protección de datos sensibles. Aunque implica mayor inversión y responsabilidad operativa. Ofrece control directo sobre la infraestructura y los recursos.
- El tercer modelo se conoce como nube híbrida. Este modelo combina infraestructura local o privada con servicios de nube pública, permitiendo la interoperabilidad entre ambos entornos. Este modelo facilita mover aplicaciones y datos según las necesidades de seguridad, latencia o costos. Por ejemplo, la información crítica puede mantenerse en infraestructura privada, mientras que cargas de trabajo temporales o picos de demanda pueden ejecutarse en la nube pública. De esta forma se optimizan costos, se mejora la continuidad operativa y se mantiene cumplimiento normativo. Asimismo, puede integrarse con esquemas de procesamiento perimetral (*edge computing*), donde parte del procesamiento se realiza cerca de la fuente de datos para reducir latencia.

Además de los tipos de nube, es fundamental distinguir los modelos de servicio, los cuales determinan el nivel de control que tiene el usuario sobre la infraestructura:

- **Infrastructure as a Service (IaaS)**: el proveedor ofrece infraestructura virtualizada, como máquinas, redes y almacenamiento. El usuario administra el sistema operativo, configuraciones y aplicaciones. Este modelo proporciona mayor control técnico y es el más relacionado con el despliegue de máquinas virtuales (VM).
- **Platform as a Service (PaaS)**: el proveedor administra la infraestructura y la plataforma base (sistema operativo, *middleware* y entorno de ejecución). El usuario se enfoca únicamente en el desarrollo y despliegue de aplicaciones, sin preocuparse por la administración del sistema subyacente.
- **Software as a Service (SaaS)**: el proveedor entrega una aplicación completa accesible a través de Internet. El usuario únicamente utiliza el servicio, sin intervenir en la infraestructura ni en la plataforma que lo soporta.

La principal diferencia entre estos modelos radica en el nivel de responsabilidad y control que asume el usuario frente al proveedor. A mayor control (*IaaS*), mayor responsabilidad técnica; a menor control (*SaaS*), mayor delegación en el proveedor.

1.1. Definición del problema

La creciente demanda en infraestructuras basadas en la nube ha incrementado la necesidad de crear y administrar máquinas virtuales de manera constante, especialmente en empresas e instituciones educativas de gran escala donde la infraestructura crece rápidamente y se requiere replicar entornos de forma continua. En muchos entornos, la creación de instancias suele realizarse mediante configuraciones manuales a través de interfaces gráficas o mediante comandos individuales. Aunque este procedimiento permite desplegar recursos funcionales, depende en gran medida de la experiencia del administrador y carece de un mecanismo formal de estandarización. Por lo que la ausencia de un código estructurado que defina de manera declarativa los recursos necesarios, provoca inconsistencias entre despliegues similares, y dificulta la replicabilidad del entorno, aumentando la probabilidad de errores humanos. Además, cuando se requiere escalar infraestructura o integrar entornos híbridos, la falta de un modelo reproducible complica la portabilidad y el mantenimiento del sistema. Aunado a lo anterior, la carencia de un esquema formal impide conocer el comportamiento de la infraestructura bajo condiciones de alta demanda o fallos operativos, lo cual incrementa el riesgo de interrupciones no planificadas. Por estas razones, la existencia de un esquema de automatización basado en Infraestructura como Código (*IaC*) permite definir, versionar y reproducir de forma uniforme la creación de máquinas virtuales, ya que sin una estandarización clara, el crecimiento de la infraestructura puede volverse desordenada y difícil de controlar.

1.2. Alcance de la tesis

El presente trabajo de tesis se enfoca en la implementación y evaluación de un esquema de automatización para el ciclo de vida de máquinas virtuales mediante *Terraform*. El alcance se delimita a la estandarización de recursos básicos de infraestructura (cómputo, red y almacenamiento) en una plataforma *IaaS* privada basada en OpenStack. La investigación contempla una fase experimental que cuantifica el comportamiento del sistema mediante pruebas de estrés (paralelismo), actualizaciones, procesos de *rollback* y destrucción de recursos.

Para validar la versatilidad del esquema propuesto, se extiende la implementación hacia *Microsoft Azure*, permitiendo un análisis comparativo sobre el grado de reutilización del código. Es importante señalar que, debido a la naturaleza experimental del trabajo, quedan fuera del alcance configuraciones avanzadas de ciberseguridad, así como arquitecturas de alta disponibilidad, centrando el esfuerzo analítico en la consistencia, reproducibilidad y portabilidad del código de infraestructura.

1.3. Objetivo

Evaluar un esquema de automatización basado en Infraestructura como Código con *Terraform*, orientado a la gestión del ciclo de vida (aprovisionamiento, actualización, *rollback* y destrucción) de máquinas virtuales en un entorno *OpenStack*. El estudio incluye una fase de experimentación para medir el rendimiento y la consistencia bajo escenarios de alta concurrencia (paralelismo). Finalmente, se realizará un análisis de portabilidad mediante la parametrización del esquema hacia *Microsoft Azure*, con el fin de cuantificar el grado de reutilización de código y la eficacia de una estrategia de nube híbrida.

1.4. Hipótesis

La implementación de un esquema de automatización basado en Infraestructura como Código (*IaC*) con *Terraform* no solo estandariza y garantiza la reproducibilidad de los despliegues de máquinas virtuales en entornos de nube privada (*OpenStack*) y pública (*Microsoft Azure*), sino que también optimiza el rendimiento operativo y la resiliencia del sistema. Asimismo, se plantea que este enfoque permite cuantificar el comportamiento de la infraestructura bajo condiciones de alta concurrencia y fallos, demostrando que es posible mantener una consistencia operativa superior y una portabilidad eficiente ante cambios de proveedor mediante la parametrización del código.

1.5. Justificación

En entornos de gran escala, la creación manual de máquinas virtuales genera variaciones críticas en la infraestructura, como diferencias en imágenes base, asignación de recursos o políticas de red, que derivan en configuraciones inconsistentes y errores operativos. Ante este escenario, la implementación de Infraestructura como Código no solo permite documentar y versionar el estado del sistema, sino que establece un estándar de reproducibilidad necesario para minimizar la dependencia de configuraciones manuales. Sin embargo, la automatización por sí sola no garantiza la robustez del sistema ante condiciones de alta demanda o fallos. Por ello, esta investigación justifica su relevancia al someter el esquema propuesto a pruebas rigurosas de concurrencia, actualización, destrucción y *rollback*. Estas pruebas son fundamentales para cuantificar los límites operativos, validar la resiliencia del entorno frente a errores y medir la eficacia real de la portabilidad del código entre plataformas heterogéneas como *OpenStack* y *Microsoft Azure*. Este enfoque permite a las instituciones transitar hacia un modelo de nube híbrida bajo un marco de trabajo validado, donde la infraestructura no solo es declarativa, sino también predecible, resiliente y eficiente ante los retos del crecimiento dinámico.

1.6. Metodología de trabajo

La metodología empleada en esta tesis se estructuró en las siguientes etapas:

- Instalación de Red Hat Enterprise Linux (*RHEL 9.5*) en una máquina virtual, con el fin de preparar el entorno donde se crearán y configurarán los nodos de *OpenStack* (*Controller* y *Compute*).
- Instalación de los componentes necesarios de *Terraform* para desarrollar el código estandarizado de aprovisionamiento de *VM* y verificar su registro y disponibilidad en *Horizon*.
- Contratación de una suscripción de *Microsoft Azure* para adaptar el código con *Terraform*, desplegar *VM* en la nube pública de *Microsoft Azure* y validar su creación en *Azure Portal*.
- Crear múltiples escenarios para someter a ambas nubes a estrés y cuantificar los límites operativos mediante pruebas como creación, destrucción, *rollback* y escenarios de alta concurrencia (paralelismo) para medir la resiliencia del entorno frente a errores y la portabilidad del código entre plataformas heterogéneas.

1.7. Estructura de la tesis

La estructura de la tesis está organizada de la siguiente manera:

- **Capítulo 2:** Se abordan los conceptos básicos del trabajo, incluyendo virtualización y cómputo en la nube, el concepto de Infraestructura como Código, la automatización de instancias, los fundamentos de *RHEL* y su relación con arquitecturas híbridas y multicloud.
- **Capítulo 3:** En este capítulo se describe el entorno de trabajo utilizado durante el desarrollo de la tesis. Se presenta la configuración general de la infraestructura, los componentes principales, las funciones de los nodos *Controller* y *Compute*, así como los servicios esenciales de *OpenStack* y el uso de *Horizon*.
- **Capítulo 4:** En este capítulo se explica el proceso manual y la propuesta de automatización. Además, se muestra el código con *Terraform*, incluyendo sus recursos, variables, archivos de configuración y manejo de estados, además de la creación en *RHEL-OpenStack* y su validación en *Horizon*.
- **Capítulo 5:** En este capítulo se muestra la configuración básica de *Microsoft Azure*, la adaptación del código de *Terraform* y el proceso de creación de máquinas virtuales, así como su validación en *Azure Portal*.
- **Capítulo 6:** En este capítulo se realizaron pruebas de paralelismo, destrucción, validación del estado de la infraestructura y *rollback*. También, se presenta una comparación entre *OpenStack* y *Microsoft Azure*. Además, se analizan las ventajas obtenidas mediante la automatización y las limitaciones identificadas durante las pruebas.
- **Capítulo 7:** En este capítulo se presentan las conclusiones obtenidas a partir de las pruebas realizadas, así como algunas propuestas de trabajo futuro.

Capítulo 2

Marco conceptual

2.1. Virtualización

La virtualización permite aprovechar un equipo físico para ejecutar múltiples máquinas virtuales de forma independiente. Esto se logra mediante un *software* llamado hipervisor, el cual administra los recursos del sistema y los vincula con el *hardware*, permitiendo que un mismo equipo pueda dividirse en distintos entornos aislados y seguros.

Para crear estas máquinas virtuales se utilizan imágenes previamente configuradas, las cuales sirven como base para generar nuevas instancias. Estas instancias suelen mantenerse activas durante largos periodos de tiempo, ya que con frecuencia operan como servidores dentro de una infraestructura virtualizada.

En términos económicos, la virtualización suele asociarse con un *CAPEX* alto (*Capital Expenditure*), debido a la inversión inicial en activos físicos como servidores, almacenamiento y equipo de red, y con un *OPEX* bajo (*Operational Expenditure*), relacionado con costos recurrentes de operación como energía, enfriamiento, soporte y mantenimiento. Esto se debe a que, aunque la infraestructura requiere administración interna, no depende de un esquema de pago por uso de un proveedor externo.

Respecto a la capacidad de ajuste de recursos, la virtualización se caracteriza por una escalabilidad vertical, en la que es posible incrementar los recursos asignados a una *VM*, como memoria *RAM* o *CPU*, sin modificar la arquitectura general del sistema. Por esta razón, la virtualización se utiliza con frecuencia en sistemas con estado (*stateful*), donde la información permanece en la misma instancia y puede reutilizarse para procesar solicitudes posteriores.

Aunque en entornos empresariales privados la virtualización suele implementarse bajo un esquema de un solo *tenant* o inquilino, también es posible configurar arquitecturas *multi-tenant*. Esta diferencia depende principalmente de la forma en que se configure la infraestructura y de las políticas de administración establecidas.

2.1.1. Hipervisor

También conocido como supervisor de máquinas virtuales, el hipervisor es un *software* que se encarga de administrar la comunicación entre el *hardware* físico y las máquinas virtuales. Su función es gestionar recursos como la *CPU*, la memoria *RAM*, el almacenamiento y la red, permitiendo que varias *VM* utilicen los recursos del mismo equipo físico de manera controlada. Gracias a ello, es posible ejecutar diferentes sistemas operativos y aplicaciones sobre un mismo servidor físico, manteniendo separados los distintos entornos de trabajo. Dentro de este esquema, el equipo físico

donde se ejecuta el hipervisor recibe el nombre de *host*, mientras que las máquinas virtuales creadas sobre él se conocen como *guest*.

2.1.2. Tipos de Hipervisor

La virtualización puede implementarse mediante dos tipos principales de hipervisores. La diferencia entre ellos se encuentra en la forma en que se ejecutan y de cómo administran los recursos del sistema.

El hipervisor tipo 1, también conocido como *bare metal*, se instala directamente sobre el *hardware* del *host*, por lo que no requiere un sistema operativo anfitrión para funcionar. Su función consiste en administrar recursos como la *CPU*, la memoria, el almacenamiento y la red, asignándolos a cada *guest*. Debido a que trabaja directamente sobre el equipo físico, suele ofrecer un mejor rendimiento y una menor sobrecarga. Por ello, es común encontrar este tipo de hipervisor en centros de datos, servidores empresariales y plataformas de nube.

Por otro lado, el hipervisor tipo 2, conocido como *hosted* o alojado, se ejecuta sobre un sistema operativo como si fuera una aplicación adicional. En este caso, el sistema operativo del *host* conserva el control directo del *hardware*, mientras que el hipervisor accede a dichos recursos a través de ese sistema para crear y operar los *guests*. Como consecuencia, este enfoque suele presentar mayor sobrecarga y menor rendimiento en comparación con el tipo 1. No obstante, ofrece una mayor facilidad de instalación y uso, por lo que resulta adecuado para entornos personales, laboratorios, prácticas académicas y escenarios de prueba donde se requieren varias *VM* sin necesidad de infraestructura dedicada.

2.1.3. KVM, QEMU y libvirt en entorno Linux

KVM (*Kernel-based Virtual Machine*) es un módulo del *kernel* de *Linux* que habilita la virtualización asistida por *hardware* (tipo 1), permitiendo que un sistema *Linux* funcione como hipervisor. Con apoyo de extensiones del procesador, como *Intel VT-x* o *AMD-V*, *KVM* permite ejecutar máquinas virtuales con un nivel adecuado de aislamiento y desempeño para entornos de servidor. En este esquema, cada máquina virtual se ejecuta como un proceso administrado por el sistema anfitrión, con una asignación controlada de recursos.

Por su parte, *QEMU* y *libvirt* cumplen funciones complementarias dentro de la pila de virtualización en sistemas *Linux*. *QEMU* actúa como monitor de máquinas virtuales (*VMM*), proporcionando emulación de *hardware* y una interfaz de bajo nivel para el funcionamiento de cada instancia. En términos operativos, una máquina virtual puede representarse como un proceso *QEMU*. En cambio, *libvirt* funciona como una capa de administración que permite a distintas herramientas gestionar máquinas virtuales, por ejemplo, para iniciarlas, detenerlas, configurarlas o migrarlas, sin interactuar directamente con *QEMU*.

Esta separación también aporta ventajas en materia de seguridad. Debido a que el proceso *QEMU* interactúa con entradas y comandos provenientes del *guest*, se considera más expuesto a acciones maliciosas originadas dentro de la instancia. Por esta razón, se procura ejecutar *QEMU* bajo el principio de mínimo privilegio y dentro de un entorno confinado. En este contexto, *libvirt*, al no ser visible para el *guest*, permite aplicar mecanismos de confinamiento mediante permisos del sistema de archivos, *cgroups* y controles de *SELinux*, con el fin de reducir el riesgo de que una máquina virtual acceda a recursos pertenecientes a otras instancias.

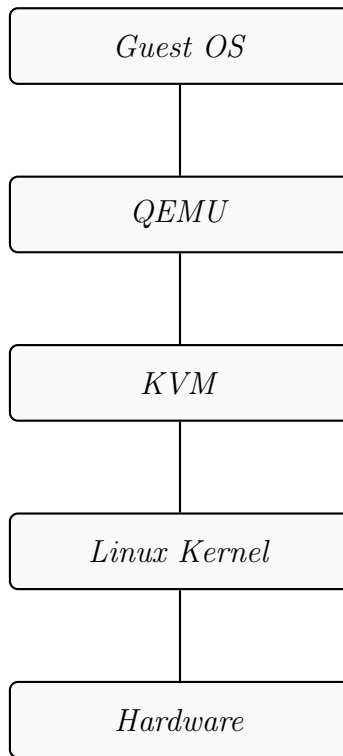


Figura 2.1: Arquitectura de *KVM* y *QEMU* [2].

2.2. Máquinas Virtuales

Una máquina virtual es un entorno informático aislado que emula una computadora completa dentro de un sistema físico. En lugar de depender de *hardware* dedicado para cada servidor o sistema operativo, una *VM* se crea a partir de recursos físicos compartidos, como *CPU*, memoria, almacenamiento y red, los cuales se asignan de manera lógica. Esto permite ejecutar múltiples sistemas operativos y cargas de trabajo sobre un mismo equipo, manteniendo independencia entre entornos y reduciendo la necesidad de infraestructura adicional.

Uno de los beneficios más importantes de las *VM* es el mejor aprovechamiento de los recursos. En servidores físicos es común que parte del *hardware* no se utilice por completo; en cambio, al ejecutar varias *VM* sobre un mismo *host*, es posible aprovechar la capacidad disponible y reducir costos relacionados con la adquisición de equipos, el consumo de energía y el espacio requerido en centros de datos. Además, el aislamiento entre *VM* representa una ventaja adicional en términos de seguridad, ya que un problema dentro de una instancia no necesariamente afecta a las demás. Esta característica también facilita tareas de respaldo, replicación y restauración, lo que resulta útil para la continuidad operativa y la recuperación ante desastres.

Las *VM* también se utilizan con frecuencia en entornos de prueba y desarrollo, ya que permiten crear escenarios controlados para validar aplicaciones o configuraciones sin afectar el entorno principal. Al tratarse de entornos aislados, es posible ejecutar procesos específicos reduciendo el riesgo de interferencia entre aplicaciones. Además, las *VM* permiten ajustar los recursos cuando existe una mayor demanda, ya sea asignando más capacidad a una instancia o creando nuevas *VM*, lo que proporciona una mayor flexibilidad operativa.

Algunas de las características más importantes de las *VM* son el aislamiento entre entornos, la facilidad para moverlas entre equipos, la consolidación de recursos y la administración centralizada.

Sin embargo, es importante diferenciarlas de los contenedores. Mientras una *VM* incluye un sistema operativo completo junto con sus bibliotecas y dependencias, los contenedores suelen compartir el sistema operativo del *host* y encapsulan principalmente la aplicación y los componentes necesarios para ejecutarla. Debido a ello, los contenedores suelen ser más ligeros y rápidos de desplegar, mientras que las *VM* proporcionan un aislamiento más completo al virtualizar un sistema operativo completo.

Las *VM* también tienen una relación importante con los servicios de nube, ya que gran parte de estos servicios utilizan tecnologías de virtualización para administrar los recursos disponibles. En este contexto, los recursos virtualizados se agrupan y gestionan mediante plataformas que permiten el autoservicio, la automatización y la asignación dinámica de recursos. Por esta razón, las *VM* son uno de los mecanismos más utilizados para implementar servicios de infraestructura, especialmente en modelos de *IaaS*, donde el *hardware* físico puede ofrecerse como capacidad computacional bajo demanda.

2.3. Cómputo en la nube

El término *Cloud Computing* hace referencia a un modelo que permite acceder a infraestructura, plataformas y aplicaciones bajo demanda, sin la necesidad de contar con *hardware* propio. Estos recursos son virtualizados y gestionados mediante *software* automatizado, lo que permite a los usuarios acceder a ellos a través de solicitudes, con capacidad de ajuste dinámico y asignación eficiente de recursos.

Algunos de los servicios de nube más utilizados son *Amazon Web Services (AWS)*, *Microsoft Azure* y *Google Cloud Platform (GCP)*. Asimismo, existen plataformas de implementación y gestión de nube de código abierto, como *OpenStack*, cuya operación requiere infraestructura propia y administración especializada. En contraste, los servicios de nube comercial se consumen generalmente mediante esquemas de pago por uso o suscripción.

En comparación con la virtualización tradicional, en *Cloud Computing* es común utilizar plantillas o *templates* y herramientas de Infraestructura como Código (*IaC*), lo que permite replicar la creación de instancias de manera consistente. En este contexto, los recursos suelen tener un ciclo de vida corto o mediano, ya que se crean y eliminan de acuerdo con la demanda.

Si se analiza el aspecto económico, los costos dependen del tipo de nube empleada. En la nube privada, el *CAPEX* es alto debido a la inversión en *hardware* propio, mientras que el *OPEX* es relativamente bajo al tratarse de operación interna. En la nube pública, el *CAPEX* es bajo, pero el *OPEX* es alto y variable, debido al modelo de pago por consumo. Por su parte, la nube híbrida combina ambos esquemas, integrando infraestructura propia con servicios de nube pública. En la tabla 2.1 se presenta una comparación general de estos costos.

Tipo de nube	CAPEX	OPEX
Privada	Alto: inversión en <i>hardware</i> , red, almacenamiento, licencias e infraestructura local.	Bajo-medio: costos de operación interna (energía, soporte, mantenimiento, personal).
Pública	Bajo: no requiere compra de infraestructura propia para iniciar.	Alto-variable: pago por consumo (<i>pay-as-you-go</i>), suscripciones y cargos por servicios utilizados.
Híbrida	Medio-alto: parte de la infraestructura es propia y parte se contrata.	Medio-variable: operación interna más pago por consumo de servicios en nube pública.

Tabla 2.1: Comparación general de *CAPEX* y *OPEX* en nube privada, pública e híbrida.

Respecto a la capacidad de crecimiento de recursos, el cómputo en la nube se caracteriza por una escalabilidad horizontal, en la que es posible añadir más máquinas virtuales o nodos a la infraestructura para responder a incrementos en la demanda. Este enfoque favorece el uso de sistemas sin estado (*stateless*), en los cuales las instancias pueden replicarse o reemplazarse sin afectar la continuidad del servicio. No obstante, el cómputo en la nube no se limita exclusivamente a este tipo de sistemas. Aun así, en arquitecturas modernas orientadas a la alta disponibilidad y la escalabilidad, suele priorizarse el diseño *stateless*.

En muchos entornos de *Cloud Computing* se utilizan arquitecturas *multi-tenant*, lo que permite que múltiples usuarios, proyectos y políticas coexistan sobre una misma infraestructura, manteniendo aislamiento lógico entre ellos y optimizando el uso de los recursos disponibles.

2.3.1. Interacción entre virtualización y cómputo en la nube

La infraestructura de nube utiliza tecnologías como servidores *bare metal*, virtualización y contenedores para compartir y administrar recursos de cómputo a través de una red. En este contexto, la virtualización emplea un hipervisor para gestionar el *hardware* físico y distribuir recursos como procesamiento, memoria y almacenamiento entre distintas *VM*. Cuando este proceso se limita únicamente a la creación y ejecución de máquinas virtuales, se considera virtualización tradicional y no un entorno de nube como tal.

Aunque la virtualización es una parte importante de la nube, por sí sola no es suficiente. Para que los recursos virtuales formen parte de un entorno de cómputo en la nube, es necesario incorporar mecanismos de gestión y automatización que permitan agrupar recursos, administrarlos de forma centralizada y ofrecer autoservicio a los usuarios.

De manera general, un entorno puede considerarse una nube cuando es accesible mediante red, dispone de recursos de tecnologías de la información (*TI*) organizados y permite aprovisionar o escalar recursos de forma ágil. El cómputo en la nube aprovecha las capacidades de la virtualización y añade mecanismos de automatización y gestión que permiten administrar los recursos con mayor flexibilidad.

Aunque la virtualización ofrece ventajas como la consolidación de servidores y un mejor aprovechamiento del *hardware*, el cómputo en la nube incorpora funciones adicionales, por ejemplo el autoservicio, la automatización de la infraestructura y la asignación dinámica de recursos.

2.4. Red Hat Enterprise Linux (*RHEL*)

Red Hat Enterprise Linux (RHEL) es un sistema operativo empresarial basado en *Linux*. Se utiliza como plataforma para ejecutar aplicaciones y servicios en servidores físicos (*bare metal*), máquinas virtuales y entornos de nube. Su uso es común en organizaciones que requieren estabilidad durante largos periodos de operación, compatibilidad con diferentes soluciones de *hardware* y *software*, así como acceso a soporte técnico y actualizaciones controladas.

RHEL permite administrar recursos como procesamiento, memoria, almacenamiento y red. También incorpora tecnologías de virtualización, entre ellas *KVM*, además de herramientas para contenedores y mecanismos de seguridad como *SELinux*. Estas herramientas facilitan la implementación y administración de infraestructura en entornos empresariales. Asimismo, mediante su modelo de suscripción es posible obtener actualizaciones, correcciones de seguridad y soporte especializado.

Otra característica importante de *RHEL* es la disponibilidad de ciclos de soporte extendidos, la compatibilidad con distintas arquitecturas de *hardware* y la posibilidad de integrarse con herramientas de automatización y administración centralizada. Esto permite mantener configuraciones homogéneas y simplificar tareas de operación y mantenimiento. Para este trabajo, *RHEL* se utilizó como sistema operativo base del entorno donde se implementó *OpenStack*. Gracias a sus características de estabilidad, seguridad y compatibilidad, proporcionó una plataforma adecuada para realizar las pruebas de automatización y administración de infraestructura descritas en esta tesis.

2.5. Infraestructura como Código

La Infraestructura como Código (*IaC*) consiste en definir la infraestructura mediante archivos de configuración en lugar de realizar las tareas de forma manual. Bajo este modelo, servidores, redes, almacenamiento y otros componentes se describen como un estado deseado a través de código, lo que permite que la infraestructura sea reproducible, verificable y más sencilla de mantener. En este sentido, uno de los principales problemas que atiende la *IaC* es la variabilidad derivada de los procesos manuales, ya que cuando un despliegue se realiza con pasos distintos o con ajustes no documentados, se generan inconsistencias entre entornos, aumentan los errores y se dificulta la replicación confiable de configuraciones.

En la práctica, la *IaC* se implementa mediante plantillas o archivos que especifican qué recursos deben existir y con qué características. Estos archivos se administran como cualquier otro código fuente: se almacenan en sistemas de control de versiones, se revisan, se modifican de manera colaborativa y permiten rastrear con precisión qué cambió, cuándo ocurrió y por qué se realizó la modificación. Esto permite conocer qué cambios se realizaron y recuperar configuraciones anteriores cuando sea necesario.

Dentro de *IaC* es común encontrar dos enfoques principales: declarativo e imperativo. En el enfoque declarativo se define el resultado esperado, por ejemplo, que debe existir una *VM* con determinadas propiedades, y la herramienta se encarga de ejecutar las acciones necesarias para alcanzar ese estado. En el enfoque imperativo, en cambio, se describen pasos específicos y el orden en que deben ejecutarse, lo que exige un mayor control por parte del administrador. Aunque ambos enfoques buscan automatizar la infraestructura, el modelo declarativo suele facilitar la consistencia y el mantenimiento a largo plazo.

El uso de *IaC* comenzó a ser más frecuente conforme aumentó el uso de la virtualización, los contenedores y los servicios de nube, ya que la infraestructura dejó de ser estática y comenzó a

crearse y modificarse con mayor frecuencia. En este contexto, la *IaC* ofrece ventajas importantes: reduce tareas repetitivas, acelera despliegues, disminuye errores humanos y mejora la uniformidad entre distintos entornos, como desarrollo, pruebas y producción. Asimismo, contribuye a evitar desajustes de configuración, es decir, situaciones en las que un entorno funciona de manera distinta a otro por diferencias acumuladas en su configuración.

Además del aprovisionamiento inicial, la *IaC* también se relaciona con prácticas modernas en las que se automatizan operaciones posteriores, como mantenimiento, monitoreo y actualizaciones, e incluso se integran políticas como código para reforzar aspectos de cumplimiento, seguridad y gestión de riesgos. De esta forma, la infraestructura no solo se crea de manera automatizada, sino que también puede administrarse y gobernarse de forma más consistente.

En términos de aplicación, la *IaC* se utiliza en escenarios como el despliegue de aplicaciones web completas, la creación de infraestructura en nube pública o privada, la integración con *pipelines* de Integración Continua y Entrega/Implementación Continua (*CI/CD*) y la preparación de estrategias de continuidad operativa y recuperación. Al contar con una descripción precisa del entorno, es posible reconstruirlo en otra región o en otra plataforma con menor tiempo de recuperación. Además, *IaC* puede utilizarse tanto en entornos híbridos como *multicloud*, reutilizando únicamente los ajustes necesarios para cada proveedor.

2.5.1. Terraform

Terraform funciona a partir de un flujo de trabajo compuesto por tres etapas principales: definición, planificación y aplicación. Antes de estas etapas, se ejecuta una fase de inicialización mediante el comando `terraform init`, cuyo propósito es preparar el directorio de trabajo, descargar los *providers* necesarios y configurar los componentes requeridos para operar con el archivo de estado (*state*).

En la etapa de definición se redactan los archivos de configuración, por ejemplo archivos *.tf*, en los que se describe el estado deseado de la infraestructura mediante la especificación de recursos y parámetros. Posteriormente, en la etapa de planificación, *Terraform* compara la infraestructura definida en el código con la infraestructura existente, identifica qué elementos deben crearse, modificarse o eliminarse, y genera un plan que permite revisar los cambios antes de ejecutarlos. Finalmente, en la etapa de aplicación, *Terraform* ejecuta dicho plan y realiza las acciones necesarias en el orden adecuado, considerando las dependencias entre recursos. Para mantener consistencia entre ejecuciones, la herramienta utiliza un archivo de estado (*state*) que registra la infraestructura administrada y permite calcular cambios con mayor precisión.

Con el fin de ejemplificar este flujo de trabajo, a continuación se presenta un fragmento mínimo de configuración correspondiente a la etapa de definición. En este ejemplo se declara un *provider* y un recurso de cómputo; posteriormente, la inicialización, la planificación y la aplicación se realizan.

Ejemplo mínimo de definición (archivo *main.tf*):

```
terraform {
  required_providers {
    openstack = {
      source = "terraform-provider-openstack/openstack"
      version = "~> 1.54"
    }
  }
}
```

```

provider "openstack" {
  auth_url      = var.auth_url
  tenant_name   = var.tenant_name
  user_name     = var.user_name
  password     = var.password
  region       = var.region
}

resource "openstack_compute_instance_v2" "vm_demo" {
  name          = "vm-demo"
  image_name    = var.image_name
  flavor_name   = var.flavor_name

  network {
    name = var.network_name
  }
}

```

El primer paso dentro del flujo de trabajo de *Terraform* consiste en ejecutar el comando `terraform init`, cuya función es inicializar el directorio de trabajo y descargar el *provider* declarado en el archivo *main.tf*. En la implementación desarrollada, el bloque `required_providers` define el uso del *provider OpenStack*, indicando tanto su origen como la versión requerida. Posteriormente, el bloque `provider openstack` establece los parámetros de autenticación y conexión con la plataforma, tales como `auth_url`, `tenant_name`, `user_name`, `password` y `region`. Una vez inicializado el entorno, la etapa de planificación se ejecuta mediante el comando `terraform plan`, el cual permite visualizar los cambios que se aplicarían sobre la infraestructura sin ejecutarlos todavía. Finalmente, el comando `terraform apply` materializa la infraestructura definida en el código, respetando las dependencias entre recursos y actualizando el archivo de estado correspondiente para ejecuciones posteriores.

En el archivo *main.tf*, los elementos principales de *Terraform* pueden identificarse directamente dentro de la estructura del código. En primer lugar, el *provider* funciona como el componente encargado de establecer la comunicación entre *Terraform* y la plataforma objetivo, en este caso *OpenStack*. Por otra parte, el bloque `resource "openstack_compute_instance_v2"` representa el recurso de infraestructura administrado por *Terraform*, correspondiente a la creación de una máquina virtual dentro del entorno. Asimismo, las variables permiten definir distintos valores del despliegue, como la imagen, el *flavor* o la red que se utilizará. Esto hace posible reutilizar el mismo código sin necesidad de modificar directamente la definición principal de los recursos. La estructura declarativa del archivo también ayuda a mantener una organización clara de la infraestructura y facilita el seguimiento de los cambios realizados dentro del entorno automatizado.

Entre las ventajas más importantes de *Terraform* se encuentran la automatización, la consistencia y la portabilidad. Al definir la infraestructura mediante archivos declarativos, disminuye la necesidad de realizar tareas repetitivas y se reducen errores asociados con configuraciones manuales. Además, un mismo conjunto de archivos puede utilizarse para recrear entornos similares en distintos momentos, lo que resulta útil en etapas de desarrollo, pruebas y producción. También permite trabajar en entornos híbridos y *multicloud*, realizando únicamente los ajustes necesarios para cada proveedor.

En comparación con otras herramientas de *IaC*, *Terraform* está orientado principalmente al aprovisionamiento de infraestructura. Por ejemplo, herramientas como *Ansible* suelen emplearse para configurar y administrar sistemas que ya fueron creados, mientras que *Terraform* se utiliza para crear y gestionar los recursos desde el inicio. A diferencia de soluciones ligadas a un único proveedor, utiliza *providers* intercambiables y mantiene una sintaxis similar entre diferentes plataformas. También adopta un enfoque declarativo, lo que facilita comprender el estado deseado de la infraestructura y compararlo con el estado existente.

2.5.2. Microsoft Azure

Microsoft Azure es una plataforma de nube pública que ofrece distintos servicios para el almacenamiento, administración y procesamiento de información. Además de proporcionar capacidad de infraestructura, incluye herramientas orientadas al análisis de datos, la seguridad y la operación de aplicaciones. Esto permite que organizaciones y usuarios accedan a recursos tecnológicos sin depender únicamente de infraestructura local. Dentro de la plataforma se encuentran modelos de servicio como *Software as a Service (SaaS)*, *Platform as a Service (PaaS)* e *Infrastructure as a Service (IaaS)*, incluyendo el uso de máquinas virtuales para ejecutar aplicaciones y servicios bajo demanda.

Para ofrecer estos recursos, *Azure* utiliza tecnologías de virtualización que permiten asignar capacidad de cómputo de forma flexible y escalable. El acceso a los servicios se controla mediante credenciales y permisos, de modo que únicamente los usuarios autorizados puedan administrarlos o utilizarlos. Además, la plataforma mantiene compatibilidad con múltiples aplicaciones y herramientas, especialmente dentro del ecosistema de *Microsoft*, aunque también admite soluciones desarrolladas por terceros.

La infraestructura de *Azure* no se limita únicamente a la virtualización. También incorpora componentes como Redes Definidas por *Software (SDN)*, sistemas distribuidos de almacenamiento, mecanismos de orquestación de recursos y diferentes herramientas de seguridad. La integración de estos elementos permite ofrecer servicios que pueden crecer de acuerdo con la demanda y administrarse desde una plataforma centralizada.

Entre las funcionalidades que ofrece la plataforma se encuentran el procesamiento y análisis de información, los servicios de infraestructura y plataforma, la integración de aplicaciones de terceros y el despliegue de soluciones de red, como *VPN*. Asimismo, incorpora mecanismos de seguridad orientados a la protección y administración de los recursos desplegados.

Otro aspecto relevante de *Azure* es su flexibilidad, ya que permite integrar distintas aplicaciones y acceder a los servicios desde diversos dispositivos. También utiliza un modelo de pago por consumo, donde los costos dependen de los recursos utilizados. Además, dispone de una amplia variedad de herramientas que facilitan la construcción, el despliegue y la operación de servicios digitales dentro del ecosistema de *Microsoft*.

2.6. Automatización de despliegues

La automatización de despliegues consiste en utilizar herramientas y procesos que permitan entregar *software* e infraestructura de forma más rápida y consistente. En lugar de depender de actividades manuales, donde una persona ejecuta comandos o realiza configuraciones paso a paso, se definen procedimientos que pueden ejecutarse automáticamente y repetirse bajo las mismas

condiciones. De esta manera, el despliegue pasa de ser una tarea manual a un proceso controlado y documentado.

La necesidad de automatizar aparece debido a las limitaciones de los despliegues manuales. Cuando cada implementación se realiza de forma diferente, pueden surgir errores humanos, diferencias entre entornos de desarrollo, pruebas y producción, así como tiempos más largos de implementación. Además, resulta más complicado identificar qué cambios se realizaron en cada ejecución. La automatización ayuda a reducir estas variaciones y facilita el seguimiento de los cambios realizados durante el proceso. En muchos casos, la automatización se integra con esquemas de integración y entrega continua. Bajo este enfoque, una modificación en el código puede iniciar automáticamente una serie de tareas, como compilación, ejecución de pruebas y validaciones antes del despliegue. Para ello se utilizan herramientas capaces de ejecutar *scripts* o configuraciones previamente definidas. Entre los beneficios de la automatización de despliegues se encuentran la reducción de errores humanos, una mayor repetibilidad de los procesos y una disminución del tiempo necesario para realizar implementaciones. También permite que el procedimiento no dependa exclusivamente de una persona, ya que las tareas quedan definidas dentro del propio sistema. En entornos basados en virtualización y servicios de nube, esta práctica resulta especialmente útil porque facilita la creación, modificación y administración de recursos de manera más eficiente.

Capítulo 3

Entorno base en RHEL y OpenStack

3.1. Referencia del entorno base

El entorno base utilizado en el presente trabajo toma como referencia la tesis titulada *Análisis de seguridad en una nube privada orquestada por OpenStack* [3]. En dicho trabajo se describe la instalación y configuración de una infraestructura de nube privada basada en *OpenStack*, implementada sobre *Red Hat Enterprise Linux (RHEL)*, considerando una arquitectura compuesta por un nodo *Controller* y un nodo *Compute*. Asimismo, se detalla el proceso de despliegue de los servicios principales de *OpenStack*, así como la habilitación de la interfaz web de administración *Horizon*. La figura 3.1 muestra los módulos principales que intervienen en la creación de máquinas virtuales en *OpenStack*: *Horizon*, *Nova*, *Glance*, *Neutron* y *Keystone*. Estos componentes permiten administrar el entorno, crear instancias, gestionar imágenes, configurar redes y controlar la autenticación. Además, pueden integrarse otros módulos según las necesidades de la infraestructura. Todos estos servicios se ejecutan sobre *RHEL*, utilizado como sistema operativo base.

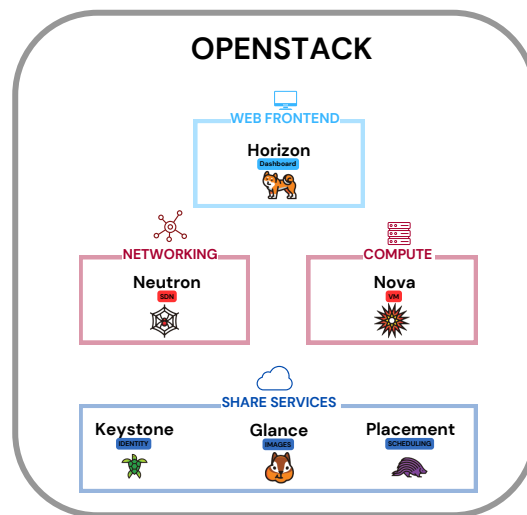


Figura 3.1: Módulos básicos de *OpenStack*.

Asimismo, se documenta el orden de instalación de los componentes fundamentales para una implementación base, el cual coincide con el esquema mínimo recomendado en la guía oficial de instalación de *OpenStack* para la versión 2024,2 *Dalmatian* [4]. Este proceso establece la instalación

secuencial de los servicios de identidad (*Keystone*), imágenes (*Glance*), componente de ubicación (*Placement*), cómputo (*Nova*) y redes (*Neutron*), finalizando con la interfaz web *Horizon*.

A partir de esta implementación funcional, el presente trabajo no replica el procedimiento de instalación, sino que parte de una infraestructura previamente operativa.

Adicionalmente, en el nodo de cómputo se realizaron distintos ajustes de configuración relacionados con la operación de los servicios de virtualización y red utilizados por *OpenStack*. Estas modificaciones tuvieron como propósito asegurar el funcionamiento correcto del entorno durante el despliegue automatizado de máquinas virtuales mediante *Terraform*.

En particular, se modificó el archivo `/etc/libvirt/qemu.conf`, el cual define parámetros asociados a la ejecución de las máquinas virtuales mediante *QEMU/KVM*. Como se muestra en el código 3.1, se estableció que las instancias se ejecuten bajo el usuario y grupo *nova*, correspondientes al servicio de cómputo, y se desactivó el mecanismo de cambio dinámico de propietario mediante el parámetro `dynamic_ownership = 0`. Esta configuración permite asegurar que las instancias puedan gestionarse correctamente dentro del entorno de virtualización, evitando conflictos de permisos entre procesos del sistema y archivos asociados a las máquinas virtuales. El contenido completo de esta configuración puede consultarse en el apéndice C.1.

```
user = "nova"
group = "nova"
dynamic_ownership = 0
```

Código 3.1. Configuración del archivo `qemu.conf`

Asimismo, en el nodo *Compute* se modificó el archivo `/etc/neutron/neutron.conf`, específicamente dentro de la sección `[agent]`, con el propósito de definir el mecanismo utilizado por *Neutron* para ejecutar operaciones privilegiadas relacionadas con red. Como se muestra en el código 3.2, se configuró el parámetro `root_helper`, permitiendo que el servicio utilice *rootwrap* para ejecutar comandos con privilegios controlados dentro del entorno. El detalle completo de esta configuración puede consultarse en el apéndice C.2.

```
[agent]
root_helper = sudo neutron-rootwrap /etc/neutron/rootwrap
.conf
```

Código 3.2. Configuración del parámetro `root_helper` en `neutron.conf`

De igual forma, en el nodo *Controller* se realizaron ajustes dentro del archivo `/etc/hosts`, con el propósito de mejorar la resolución local de nombres entre los distintos componentes de la infraestructura *OpenStack*. En particular, se añadió una entrada asociada a la interfaz externa del controlador, permitiendo identificar el nodo mediante el nombre *controller-ext*. Esta configuración facilitó la comunicación entre los servicios internos de la plataforma y la interfaz externa utilizada para el acceso al entorno. Como se muestra en el código 3.3, se añadió la dirección correspondiente al nodo *controller-ext* dentro del archivo de configuración. El contenido completo del archivo puede consultarse en el apéndice D.1.

```
x.x.x.x controller-ext
```

Código 3.3. Configuración del archivo `/etc/hosts` en el nodo *Controller*

En conjunto, estas configuraciones permitieron mejorar la integración entre los servicios de virtualización, red y resolución de nombres dentro del entorno utilizado para las pruebas de automatización y despliegue de infraestructura.

3.2. Proceso manual de creación de *VM*

En un entorno *OpenStack* administrado mediante *Horizon*, el despliegue manual de una máquina virtual sigue un flujo operativo definido por la propia plataforma. A continuación se describe este procedimiento con el propósito de caracterizar el enfoque tradicional basado en intervención humana, el cual posteriormente es sustituido por un esquema automatizado. La figura 3.2 muestra la interfaz de acceso al panel web *Horizon*, desde la cual el usuario se autentica mediante sus credenciales para ingresar al entorno de administración de *OpenStack* e iniciar la gestión y el despliegue de recursos virtuales, como la creación de instancias. Antes de lanzar una instancia (*VM*), se preparan mecanismos de acceso seguro, como la generación de un par de llaves (*key pair*) para autenticación por clave pública y la configuración de reglas dentro de los *security groups*. En particular, es necesario habilitar una regla de entrada que permita conexiones por *SSH*, ya que de forma predeterminada las conexiones entrantes se encuentran restringidas. Posteriormente, la creación de la instancia se realiza mediante un asistente de lanzamiento, en el cual se selecciona la imagen base del sistema operativo, se asigna un nombre identificador y se elige un *flavor* que determina recursos como *V-CPU*, memoria *RAM* y, en algunos casos, características de almacenamiento. El *flavor*, también denominado “sabor”, es una plantilla de recursos que define la capacidad de la máquina virtual, especificando principalmente el número de *V-CPU*, la cantidad de memoria *RAM* y, según la configuración del entorno, el tamaño del disco duro.

En la misma etapa se define la conectividad de la instancia, asignando la red correspondiente, ya sea externa o privada, así como el grupo de seguridad y el par de llaves asociados. Cuando la instancia se despliega dentro de una red privada, el acceso hacia redes externas suele requerir configuración adicional. En este esquema, se define una red privada con su direccionamiento y, comúnmente, con *DHCP* habilitado. Posteriormente, se configura un *router* virtual que enlaza dicha red con la red externa. Para permitir acceso directo desde fuera del entorno, se asigna una *IP* flotante, entendida como una dirección *IP* pública que puede asociarse dinámicamente a una máquina virtual para proporcionar conectividad desde redes externas. Esta dirección se obtiene de un *pool* disponible y permite que la instancia sea accesible desde el exterior. Finalmente, cuando se requiere almacenamiento adicional, se recurre a la creación de volúmenes independientes de tipo *block storage*. Estos volúmenes se asocian a la instancia y, desde el sistema operativo invitado, se formatean y montan para su utilización. En conjunto, este flujo manual evidencia que el despliegue depende de múltiples decisiones relacionadas con seguridad, red, cómputo y almacenamiento. Además, requiere seleccionar y configurar diversos parámetros en cada ejecución, lo que puede generar diferencias entre instancias similares. Por ello, el presente trabajo propone sustituir este enfoque por un procedimiento estandarizado basado en Infraestructura como Código.

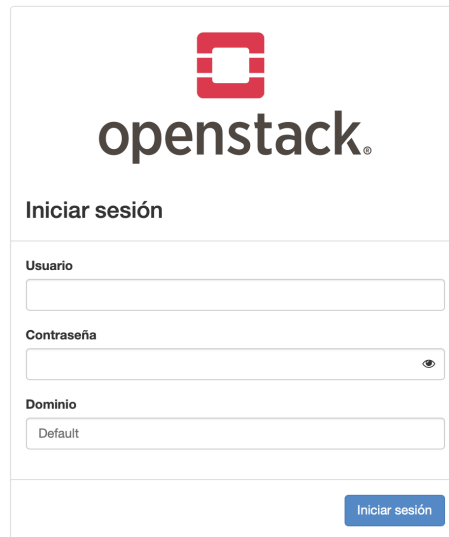


Figura 3.2: Interfaz de *Horizon*: pantalla de inicio de sesión.

3.3. Limitaciones identificadas en el entorno

El despliegue manual de máquinas virtuales en *OpenStack* mediante *Horizon* presenta limitaciones operativas que afectan la uniformidad, la trazabilidad y la capacidad de reproducir infraestructura de manera consistente. Estas limitaciones no dependen únicamente de la plataforma, sino del hecho de que la creación de instancias se basa en decisiones humanas repetidas en cada ejecución, lo que introduce variabilidad y dificulta el control del entorno a medida que aumenta el número de *VM*.

Las principales limitaciones identificadas son las siguientes:

- **Alta dependencia del operador.** En un despliegue manual, la selección de parámetros como *imagen*, *flavor*, red, *security groups*, *key pairs*, volúmenes y asignación de *IP* (fija o flotante) recae en la persona que ejecuta el procedimiento. Aunque la interfaz guía el proceso, la consistencia entre instancias depende de que el operador recuerde, documente y replique exactamente las mismas elecciones. Esto hace que el despliegue sea vulnerable a errores por omisión, por ejemplo, no asociar un *security group* adecuado, o por selección incorrecta, como elegir un *flavor* distinto al requerido, generando instancias que pueden iniciar correctamente pero no cumplir con los requisitos técnicos esperados.
- **Ausencia de un mecanismo formal de estandarización.** La estandarización implica que la configuración correcta quede definida de manera explícita y reutilizable. Sin embargo, cuando el despliegue se realiza desde la interfaz gráfica, los parámetros quedan asociados a cada instancia de forma individual, sin que exista un archivo declarativo que describa el estado deseado de la infraestructura. En consecuencia, dos *VM* creadas con el mismo propósito pueden presentar diferencias relevantes, por ejemplo, en la red asignada, las reglas de acceso o la capacidad de almacenamiento, incluso si ambas aparecen como activas en *Horizon*.

- **Limitaciones de trazabilidad y auditoría.** En un esquema manual no se genera automáticamente un historial estructurado que permita identificar con claridad qué se creó, con qué parámetros, en qué momento, por quién y bajo qué justificación técnica. Si ocurre una incidencia posterior, por ejemplo, una *VM* inaccesible por *SSH* o una instancia sin conectividad externa, la causa debe verificarse inspeccionando cada componente en la interfaz, revisando configuraciones dispersas y reconstruyendo el procedimiento de manera retrospectiva. Esto incrementa el tiempo de diagnóstico y dificulta la comparación entre despliegues.
- **Replicabilidad limitada del entorno.** En escenarios donde se requiere reproducir un conjunto de instancias de manera periódica, como laboratorios, pruebas, despliegues iterativos o expansión de capacidad, el procedimiento manual obliga a repetir las mismas decisiones múltiples veces. A medida que el número de *VM* crece, el esfuerzo se vuelve acumulativo. Por ejemplo, crear diez instancias implica repetir diez veces el mismo flujo, con el riesgo de introducir variaciones no intencionales. Esto puede afectar directamente la consistencia del entorno y limita la posibilidad de garantizar que un despliegue actual sea equivalente a uno previo.
- **Complejidad en la configuración de red.** La configuración manual incrementa la complejidad cuando se utilizan esquemas con red privada, *router* e *IP* flotante. En este tipo de topologías se requiere coordinar varios componentes: creación de la red privada con el direccionamiento correcto, habilitación de *DHCP*, creación del *router*, enlace del *router* a la red externa, adición de la interfaz hacia la red privada y, finalmente, asignación de una *IP* flotante a la instancia. Debido a que cada paso depende del anterior, un error en un solo punto puede provocar que la instancia quede aislada o sin salida a la red. Al no existir una definición declarativa que agrupe estas dependencias, la verificación de consistencia se vuelve manual y propensa a fallos.
- **Variabilidad en el manejo del almacenamiento adicional.** El manejo de almacenamiento adicional mediante volúmenes también introduce variabilidad. En un procedimiento manual, la creación del volumen, su asociación a la instancia y su preparación dentro del sistema operativo, incluyendo formateo y montaje, dependen de acciones separadas que pueden ejecutarse con diferencias entre instancias, como distintos puntos de montaje, sistemas de archivos o tamaños inconsistentes. Como resultado, el entorno puede presentar *VM* con configuraciones heterogéneas, lo que dificulta su administración y mantenimiento.
- **Limitaciones en la portabilidad del procedimiento.** La creación manual mediante *Horizon* está estrechamente vinculada a la interfaz gráfica y a los mecanismos propios del entorno *OpenStack*. Por esta razón, al trasladar el proceso de despliegue hacia otra plataforma, como *Microsoft Azure*, no existe una representación directa que pueda reutilizarse automáticamente. Aunque la lógica general de la infraestructura puede reproducirse, es necesario redefinir y adaptar la configuración de acuerdo con los recursos y servicios disponibles en cada proveedor, lo que incrementa el esfuerzo de implementación y dificulta mantener una configuración homogénea entre distintos entornos.

En conjunto, el enfoque manual permite desplegar instancias funcionales, pero dificulta garantizar uniformidad, control y repetibilidad a medida que la infraestructura crece. Por ello, el problema identificado se orienta a sustituir este procedimiento por un esquema estandarizado basado en Infraestructura como Código, donde la configuración quede definida de forma declarativa, versionable y reproducible.

Capítulo 4

Automatización de la creación de máquinas virtuales

4.1. Diseño del proceso automatizado

El diseño del proceso automatizado se planteó a partir de las limitaciones identificadas en el procedimiento manual de creación de máquinas virtuales. En el flujo tradicional, el operador debe seleccionar manualmente diversos parámetros dentro de la interfaz gráfica de *Horizon*, lo que introduce variabilidad entre despliegues y aumenta la probabilidad de inconsistencias. Con el fin de reducir estas dependencias operativas, se propuso trasladar la definición del despliegue a un esquema declarativo que permitiera ejecutar el proceso de forma controlada y repetible.

Para ello se adoptó el enfoque de Infraestructura como Código, utilizando *Terraform* como herramienta de automatización. Este enfoque permite describir los recursos necesarios para el despliegue de una máquina virtual mediante archivos de configuración, de modo que la infraestructura deseada pueda definirse, revisarse y ejecutarse de forma consistente. En lugar de realizar cada paso manualmente desde la interfaz de administración, el estado esperado de la infraestructura se expresa mediante código.

Es importante señalar que el diseño del proceso automatizado se apoya en un entorno previamente configurado dentro de *OpenStack*. Los recursos fundamentales del entorno, tales como el proyecto, las credenciales administrativas, la red, la imagen del sistema operativo y los *flavors*, ya se encuentran definidos dentro de la plataforma y son compartidos entre distintos despliegues. Debido a que estos elementos forman parte de la infraestructura base, el proceso automatizado no los crea nuevamente, sino que los reutiliza como recursos existentes.

Bajo esta lógica, el *script* desarrollado se enfoca específicamente en automatizar la creación de instancias de cómputo a partir de dichos recursos compartidos. La máquina virtual se despliega referenciando la red disponible, la imagen previamente cargada en el servicio de imágenes (*Glance*) y el *flavor* configurado en el entorno. De este modo, la automatización se integra con la infraestructura existente sin modificar su estructura base.

Otro aspecto considerado en el diseño fue la reutilización del código. La configuración se estructuró de manera que los parámetros relevantes del despliegue pudieran modificarse sin alterar la lógica principal del *script*. Esta separación entre la definición de recursos y los parámetros de configuración permite adaptar el despliegue a distintos escenarios, manteniendo una misma base de automatización.

En términos operativos, el proceso automatizado sigue un flujo estructurado en el que primero

se inicializa el entorno de trabajo, posteriormente se valida la configuración definida y finalmente se aplica la infraestructura declarada. Este mecanismo permite verificar los cambios antes de ejecutarlos y facilita el control del despliegue dentro del entorno *OpenStack*.

En conjunto, el diseño propuesto busca estandarizar el despliegue de máquinas virtuales aprovechando los recursos ya disponibles en la plataforma. Al reutilizar la infraestructura existente y trasladar la definición del despliegue a un esquema declarativo.

4.2. Desarrollo del *script* en *Terraform*

Bajo este enfoque, el *script* se estructuró de manera que mantuviera una separación clara entre la definición de recursos y los parámetros variables del despliegue. Esta organización permite conservar una lógica reutilizable y facilita el mantenimiento del código, ya que los ajustes en nombres, credenciales o configuraciones específicas no requieren modificar toda la definición de la infraestructura.

En la siguiente sección se describen los elementos principales que conforman el *script*, comenzando por la definición de los recursos necesarios para el despliegue y continuando con la organización de variables y archivos de configuración que permiten parametrizar y reutilizar la solución desarrollada.

4.2.1. Definición de recursos

La definición de recursos en el *script* desarrollado se centró en automatizar el despliegue de una instancia dentro de un entorno *OpenStack* previamente configurado. Para ello, se utilizó el recurso `openstack_compute_instance_v2`, el cual permite describir mediante código las características principales de una máquina virtual y ejecutar su creación dentro del proyecto correspondiente.

Es importante señalar que el *script* no se diseñó para crear toda la infraestructura del entorno, sino para reutilizar recursos que ya se encontraban disponibles dentro de la plataforma. En particular, el proyecto, el usuario, la red, la imagen y el *flavor* habían sido configurados previamente en *OpenStack*. Por esta razón, el código únicamente hace referencia a estos elementos existentes para realizar el despliegue de la instancia. La figura 4.1 muestra los proyectos disponibles en *OpenStack Horizon*. Específicamente, *admin* y *service* registrados en la plataforma *OpenStack*. Para el despliegue de las máquinas virtuales utilizadas en esta tesis se trabajó sobre el proyecto *admin*.

openstack.

Default • admin

admin

Proyecto

Administrador

Vista general

Computación

Volumen

Red

Sistema

Identidad

Identidad / Proyectos

Proyectos

Nombre de proyecto =

Filtrar

+ Crear proyecto

Eliminar Proyectos

Mostrando 2 artículos

☐	Name	Description	Project ID	Domain Name	Enabled	Actions
☐	service		51ca9860644945688a9a4f8613b271ad	Default	Sí	Administrar Miembros
☐	admin	Bootstrap project for initializing the cloud.	5b9dfcb866ad42818d9b3701feddaa7b	Default	Sí	Administrar Miembros

Mostrando 2 artículos

Figura 4.1: Vista de los proyectos disponibles en *OpenStack Horizon*.

En la implementación realizada se empleó la imagen *cirros*, la cual ya se encontraba registrada en el servicio de imágenes de *OpenStack*. Esta imagen se utilizó como sistema operativo base para la máquina virtual, ya que se trata de una distribución ligera comúnmente utilizada para pruebas dentro de entornos de virtualización. Asimismo, el *flavor* seleccionado fue *m1.tiny*, el cual define los recursos de cómputo asignados a la instancia, tales como el número de *vCPU*, la memoria *RAM* y el almacenamiento asociado. Este tipo de plantillas permite estandarizar las características de las máquinas virtuales dentro del entorno. La figura 4.2 muestra el registro del *flavor* dentro de *OpenStack*. La información indica que este *flavor* dispone de 1 *vCPU*, 512 MB de memoria *RAM* y 5 GB de almacenamiento. Estas características fueron suficientes para ejecutar las máquinas virtuales empleadas en el entorno experimental sin requerir una cantidad elevada de recursos de cómputo.

Flavor Name	VCPUs	RAM	Root Disk	Ephemeral Disk	Swap Disk	RX/TX factor	ID	Public	Metadata	Actions
m1.sma	1	2GB	20GB	0GB	0MB	1.0	77416355-72b0-4a3e-bca2-8ada338ebf5a	Si	No	Actualizar metadatos
m1.tiny	1	512MB	5GB	0GB	0MB	1.0	00b8ada9-4d90-441b-bc42-017f5664a806	Si	No	Actualizar metadatos

Figura 4.2: Registro del *flavor* *m1.tiny* dentro del panel de administración de *OpenStack Horizon*.

Por otra parte, la instancia se conecta a la red *private*, previamente configurada en la infraestructura de *OpenStack*. Esta red proporciona conectividad a la máquina virtual dentro del entorno virtualizado y permite su comunicación con otros recursos definidos en la plataforma.

Desde la perspectiva del *script*, estos recursos no se crean nuevamente, sino que se reutilizan mediante referencias a configuraciones previamente existentes dentro del entorno. Esto permite que el recurso `openstack_compute_instance_v2` utilice imágenes, redes y parámetros ya configurados en la plataforma, evitando duplicidad de infraestructura y manteniendo coherencia con la preparación previa realizada en *OpenStack*.

4.2.2. Variables y archivos de configuración

La organización del *script* se planteó con el propósito de mantener una configuración clara y reutilizable para el despliegue automatizado de la instancia. Aunque la implementación desarrollada se concentró en un archivo principal denominado *main.tf*, en este mismo archivo se definieron tanto la configuración del proveedor como las variables necesarias para parametrizar el despliegue. Esta estructura permitió concentrar en un solo archivo los elementos esenciales del proceso automatizado y facilitar su ejecución dentro del entorno de prueba.

En primer lugar, el archivo *main.tf* contiene la definición del proveedor de *OpenStack*, mediante la cual se establecen los datos necesarios para la autenticación y conexión con la plataforma. Entre estos parámetros se encuentran la dirección del servicio de identidad, el nombre de usuario, la contraseña, el proyecto, el dominio y la región. Gracias a esta configuración, *Terraform* puede comunicarse con la infraestructura de *OpenStack* y ejecutar el despliegue de la máquina virtual dentro del entorno previamente preparado.

Además de la configuración del proveedor, en el mismo archivo se declararon variables para parametrizar algunos elementos del despliegue. En particular, se definieron variables asociadas al nombre de la máquina virtual, el *flavor*, la imagen y la red. Esto permite modificar ciertos valores del despliegue sin alterar directamente la lógica principal del recurso declarado. En la implementación realizada, las variables utilizadas fueron *nombre_vm*, *flavor*, *imagen* y *red*, con valores por defecto que corresponden a los recursos disponibles en el entorno, tales como *vm-terraform-1*, *m1.tiny*, *cirros* y *private*. El código 4.1 muestra el archivo principal empleado en la automatización, mientras que su contenido completo puede consultarse en el Apéndice A.1.

```
terraform {
  required_providers {
    openstack = {
      source = "terraform-provider-openstack/openstack"
      version = ">=1.54.0"
    }
  }
}

provider "openstack" {
  auth_url    = "http://ip/v3"
  user_name   = "admin"
  password    = "contrasea"
  tenant_name = "admin"
  domain_name = "Default"
  region      = "RegionOne"
  insecure    = true
}

variable "nombre_vm" {
  default = "vm-terraform-1"
}

variable "flavor" {
  default = "m1.tiny"
}

variable "imagen" {
  default = "cirros"
}

variable "red" {
  default = "private"
}

data "openstack_networking_network_v2" "red" {
  name = var.red
}

resource "openstack_compute_instance_v2" "instancia" {
  count = 5
}
```

```
name = "vm-terraform-${count.index+1}"
flavor_name = var.flavor
image_id = "9681ce2d-b638-43b5-ae1a-4df0d2203df1"
availability_zone = "nova"
network {
  uuid = data.openstack_networking_network_v2.red.id
}
}
```

Código 4.1. Archivo principal *main.tf* utilizado en la automatización.

El uso de variables aporta flexibilidad al *script*, ya que separa la configuración modificable de la definición estructural del despliegue. De esta manera, si se requiere cambiar el nombre de la instancia o utilizar una imagen, un *flavor* o una red distinta, la modificación puede realizarse en los parámetros definidos sin necesidad de reescribir el bloque principal del recurso. Esto favorece el mantenimiento del código y mejora su reutilización en distintos escenarios dentro del mismo entorno.

Otro elemento importante dentro de la configuración fue el uso del bloque *data*, el cual permite consultar información de recursos ya existentes en la plataforma. En este caso, se utilizó el recurso `openstack_networking_network_v2` para recuperar la información asociada a la red especificada en la variable correspondiente. Con ello, el *script* no crea una red nueva, sino que obtiene la referencia de una red ya registrada en *OpenStack* y la utiliza posteriormente en la definición de la instancia.

Asimismo, dentro de la definición del recurso principal se utilizó el parámetro `count = 5`, el cual permite indicar la cantidad de máquinas virtuales que se desean crear automáticamente. Gracias a este mecanismo, *Terraform* genera múltiples instancias a partir de una sola definición de recurso, asignando a cada máquina un identificador consecutivo mediante la expresión `count.index + 1`. En consecuencia, el despliegue automatizado produce las instancias *vm-terraform-1*, *vm-terraform-2*, hasta *vm-terraform-5*, reduciendo la necesidad de repetir manualmente la configuración para cada máquina virtual.

Desde el punto de vista de la organización del código, la integración del proveedor, las variables, la consulta de datos y el recurso principal dentro de un mismo archivo permitió desarrollar una solución funcional y compacta para el despliegue automatizado. Si bien en implementaciones de mayor escala suele ser recomendable separar estos componentes en varios archivos, para los fines de este trabajo la estructura empleada resultó suficiente para demostrar la viabilidad de la automatización y mantener una comprensión directa del flujo de configuración.

4.3. Despliegue automatizado en *RHEL/OpenStack*

La figura 4.3 muestra la ejecución de los comandos principales del proceso de automatización. A partir de esta evidencia, puede observarse que *Terraform* no solo permite desplegar infraestructura, sino también verificar si el entorno real coincide con la configuración definida en código. En este sentido, el despliegue automatizado se convierte en un mecanismo de control y validación, además de ser una herramienta para la creación de recursos.

Una vez definida la configuración del despliegue en el archivo *main.tf*, la automatización se ejecutó dentro del entorno *RHEL/OpenStack* desde el nodo *Controller*. Esta etapa tuvo como objetivo materializar la definición declarativa de la infraestructura mediante *Terraform*.

El proceso comenzó con la ejecución del comando `terraform init`, cuya función consiste en inicializar el directorio de trabajo y preparar el entorno para la ejecución del *script*. Durante esta fase, *Terraform* verifica la configuración del proyecto y carga el proveedor necesario para interactuar con *OpenStack*. Para este trabajo, esta inicialización permitió reutilizar el proveedor previamente instalado, dejando listo el entorno para las siguientes etapas del despliegue.

Posteriormente, se ejecutó el comando `terraform plan`, mediante el cual *Terraform* compara la infraestructura definida en el *script* con el estado actual del entorno. Esta etapa resulta importante porque permite validar, antes de aplicar cambios, qué acciones serían necesarias para llevar la infraestructura real al estado deseado. En la ejecución realizada, el sistema indicó que no existían diferencias entre la configuración declarada y la infraestructura ya registrada, lo que confirmó que el recurso definido en el *script* coincidía con el estado actual del entorno.

Finalmente, se empleó el comando `terraform apply -auto-approve`, encargado de aplicar la configuración definida y ejecutar las acciones requeridas sobre la infraestructura. En este caso, la salida mostró que no se agregaron, modificaron ni destruyeron recursos, debido a que la instancia ya se encontraba desplegada y registrada dentro del estado administrado por *Terraform*. Este resultado no representa un error, sino una confirmación de consistencia entre la definición declarativa y la infraestructura existente.

```
[root@controller terraform-openstack]# terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of terraform-provider-openstack/openstack from the dependency lock file
- Using previously-installed terraform-provider-openstack/openstack v3.4.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
[root@controller terraform-openstack]# terraform plan
data.openstack_networking_network_v2.red: Reading...
data.openstack_networking_network_v2.red: Read complete after 1s [id=86eaa5ae-a89f-4300-bd28-4e148eb34708]
openstack_compute_instance_v2.instancia: Refreshing state... [id=35cf9f65-0af3-4c8e-89b6-99c317c71301]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.
[root@controller terraform-openstack]# terraform apply -auto-approve
data.openstack_networking_network_v2.red: Reading...
data.openstack_networking_network_v2.red: Read complete after 1s [id=86eaa5ae-a89f-4300-bd28-4e148eb34708]
openstack_compute_instance_v2.instancia: Refreshing state... [id=35cf9f65-0af3-4c8e-89b6-99c317c71301]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
[root@controller terraform-openstack]#
```

Figura 4.3: Ejecución de los comandos `terraform init`, `terraform plan` y `terraform apply` durante el despliegue automatizado de la instancia en *OpenStack*.

Como se observa en la figura 4.3, la salida de *Terraform* indica que no fue necesario agregar, modificar ni eliminar recursos (*0 added, 0 changed, 0 destroyed*). Este resultado se obtuvo de manera intencional como parte de una verificación controlada posterior al despliegue inicial, con el propósito de demostrar que la infraestructura previamente creada coincide con la configuración declarada en el *script*. En otras palabras, la ejecución permitió comprobar que el recurso ya existente

se encontraba alineado con el estado deseado definido en código. Este comportamiento evidencia el mecanismo de control de estado de *Terraform*, mediante el cual es posible validar la consistencia entre la infraestructura real y la configuración declarativa sin introducir cambios adicionales cuando estos no son necesarios.

En conjunto, esta etapa demuestra que la automatización desarrollada es funcional dentro del entorno *RHEL-OpenStack* y que la definición en código permite mantener coherencia entre lo configurado y lo desplegado.

4.4. Visualización y validación en *Horizon*

Una vez ejecutado el despliegue automatizado, la validación del resultado se realizó desde la interfaz web *Horizon*, con el propósito de verificar que la instancia definida en el *script* se encontrara registrada correctamente dentro del entorno *OpenStack*. Esta etapa es relevante porque permite contrastar la infraestructura declarada en *Terraform* con la representación visual de los recursos disponibles en la plataforma.

La comprobación se llevó a cabo desde la sección de instancias del panel de administración, donde fue posible observar la máquina virtual creada junto con sus atributos principales. Entre los elementos visibles se encuentran el nombre de la instancia, la imagen asociada, la dirección *IP*, el *flavor*, el estado operativo y la zona de disponibilidad. Estos datos permiten confirmar que la máquina virtual desplegada corresponde a la configuración definida previamente en el *script*.

En la implementación realizada, la instancia registrada en *Horizon* mostró coincidencia con los parámetros establecidos en la automatización. En particular, se verificó el uso de la imagen *cirros*, el *flavor m1.tiny*, la red *private* y la zona de disponibilidad *nova*. Asimismo, se observó que la instancia se encontraba en estado activo, lo que confirma que el despliegue fue reconocido correctamente por la plataforma.

La figura 4.4 presenta la visualización de la instancia dentro de *Horizon*. Esta evidencia permite comprobar que la automatización no solo fue ejecutada desde la terminal, sino que también produjo un resultado observable y verificable dentro del entorno de administración de *OpenStack*. De esta manera, la validación en *Horizon* funciona como un mecanismo complementario para confirmar la correcta materialización de la infraestructura definida en código.

Project	Host	Name	Image Name	IP Address	Flavor	Status	Task	Power State	Age	Actions
admin	comp ute	vm-terraform-1	cirros	192.168.100.45	m1.tiny	Activa	Ninguno	Corriendo	1 mes, 3 semanas	Rescatar instancia

Figura 4.4: Visualización en *Horizon* de la instancia desplegada mediante *Terraform*.

Capítulo 5

Despliegue automatizado en Microsoft Azure

5.1. Configuración del proveedor *Azure*

Para iniciar el despliegue en *Microsoft Azure*, fue necesario configurar el proveedor *azurerm* dentro de *Terraform*. Este proveedor permite establecer la comunicación entre el *script* y la infraestructura de *Azure*, utilizando una suscripción activa como contexto de operación.

Antes de ejecutar el código, es necesario autenticar el entorno mediante la interfaz de línea de comandos de *Azure*, (*Azure CLI*), la cual permite administrar y gestionar recursos en la nube a través de comandos ejecutados desde la terminal. Esta herramienta facilita tareas como la autenticación, la consulta de recursos y la validación de configuraciones, así como su integración con herramientas de infraestructura como código, como *Terraform*. Para ello se utilizó el comando `az login -use-device-code`, el cual permite iniciar sesión mediante un código de verificación y habilitar el acceso a los recursos asociados a la cuenta.

Asimismo, es necesario contar con el identificador de la suscripción (*subscription_id*), el cual se obtiene desde el portal de *Microsoft Azure*, en la sección de *Resource Manager*, dentro del apartado de suscripciones. Este identificador se utiliza en la configuración del proveedor para asegurar que los recursos se creen dentro del entorno correcto. En la figura 5.1 se observa la suscripción activa utilizada durante las pruebas realizadas en esta tesis. Asimismo, puede identificarse que la suscripción se encontraba en estado *Activo*, condición necesaria para permitir la creación y administración de recursos dentro de *Microsoft Azure*. La información mostrada también incluye el identificador de la suscripción (*subscription_id*), el cual fue empleado durante la configuración del proveedor *azurerm* en *Terraform*.

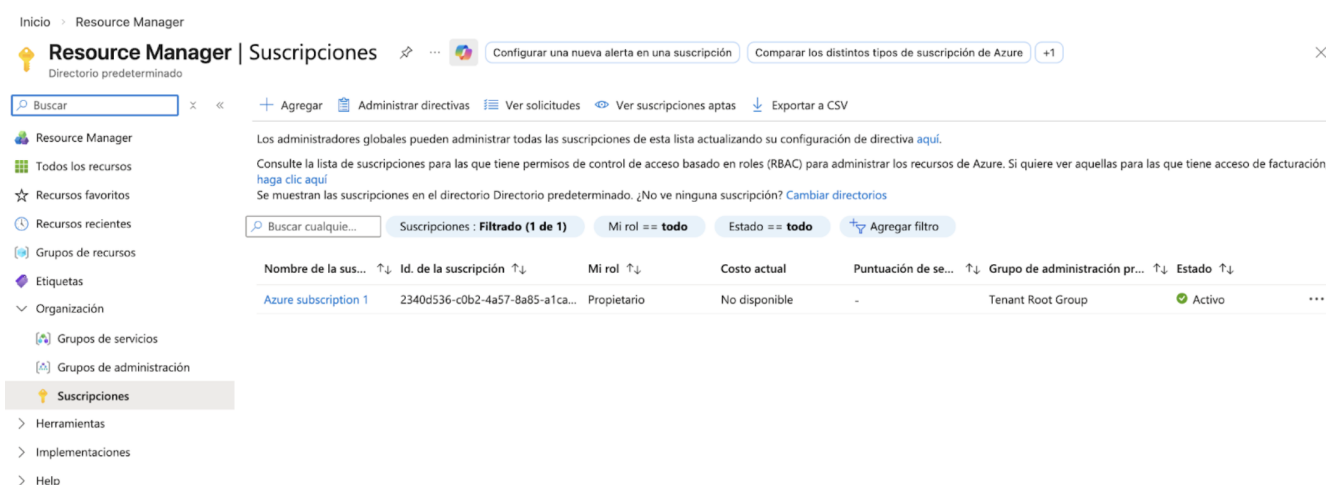


Figura 5.1: Sección de suscripciones en el portal de *Microsoft Azure*.

5.2. Adaptación del código *Terraform*

El código desarrollado para *Microsoft Azure* fue estructurado de manera que permitiera crear, desde cero, la infraestructura básica necesaria para desplegar una máquina virtual *Linux*. Mientras que en *OpenStack* el despliegue automatizado reutiliza recursos previamente creados, en *Microsoft Azure* el *script* define y crea directamente los recursos principales de infraestructura necesarios para la máquina virtual.

En particular, se utilizaron los siguientes recursos:

- `azurerm_resource_group`
- `azurerm_virtual_network`
- `azurerm_subnet`
- `azurerm_network_interface`
- `azurerm_linux_virtual_machine`

Estos recursos permiten definir de manera completa la infraestructura necesaria para el despliegue.

Adicionalmente, se emplearon elementos complementarios como el bloque *provider* `azurerm`, las variables para parametrizar valores del despliegue y el bloque *output*, el cual permite mostrar información relevante de la máquina virtual creada.

Asimismo, el código de *Terraform* se ejecuta dentro de un directorio de trabajo específico, en el cual se generan archivos asociados al estado de la infraestructura, como `terraform.tfstate`. Este archivo permite llevar un control de los recursos creados y evitar duplicidades. Por esta razón, si se desea desplegar una infraestructura adicional independiente, es recomendable utilizar un directorio distinto o modificar los parámetros del despliegue.

El código 5.1 muestra el archivo principal utilizado para el despliegue automatizado en *Microsoft Azure*, incluyendo la definición del proveedor, las variables del despliegue, los recursos de red y la creación de la máquina virtual. El contenido completo de este archivo puede consultarse en el Apéndice B.1.

```

terraform {
  required_version = ">=1.5.0"
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = ">=3.100.0"
    }
  }
}

provider "azurerm" {
  features {}
  subscription_id      = var.subscription_id
  resource_provider_registrations = "none"
}

# ===== VARIABLES =====
variable "num_vms" {
  type    = number
  default = 5
}

variable "nombre_vm" {
  type    = string
  default = "vm-terraform-azure-test"
}

variable "subscription_id" {
  type = string
}

variable "region" {
  type    = string
  default = "southcentralus"
}

variable "rg_name" {
  type    = string
  default = "rg-terraform-azure-test"
}

variable "vm_size" {
  type    = string
  default = "Standard_D2s_v3"
}

variable "admin_user" {
  type    = string
  default = "azureuser"
}

variable "admin_pass" {
  type      = string
  sensitive = true
}

variable "vnet_name" {
  type    = string
  default = "vnet-terraform-test"
}

variable "subnet_name" {
  type    = string
  default = "subnet-terraform-test"
}

variable "vnet_cidr" {
  type    = string
  default = "10.10.0.0/16"
}

variable "subnet_cidr" {
  type    = string
  default = "10.10.1.0/24"
}

# ===== RESOURCE GROUP =====
resource "azurerm_resource_group" "rg" {
  name     = var.rg_name
  location = var.region
}

# ===== NETWORK =====
resource "azurerm_virtual_network" "vnet" {
  name      = var.vnet_name

```

```

location          = azurerm_resource_group.rg.location
resource_group_name = azurerm_resource_group.rg.name
address_space      = [var.vnet_cidr]
}
resource "azurerm_subnet" "subnet" {
  name                = var.subnet_name
  resource_group_name = azurerm_resource_group.rg.name
  virtual_network_name = azurerm_virtual_network.vnet.name
  address_prefixes     = [var.subnet_cidr]
}
# ===== NIC =====
resource "azurerm_network_interface" "nic" {
  count = var.num_vms
  name   = "${var.nombre_vm}-${count.index+1}-nic"
  location          = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  ip_configuration {
    name                = "ipconfig1"
    subnet_id           = azurerm_subnet.subnet.id
    private_ip_address_allocation = "Dynamic"
  }
}
# ===== VM =====
resource "azurerm_linux_virtual_machine" "vm" {
  count = var.num_vms
  name   = "${var.nombre_vm}-${count.index+1}"
  location          = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  size             = var.vm_size
  admin_username    = var.admin_user
  admin_password     = var.admin_pass
  disable_password_authentication = false
  network_interface_ids = [
    azurerm_network_interface.nic[count.index].id
  ]
  os_disk {
    caching              = "ReadWrite"
    storage_account_type = "Standard_LRS"
  }
  source_image_reference {
    publisher = "Canonical"
    offer     = "0001-com-ubuntu-server-jammy"
    sku       = "22_04-lts"
    version   = "latest"
  }
}
# ===== OUTPUT =====
output "vm_ids" {
  value = azurerm_linux_virtual_machine.vm[*].id
}

```

Código 5.1. Archivo principal utilizado para el despliegue automatizado en *Microsoft Azure*.

5.3. Creación automatizada de máquinas virtuales

Una vez configurado el proveedor y definidos los recursos, se puede crear una máquina virtual en *Microsoft Azure*. En esta etapa fue necesario verificar la compatibilidad entre la región de despliegue, el tamaño de la máquina virtual y la imagen del sistema operativo, ya que estos elementos no pueden seleccionarse de manera arbitraria. La región elegida fue *southcentralus*, mientras que el tamaño de máquina virtual seleccionado fue *Standard_B2ps_v2*. Este tamaño corresponde a una arquitectura *ARM64*, por lo que la imagen del sistema operativo debía ser compatible con dicha arquitectura. Para verificar qué tamaños de máquina virtual estaban disponibles en la región seleccionada, se utilizó el comando:

```
az vm list-skus --location southcentralus --size Standard B2 --all -o table
```

Este comando permite identificar los distintos tamaños compatibles en la zona y revisar el campo *Restrictions*, donde debe aparecer el valor *None* para indicar que el recurso puede utilizarse sin restricciones en esa región. Como se muestra en la figura 5.2, el campo *Restrictions* indica *None*, lo que confirma la compatibilidad del recurso con la región.

[root@controller ~]# terraform-azure\az vm list-skus	Location	skuName	Size	Standard_B2ms	all	table
ResourceType	Locations	Name	Zones	Restrictions		
virtualMachines	southcentralus	Standard_B20ms	1,2,3	['NotAvailableForSubscription, type: Location, locations: southcentralus', 'NotAvailableForSubscription, type: Zone, locations: southcentralus, zones: 2,3']		
virtualMachines	southcentralus	Standard_B20ls_v2	1,2,3	['NotAvailableForSubscription, type: Location, locations: southcentralus', 'NotAvailableForSubscription, type: Zone, locations: southcentralus, zones: 2,3']		
virtualMachines	southcentralus	Standard_B20s_v2	1,2,3	['NotAvailableForSubscription, type: Location, locations: southcentralus', 'NotAvailableForSubscription, type: Zone, locations: southcentralus, zones: 2,3']		
virtualMachines	southcentralus	Standard_B20ts_v2	1,2,3	['NotAvailableForSubscription, type: Location, locations: southcentralus', 'NotAvailableForSubscription, type: Zone, locations: southcentralus, zones: 2,3']		
virtualMachines	southcentralus	Standard_B21s_v2	1,2,3	['NotAvailableForSubscription, type: Location, locations: southcentralus', 'NotAvailableForSubscription, type: Zone, locations: southcentralus, zones: 2,3']		
virtualMachines	southcentralus	Standard_B2ms	1,2,3	['NotAvailableForSubscription, type: Location, locations: southcentralus', 'NotAvailableForSubscription, type: Zone, locations: southcentralus, zones: 2,3']		
virtualMachines	southcentralus	Standard_B2pls_v2	1,2,3	None		
virtualMachines	southcentralus	Standard_B2ps_v2	1,2,3	None		
virtualMachines	southcentralus	Standard_B2pts_v2	1,2,3	['NotAvailableForSubscription, type: Location, locations: southcentralus', 'NotAvailableForSubscription, type: Zone, locations: southcentralus, zones: 2,3']		
virtualMachines	southcentralus	Standard_B2s	1,2,3	['NotAvailableForSubscription, type: Location, locations: southcentralus', 'NotAvailableForSubscription, type: Zone, locations: southcentralus, zones: 2,3']		
virtualMachines	southcentralus	Standard_B2s_v2	1,2,3	['NotAvailableForSubscription, type: Location, locations: southcentralus', 'NotAvailableForSubscription, type: Zone, locations: southcentralus, zones: 2,3']		
virtualMachines	southcentralus	Standard_B2ts_v2	1,2,3	['NotAvailableForSubscription, type: Location, locations: southcentralus', 'NotAvailableForSubscription, type: Zone, locations: southcentralus, zones: 2,3']		

Figura 5.2: Consulta de compatibilidad de tamaños de máquina virtual en la región *southcentralus*.

Posteriormente, se verifica la disponibilidad de imágenes compatibles mediante el comando:

```
az vm image list-skus -l southcentralus -p Canonical -f
0001-com-ubuntu-server-jammy -o table
```

Esta consulta permite identificar qué *SKU* (*Stock Keeping Unit*) de imagen de Ubuntu está disponible en la región *southcentralus* y seleccionar una opción compatible con la arquitectura de la máquina virtual. En este contexto, un *SKU* representa una variante específica de una imagen dentro del catálogo de *Azure*, la cual define características como la versión del sistema operativo, el tipo de arquitectura (por ejemplo, *x86_64* o *ARM64*) y el nivel de soporte disponible. Su función es permitir seleccionar una imagen que sea técnicamente compatible con el tamaño de máquina virtual elegido, ya que no todas las combinaciones de tamaño, región e imagen son válidas. A partir de esta verificación, se confirma la disponibilidad de la opción *22_04-lts-arm64*, la cual es necesaria debido al tamaño de máquina virtual previamente seleccionado. La figura 5.3 muestra el resultado de la consulta de *SKU* de imágenes de Ubuntu disponibles en la región *southcentralus*, donde se observa la presencia de la opción compatible con la arquitectura requerida.

```
[root@controller terraform-azure]# az vm image list-skus -l southcentralus -p Canonical -f 0001-com-ubuntu-server-jammy -o table
```

Location	Name
-----	-----
southcentralus	22_04-lts
southcentralus	22_04-lts-arm64
southcentralus	22_04-lts-gen2

Figura 5.3: Consulta de *SKU* de imágenes Ubuntu disponibles en la región *southcentralus*, donde se identifica la opción *22_04-lts-arm64*.

Una vez completadas estas validaciones, se ejecuta el flujo principal de *Terraform* mediante los comandos `terraform init`, `terraform plan` y `terraform apply`. Estos comandos permitieron inicializar el directorio de trabajo, analizar la infraestructura declarada y finalmente aplicar la configuración para crear la máquina virtual en *Azure*. De esta forma, la creación de la instancia quedó completamente integrada dentro del procedimiento automatizado, reduciendo la intervención manual a la preparación previa de la autenticación y la validación de compatibilidades.

Las figuras 5.4, 5.5, 5.6 y 5.7 muestran la ejecución secuencial de los comandos `terraform init -upgrade`, `terraform plan` y `terraform apply`, los cuales conforman el flujo principal del despliegue automatizado en *Microsoft Azure*. Como se observa en la figura 5.7, el mensaje “*Apply complete! Resources: 1 added, 0 changed, 0 destroyed*”, el cual indica que *Terraform* creó exitosamente un recurso nuevo dentro de la infraestructura, sin modificar ni eliminar recursos previamente existentes. En este contexto, el valor *1 added* confirma que la máquina virtual definida en el *script* fue aprovisionada correctamente, mientras que los valores *0 changed* y *0 destroyed* indican que durante la ejecución no fue necesario alterar ni eliminar otros componentes de la configuración declarada. Este resultado permite verificar que el despliegue se realizó de manera controlada y consistente con lo definido en el código.

```
(root@controller terraform-azure)# terraform init -upgrade
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/azurerm versions matching ">= 3.100.0"...
- Using previously-installed hashicorp/azurerm v4.59.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
(root@controller terraform-azure)# terraform plan
azurerm_resource_group: Refreshing state... [id=/subscriptions/2340d536-c0b2-4a57-8a85-a1ca847d8e2a/resourceGroups/rg-terraform]
azurerm_virtual_network_vnet: Refreshing state... [id=/subscriptions/2340d536-c0b2-4a57-8a85-a1ca847d8e2a/resourceGroups/rg-terraform/providers/Microsoft.Network/virtualNetworks/vnet-terraform]
azurerm_subnet_subnet1: Refreshing state... [id=/subscriptions/2340d536-c0b2-4a57-8a85-a1ca847d8e2a/resourceGroups/rg-terraform/providers/Microsoft.Network/virtualNetworks/vnet-terraform/subnets/subnet-terraform]
azurerm_network_interface_nic: Refreshing state... [id=/subscriptions/2340d536-c0b2-4a57-8a85-a1ca847d8e2a/resourceGroups/rg-terraform/providers/Microsoft.Network/networkInterfaces/vm-terraform-azure-1-nic]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

# azurerm_linux_virtual_machine.vm will be created
+ resource "azurerm_linux_virtual_machine" "vm" {
  + admin_password      = (sensitive value)
  + admin_username      = "azureuser"
  + allow_extension_operations = (known after apply)
  + bypass_platform_safety_checks_on_user_schedule_enabled = false
  + computer_name       = (known after apply)
  + disable_password_authentication = false
  + disk_controller_type = (known after apply)
  + extensions_time_budget = "PT1H30M"
  + id                  = (known after apply)
  + location             = "southcentralus"
  + max_bid_price        = -1
  + name                 = "vm-terraform-azure-1"
  + network_interface_ids = [
    + "/subscriptions/2340d536-c0b2-4a57-8a85-a1ca847d8e2a/resourceGroups/rg-terraform/providers/Microsoft.Network/networkInterfaces/vm-terraform-azure-1-nic",
  ]
  + os_managed_disk_id = (known after apply)
  + patch_assessment_mode = (known after apply)
  + patch_mode          = (known after apply)
  + platform_fault_domain = -1
  + priority             = "Regular"
  + private_ip_address   = (known after apply)
  + private_ip_addresses = (known after apply)
  + provision_vm_agent   = (known after apply)
  + public_ip_address     = (known after apply)
  + public_ip_addresses  = (known after apply)
  + resource_group_name   = "rg-terraform"
  + size                 = "Standard_D2ps_v2"
  + virtual_machine_id    = (known after apply)
  + vm_agent_platform_updates_enabled = (known after apply)
  + os_disk {
    + caching           = "ReadWrite"
    + disk_size_gb      = (known after apply)
    + id                = (known after apply)
    + name              = (known after apply)
    + storage_account_type = "Standard_LRS"
    + write_accelerator_enabled = false
  }
}
```

Figura 5.4: Ejecución del comando `terraform init -upgrade` y primera parte del comando `terraform plan`.

```
+ source_image_reference {
  + offer      = "0001-com-ubuntu-server-jammy"
  + publisher  = "Canonical"
  + sku        = "22_04-lts-arm64"
  + version    = "latest"
}
+ termination_notification (known after apply)
}

Plan: 1 to add, 0 to change, 0 to destroy.

Changes to Outputs:
+ vm_id = (known after apply)

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
```

Figura 5.5: Segunda parte de la ejecución del comando `terraform plan`.

En las figuras 5.4 y 5.5 puede observarse que *Terraform* identificó la creación de una máquina virtual llamada *vm-terraform-azure-1* dentro de la región *southcentralus*. Asimismo, se observa el

grupo de recursos *rg-terraform*, el cual fue utilizado para organizar los recursos creados durante el despliegue automatizado. Esto permitió que la máquina virtual y los componentes asociados quedaran agrupados dentro de una misma estructura administrativa en *Microsoft Azure*.

```
[root@controller terraform-azure]# terraform apply
azurerm_resource_group.rg: Refreshing state... [id=/subscriptions/2340d536-c0b2-4a57-8a85-a1ca847d8e2a/resourceGroups/rg-terraform]
azurerm_virtual_network.vnet: Refreshing state... [id=/subscriptions/2340d536-c0b2-4a57-8a85-a1ca847d8e2a/resourceGroups/rg-terraform/providers/Microsoft.Network/virtualNetworks/vnet-terraform]
azurerm_subnet.subnet: Refreshing state... [id=/subscriptions/2340d536-c0b2-4a57-8a85-a1ca847d8e2a/resourceGroups/rg-terraform/providers/Microsoft.Network/virtualNetworks/vnet-terraform/subnets/subnet-terraform]
azurerm_network_interface.nic: Refreshing state... [id=/subscriptions/2340d536-c0b2-4a57-8a85-a1ca847d8e2a/resourceGroups/rg-terraform/providers/Microsoft.Network/networkInterfaces/vm-terraform-azure-1-nic]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

# azurerm_linux_virtual_machine.vm will be created
+ resource "azurerm_linux_virtual_machine" "vm" {
  + admin_password      = (sensitive value)
  + admin_username      = "azureuser"
  + allow_extension_operations = (known after apply)
  + bypass_platform_safety_checks_on_user_schedule_enabled = false
  + computer_name       = (known after apply)
  + disable_password_authentication = false
  + disk_controller_type = (known after apply)
  + extensions_time_budget = "PT1H30M"
  + id                  = (known after apply)
  + location             = "southcentralus"
  + max_bid_price        = -1
  + name                 = "vm-terraform-azure-1"
  + network_interface_ids = [
    + "/subscriptions/2340d536-c0b2-4a57-8a85-a1ca847d8e2a/resourceGroups/rg-terraform/providers/Microsoft.Network/networkInterfaces/vm-terraform-azure-1-nic",
  ]
  + os_managed_disk_id = (known after apply)
  + patch_assessment_mode = (known after apply)
  + patch_mode          = (known after apply)
  + platform_fault_domain = -1
  + priority             = "Regular"
  + private_ip_addresses = (known after apply)
  + private_ip_addresses = (known after apply)
  + provision_vm_agent   = (known after apply)
  + public_ip_addresses  = (known after apply)
  + public_ip_addresses  = (known after apply)
  + resource_group_name  = "rg-terraform"
  + size                 = "Standard_B2ps_v2"
  + virtual_machine_id   = (known after apply)
  + vm_agent_platform_updates_enabled = (known after apply)

  + os_disk {
    + caching           = "ReadWrite"
    + disk_size_gb      = (known after apply)
    + id                = (known after apply)
    + name              = (known after apply)
    + storage_account_type = "Standard_LRS"
    + write_accelerator_enabled = false
  }

  + source_image_reference {
    + offer = "0001-com-ubuntu-server-jammy"
    + publisher = "Canonical"
    + sku = "22_04-lts-arm64"
    + version = "latest"
  }

  + termination_notification (known after apply)
}

Plan: 1 to add, 0 to change, 0 to destroy.

Changes to Outputs:
- vm_id = (known after apply)
```

Figura 5.6: Primera parte de la ejecución del comando `terraform apply`.

```
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

azurerm_linux_virtual_machine.vm: Creating...
azurerm_linux_virtual_machine.vm: Still creating... [10s elapsed]
azurerm_linux_virtual_machine.vm: Still creating... [20s elapsed]
azurerm_linux_virtual_machine.vm: Still creating... [30s elapsed]
azurerm_linux_virtual_machine.vm: Still creating... [40s elapsed]
azurerm_linux_virtual_machine.vm: Creation complete after 49s [id=/subscriptions/2340d536-c0b2-4a57-8a85-a1ca847d8e2a/resourceGroups/rg-terraform/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-1]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

Outputs:
vm_id = "/subscriptions/2340d536-c0b2-4a57-8a85-a1ca847d8e2a/resourceGroups/rg-terraform/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-1"
```

Figura 5.7: Segunda parte de la ejecución del comando `terraform apply`.

5.4. Configuración de red y acceso

En *Microsoft Azure*, la configuración de red se realiza mediante la definición explícita de una red virtual, una subred y una interfaz de red dentro del propio *script* de *Terraform*. A diferencia de *OpenStack*, donde estos elementos pueden estar previamente disponibles para su reutilización, en *Azure* se crean dinámicamente como parte del despliegue automatizado.

En la implementación realizada, la red virtual fue definida mediante un bloque *CIDR* (*Classless Inter-Domain Routing*), el cual es una notación utilizada para representar el rango de direcciones *IP*

que puede abarcar una red. Esta notación combina una dirección base con un prefijo numérico que indica cuántos bits están reservados para identificar la red, lo que permite determinar su tamaño y capacidad de direccionamiento. A partir de esta red general, la subred delimitó el segmento específico en el que se desplegó la máquina virtual. Posteriormente, se creó una interfaz de red asociada a dicha subred, la cual permitió conectar la *VM* a la infraestructura de red declarada en el código. Esta organización es coherente con el modelo de *Azure*, donde la conectividad de la *VM* depende de la correcta definición de estos recursos dentro del grupo de recursos correspondiente.

Este enfoque declarativo hace que la configuración de red sea reproducible y consistente, evitando depender de configuraciones manuales dentro del portal. Asimismo, facilita la integración entre recursos, ya que la máquina virtual puede asociarse directamente a la interfaz de red y esta, a su vez, a la subred previamente definida en la misma ejecución del *script*.

5.5. Validación del despliegue

Una vez ejecutado el despliegue automatizado, la validación del resultado se realizó a través del portal de *Microsoft Azure*. Para ello, se accedió a la sección de máquinas virtuales, donde fue posible observar la instancia creada y verificar su registro dentro del entorno. En la figura 5.8 puede observarse la máquina virtual *vm-terraform-azure-1* creada mediante *Terraform*. La instancia se encuentra asociada al grupo de recursos *rg-terraform*, ubicada en la región *South Central US* y en estado *En ejecución*. Además, utiliza el sistema operativo *Linux* y el tamaño de máquina virtual *Standard_B2ps_v2*, parámetros que coinciden con la configuración definida previamente en el proceso de automatización.



Figura 5.8: Máquina virtual desplegada en *Microsoft Azure*.

La validación realizada en este trabajo consistió en comprobar que la máquina virtual apareciera correctamente dentro del portal de *Azure* y que los parámetros mostrados coincidieran con los definidos en el *script*. Entre estos elementos se verificaron el nombre de la instancia, la región de despliegue, el tamaño de la máquina virtual y el sistema operativo asociado a la imagen seleccionada. Esta comprobación permitió confirmar que la configuración declarativa fue materializada correctamente por la plataforma. Como se observa en la figura 5.8, la instancia se encuentra registrada dentro del portal de administración, lo que confirma que el despliegue automatizado fue ejecutado de forma exitosa. En este sentido, la validación en *Azure* no se limitó únicamente a la ejecución de comandos en terminal, sino que también incluyó la verificación visual del recurso ya desplegado dentro de la plataforma.

En conjunto, este capítulo demuestra que la solución desarrollada pudo adaptarse tanto a un entorno de nube privada como a una nube pública mediante el uso de *Terraform*. Las pruebas realizadas permitieron comprobar que la estructura general de la automatización, el uso de variables y la organización modular de la infraestructura pudieron mantenerse entre *OpenStack* y *Microsoft Azure*. Sin embargo, la portabilidad no fue completamente transparente a nivel de código, ya que cada proveedor utiliza recursos, servicios y parámetros específicos que requieren ajustes durante la implementación. Por ello, aunque fue posible conservar la lógica general de aprovisionamiento, fue necesario adaptar la definición de los recursos para cada entorno. Estos resultados permitieron comprobar la viabilidad de la propuesta en escenarios *multicloud*.

Capítulo 6

Análisis y resultados

6.1. Pruebas de rendimiento y escalabilidad

Con el objetivo de evaluar el funcionamiento del sistema automatizado desarrollado mediante *Terraform*, se realizaron diversas pruebas de rendimiento y escalabilidad en entornos de *OpenStack* y *Microsoft Azure*. Estas pruebas incluyeron operaciones de creación simultánea de máquinas virtuales, destrucción de infraestructura, actualización de recursos, validación del archivo de estado `terraform.tfstate` y recuperación de configuraciones previas mediante mecanismos de *rollback*. Asimismo, se efectuaron pruebas de saturación y consistencia para observar la estabilidad del entorno durante ejecuciones repetidas y bajo distintos niveles de carga.

Durante cada escenario se analizaron aspectos relacionados con tiempos de ejecución, automatización de tareas, comportamiento del archivo de estado y respuesta del entorno ante múltiples operaciones concurrentes. De igual manera, se observó la forma en que cada plataforma administra recursos de cómputo, red y almacenamiento cuando las operaciones son ejecutadas desde *Terraform*.

6.1.1. Prueba de paralelismo

La prueba de paralelismo tuvo como objetivo analizar el comportamiento del despliegue automatizado al ejecutar múltiples operaciones simultáneamente mediante *Terraform*. Para ello, se modificó el parámetro `-parallelism`, el cual controla la cantidad máxima de tareas que pueden ejecutarse al mismo tiempo durante la creación o administración de recursos.

Esta prueba permitió evaluar el impacto del paralelismo sobre los tiempos de despliegue y observar el comportamiento del entorno al incrementar la cantidad de operaciones concurrentes.

Antes de realizar las pruebas de escalabilidad, fue necesario aumentar la cantidad de memoria *RAM* asignada a las máquinas virtuales utilizadas como nodo *controller* y nodo *compute*. Inicialmente, ambas máquinas virtuales contaban con aproximadamente 8 *GB* de memoria asignada.

Para verificar las características iniciales del entorno se utilizó el comando `virsh dominfo <vm_name>`, mientras que el comando `free -h` permite observar la memoria disponible dentro del servidor físico.

Mediante el comando `virsh edit <vm_name>`, se modificaron los parámetros `<memory>` y `<currentMemory>` dentro de la configuración XML de cada máquina virtual. Al nodo *controller* se le asignó 26 *GB* de memoria *RAM*, mientras que al nodo *compute* se le asignó 126 *GB*.

La figura 6.1 muestra las características iniciales del entorno antes de incrementar la memoria asignada a las máquinas virtuales, se puede observar en esta figura que tanto el nodo *DJcontroller*

como el nodo *DJcompute* tenían asignados aproximadamente 8 GB de memoria RAM (8388608 KiB). Asimismo, ambos nodos disponían de 6 *vCPU*, configuración que resultó insuficiente para las pruebas de escalabilidad realizadas posteriormente.

```
root@servidor:~# virsh dominfo DJcontroller
Id: -
Name: DJcontroller
UUID: 9718cc80-023e-4239-a50e-22f6fd4d6f48
OS Type: hvm
State: shut off
CPU(s): 6
Max memory: 8388608 KiB
Used memory: 8388608 KiB
Persistent: yes
Autostart: disable
Managed save: no
Security model: apparmor
Security DOI: 0

root@servidor:~# virsh dominfo DJcompute
Id: -
Name: DJcompute
UUID: 86ee96b6-6a61-4b09-81af-b1339911d2d0
OS Type: hvm
State: shut off
CPU(s): 6
Max memory: 8388608 KiB
Used memory: 8388608 KiB
Persistent: yes
Autostart: disable
Managed save: no
Security model: apparmor
Security DOI: 0

root@servidor:~# free -h
              total        used        free      shared  buff/cache   available
Mem:          251Gi        951Mi        250Gi         2.0Mi         451Mi        249Gi
Swap:          14Gi           0B         14Gi

root@servidor:~# virsh edit DJcontroller
Domain 'DJcontroller' XML configuration edited.

root@servidor:~# virsh edit DJcontroller
Domain 'DJcontroller' XML configuration not changed.

root@servidor:~# virsh edit DJcompute
Domain 'DJcompute' XML configuration edited.
```

Figura 6.1: Configuración inicial de memoria en los nodos *controller* y *compute*.

La figura 6.2 muestra el incremento de memoria aplicado al nuevo entorno después de modificar la configuración y reiniciar las máquinas virtuales. Ahora se puede observar el incremento de memoria aplicado a ambos nodos. El nodo *DJcontroller* tiene aproximadamente 32 GB de memoria (33554432 KiB), mientras que el nodo *DJcompute* fue configurado con aproximadamente 192 GB (201326592 KiB). Este aumento permitió ejecutar pruebas con una mayor cantidad de máquinas virtuales simultáneas.

```

[root@servidor:~# virsh dominfo DJcontroller
Id: -
Name: DJcontroller
UUID: 9718cc80-023e-4239-a50e-22f6fd4d6f48
OS Type: hvm
State: shut off
CPU(s): 6
Max memory: 33554432 KiB
Used memory: 33554432 KiB
Persistent: yes
Autostart: disable
Managed save: no
Security model: apparmor
Security DOI: 0

[root@servidor:~# virsh dominfo DJcompute
Id: -
Name: DJcompute
UUID: 86ee96b6-6a61-4b09-81af-b1339911d2d0
OS Type: hvm
State: shut off
CPU(s): 6
Max memory: 201326592 KiB
Used memory: 201326592 KiB
Persistent: yes
Autostart: disable
Managed save: no
Security model: apparmor
Security DOI: 0

[root@servidor:~# free -h

```

	total	used	free	shared	buff/cache	available
Mem:	251Gi	958Mi	250Gi	2.0Mi	452Mi	249Gi
Swap:	14Gi	0B	14Gi			

Figura 6.2: Incremento de memoria *RAM* en los nodos *controller* y *compute*.

Posteriormente, se cargaron las credenciales administrativas de *OpenStack* y se verificaron las estadísticas generales del hipervisor utilizando el comando `openstack hypervisor stats show`. Esto permite comprobar la cantidad de memoria, almacenamiento y *vCPU* disponibles dentro del entorno. La figura 6.3 muestra las estadísticas generales del hipervisor utilizadas durante las pruebas. Puede observarse que el entorno disponía de 385438 MB de memoria total y 384414 MB libres al momento de la consulta. Asimismo, se reportan 12 *vCPU* disponibles y ninguna máquina virtual en ejecución, lo que permitió utilizar la mayor parte de los recursos durante las pruebas de paralelismo.

```
[root@controller ~]# source /root/admin-openrc.sh
[root@controller ~]# openstack hypervisor stats show
This command is deprecated.
```

Field	Value
count	2
current_workload	0
disk_available_least	50
free_disk_gb	86
free_ram_mb	384414
local_gb	86
local_gb_used	0
memory_mb	385438
memory_mb_used	1024
running_vms	0
vcpus	12
vcpus_used	0

Figura 6.3: Estadísticas del hipervisor en *OpenStack*.

Además de la memoria disponible, también fue necesario considerar la capacidad de almacenamiento del volumen físico `vg0`, ya que el tamaño de disco asignado a cada máquina virtual influía directamente en la cantidad máxima de instancias que podían desplegarse. Con el propósito de reducir el consumo de recursos durante las pruebas de escalabilidad, se creó un nuevo *flavor* denominado `m1.nanotiny`, configurado con recursos mínimos. La creación del nuevo *flavor* se realizó mediante el siguiente comando:

```
openstack flavor create \
--ram 256 \
--disk 1 \
--vcpus 1 \
m1.nanotiny
```

La figura 6.4 muestra la creación y configuración del *flavor* `m1.nanotiny`. La configuración asigna 1 *vCPU*, 256 MB de memoria *RAM* y 1 GB de almacenamiento, características que permitieron reducir el consumo de recursos durante las pruebas de escalabilidad.

```
[root@controller terraform-openstack]# openstack flavor create \
--ram 256 \
--disk 1 \
--vcpus 1 \
m1.nanotiny
```

Field	Value
OS-FLV-DISABLED:disabled	False
OS-FLV-EXT-DATA:ephemeral	0
description	None
disk	1
id	9719b4ea-deb2-47c8-8d22-007846739ef0
name	m1.nanotiny
os-flavor-access:is_public	True
properties	
ram	256
rxtx_factor	1.0
swap	0
vcpus	1

Figura 6.4: Creación del *flavor* `m1.nanotiny` en *OpenStack*.

Una vez realizados estos ajustes, se continuó con las pruebas de paralelismo utilizando distintos niveles de concurrencia en *Terraform*.

Durante las primeras pruebas en *OpenStack* también se detectaron limitaciones relacionadas con el manejo de conexiones en *MariaDB*, sistema gestor de bases de datos utilizado por distintos servicios internos de la plataforma. Al aumentar la cantidad de operaciones simultáneas, algunos despliegues comenzaron a presentar errores debido al límite máximo de conexiones disponibles. Inicialmente, el entorno utilizaba una configuración cercana a 151 conexiones máximas, valor común en entornos pequeños o de laboratorio. Sin embargo, este límite resultó insuficiente para ejecutar múltiples despliegues concurrentes. Con el propósito de mejorar la estabilidad del entorno, se modificó el archivo:

`/etc/my.cnf.d/mariadb-server.cnf`

Dentro del archivo `mariadb-server.cnf`, se realizó la modificación dentro de la sección `[mysqld]`. La configuración aplicada fue la siguiente:

```
[mysqld]
max_connections = 2000
```

El incremento a 2000 conexiones permitió soportar una mayor cantidad de operaciones simultáneas durante las pruebas de escalabilidad.

La figura 6.5 muestra la configuración inicial del entorno antes de realizar los ajustes en *MariaDB*. Se puede observar en la figura que el parámetro `max_connections` tenía un valor de 151 conexiones. Asimismo, el campo `Threads_connected` muestra 152 conexiones activas al momento de la consulta, situación que evidenciaba la necesidad de incrementar la capacidad disponible para las pruebas de paralelismo. Posteriormente, se aplicó la nueva configuración incrementando el límite máximo de conexiones.

```
[MariaDB [(none)]]>
MariaDB [(none)]]> SHOW VARIABLES LIKE 'max_connections';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| max_connections | 151 |
+-----+-----+
1 row in set (0.001 sec)

[MariaDB [(none)]]> SHOW STATUS LIKE 'Threads_connected';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Threads_connected | 152 |
+-----+-----+
1 row in set (0.001 sec)
```

Figura 6.5: Configuración inicial de conexiones en *MariaDB*.

La figura 6.6 muestra el resultado obtenido después de modificar el archivo de configuración y reiniciar el servicio. Se puede observar en la figura que el parámetro `max_connections` fue incrementado a 2000 conexiones. Además, el valor de `Threads_connected` muestra 29 conexiones activas durante la verificación, lo que confirma que la nueva configuración fue aplicada correctamente.

```
[root@controller terraform-openstack]# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 23
Server version: 10.5.29-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SHOW VARIABLES LIKE 'max_connections';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| max_connections | 2000 |
+-----+-----+
1 row in set (0.001 sec)

MariaDB [(none)]> SHOW STATUS LIKE 'Threads_connected';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Threads_connected | 29 |
+-----+-----+
1 row in set (0.001 sec)
```

Figura 6.6: Configuración de *MariaDB* después de incrementar el parámetro `max_connections`.

Estos ajustes permitieron mejorar la estabilidad del entorno durante las pruebas de paralelismo y reducir problemas relacionados con saturación de conexiones. De manera similar, durante las pruebas realizadas en *Azure* también se identificaron limitaciones relacionadas con la capacidad disponible de la suscripción utilizada. Al intentar desplegar múltiples máquinas virtuales simultáneamente, la plataforma comenzó a rechazar nuevas instancias debido al límite máximo de *vCPU* permitido dentro de la región `southcentralus`. Las máquinas virtuales utilizadas correspondían al tipo `Standard_D2s_v3`, donde cada instancia consume 2 *vCPU*. Debido a que la suscripción únicamente disponía de 10 *vCPU* regionales, solo fue posible desplegar simultáneamente un máximo de 5 máquinas virtuales. Cuando se intentó desplegar 10 máquinas virtuales de manera concurrente, *Azure* devolvió errores de tipo 409 `Conflict`, indicando que la cuota regional había sido excedida.

6.1.1.1. Prueba de paralelismo en *OpenStack*

El número total de máquinas virtuales se definió dentro del archivo `main.tf` mediante el parámetro `count`, mientras que `-parallelism` determinó cuántas de esas operaciones se ejecutaban al mismo tiempo. Los comandos utilizados para la prueba fueron los siguientes:

```
terraform fmt
terraform init
terraform plan
time terraform apply -auto-approve -parallelism=1
```

Con el propósito de evaluar la consistencia de los resultados obtenidos, cada escenario fue ejecutado en 20 ocasiones bajo las mismas condiciones de operación. A partir de estas repeticiones se calcularon la media aritmética, la desviación estándar muestral, la variación típica y el coeficiente de variación (*CV*). Los resultados se presentan en la Tabla 6.1.

Escenario	Repeticiones	Media (s)	Desv. estándar	Variación típica	CV (%)	Interpretación
OpenStack P1/1VM	20	34.781	0.812	$\pm 0,812$	2.33	Excelente consistencia
OpenStack P1/5VM	20	161.935	3.102	$\pm 3,102$	1.92	Excelente consistencia
OpenStack P5/5VM	20	59.539	8.195	$\pm 8,195$	13.76	Consistencia aceptable
OpenStack P1/10VM	20	324.114	5.382	$\pm 5,382$	1.66	Excelente consistencia
OpenStack P5/10VM	20	114.558	9.855	$\pm 9,855$	8.60	Buena consistencia
OpenStack P10/10VM	20	100.748	17.583	$\pm 17,583$	17.45	Consistencia aceptable
OpenStack P1/15VM	20	496.950	11.318	$\pm 11,318$	2.28	Excelente consistencia
OpenStack P5/15VM	20	169.044	22.490	$\pm 22,490$	13.30	Consistencia aceptable
OpenStack P15/15VM	20	183.143	33.412	$\pm 33,412$	18.24	Consistencia aceptable
OpenStack P1/30VM	20	276.963	1.294	$\pm 1,294$	0.47	Excelente consistencia
OpenStack P5/30VM	20	100.362	0.625	$\pm 0,625$	0.62	Excelente consistencia
OpenStack P10/30VM	20	85.349	0.556	$\pm 0,556$	0.65	Excelente consistencia
OpenStack P15/30VM	20	47.434	0.495	$\pm 0,495$	1.04	Excelente consistencia
OpenStack P30/30VM	20	56.402	0.494	$\pm 0,494$	0.88	Excelente consistencia

Tabla 6.1: Resumen estadístico de las pruebas de paralelismo en *OpenStack*.

A partir de los resultados mostrados en la Tabla 6.1, se observa que la mayoría de los escenarios presentaron coeficientes de variación inferiores al 10 %, indicando una buena o excelente consistencia entre repeticiones. Algunos escenarios con mayores niveles de paralelismo registraron coeficientes entre 10 % y 20 %, clasificados como consistencia aceptable. En conjunto, ninguno de los escenarios superó el umbral de 20 %, por lo que los resultados pueden considerarse estadísticamente consistentes.

Las 20 repeticiones realizadas para cada escenario permitieron comprobar que los tiempos obtenidos fueron reproducibles. Asimismo, las diferencias observadas estuvieron asociadas principalmente al nivel de paralelismo utilizado y a la carga interna del entorno durante cada ejecución.

El comando `time` permitió medir el tiempo total de ejecución del despliegue. En esta prueba, el valor más importante fue `real`, ya que representa el tiempo total percibido por el usuario durante la creación de las máquinas virtuales.

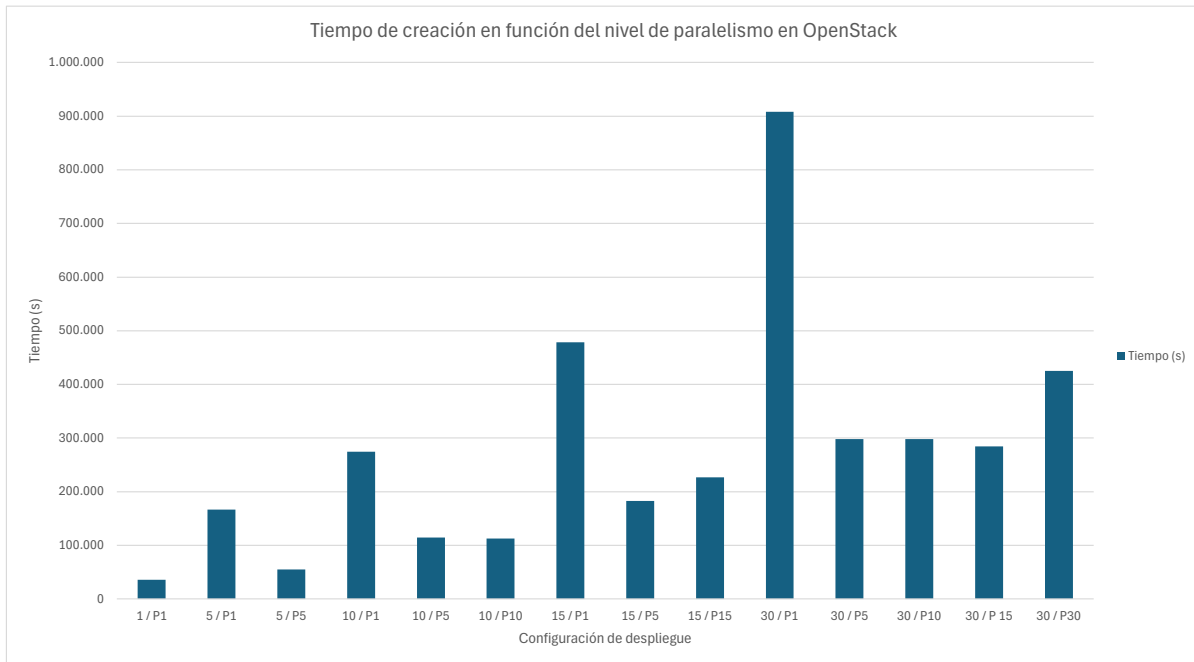


Figura 6.7: Tiempo de despliegue en función del nivel de paralelismo en *OpenStack*.

La figura 6.7 muestra los tiempos obtenidos durante las pruebas de paralelismo en *OpenStack*. En el eje horizontal se representan los distintos escenarios evaluados, donde P1, P5, P10, P15 y P30 corresponden a los niveles de paralelismo utilizados.

Los resultados muestran que el incremento del paralelismo redujo considerablemente los tiempos de despliegue. Para 5 máquinas virtuales, el tiempo promedio disminuyó de aproximadamente 162 s con P1 a 60 s con P5. En los escenarios de 10 máquinas virtuales, los tiempos se redujeron de aproximadamente 324 s con P1 a cerca de 115 s y 101 s utilizando P5 y P10, respectivamente. Para 15 máquinas virtuales, el mejor resultado se obtuvo con P5, con un tiempo cercano a 169 s, mientras que P15 registró aproximadamente 183 s. Para 30 máquinas virtuales, el tiempo disminuyó de aproximadamente 277 s con P1 a cerca de 48 s utilizando P15. Sin embargo, al incrementar el paralelismo hasta P30, el tiempo aumentó ligeramente a aproximadamente 56 s.

Este comportamiento indica que incrementar el paralelismo mejora el rendimiento únicamente hasta cierto punto. A partir de niveles elevados de concurrencia, la sobrecarga generada sobre los servicios internos de *OpenStack*, como *Nova Scheduler*, *MariaDB* y el hipervisor, reduce los beneficios obtenidos y puede incrementar los tiempos de ejecución.

Durante las pruebas con 30 máquinas virtuales únicamente fue posible crear correctamente 24 instancias, mientras que las 6 restantes quedaron en estado **ERROR**.

Aunque los resultados estadísticos mostraron una alta consistencia entre repeticiones, durante los escenarios de mayor carga se identificaron limitaciones operativas del entorno. En particular, las pruebas correspondientes a 30 máquinas virtuales no lograron completar la creación de todas las instancias solicitadas, observándose que únicamente 24 fueron aprovisionadas correctamente. Debido a ello, fue necesario analizar con mayor detalle el estado de las máquinas virtuales y los errores generados por la plataforma para identificar la causa del problema.

Para analizar el problema, primero se verificó el estado general de las instancias utilizando el comando `openstack server list --all --long`. Esto permitió identificar cuáles máquinas virtuales fueron creadas correctamente y cuáles presentaron errores durante el despliegue. La figura 6.8 muestra el estado de las instancias después de la ejecución de las pruebas de paralelismo. En la figura se puede observar que varias instancias aparecen con estado **ERROR**. Asimismo, en la columna correspondiente al estado de energía se muestra el valor **NOSTATE**, indicando que las máquinas virtuales no lograron completar correctamente el proceso de creación dentro del entorno *OpenStack*.

```
[root@controller terraform-openstack]# openstack server list --all --long
```

ID	Availability Zone	Pinned Availability Zone	Name	Status	Task State	Power State	Networks	Image Name	Image ID	Flavor
			Host	Properties	Host Status					
bdc83eb6-fa36-409a-a156-2eff7fea652e	nova		vm-terraform-13	ERROR	None	NOSTATE		cirros	9681ce2d-b638-43b5-ae1a-4df8d2203df1	m1.microtiny
d07ede53-c08d-4cd2-87e4-1f51e91e12c2	nova		vm-terraform-22	ERROR	None	NOSTATE		cirros	9681ce2d-b638-43b5-ae1a-4df8d2203df1	m1.microtiny
2ed5e2a6-af0c-46ea-974d-b4d49a01b9d1	nova		vm-terraform-6	ERROR	None	NOSTATE		cirros	9681ce2d-b638-43b5-ae1a-4df8d2203df1	m1.microtiny
3d0681f0-4338-47f8-b339-9c00db4e200b	nova		vm-terraform-15	ERROR	None	NOSTATE		cirros	9681ce2d-b638-43b5-ae1a-4df8d2203df1	m1.microtiny
ff2a1838-35fb-417b-9a72-d808353be858	nova		vm-terraform-19	ERROR	None	NOSTATE		cirros	9681ce2d-b638-43b5-ae1a-4df8d2203df1	m1.microtiny
01588061-f650-4067-a4e3-a0b0b3e30b35	nova		vm-terraform-23	ERROR	None	NOSTATE		cirros	9681ce2d-b638-43b5-ae1a-4df8d2203df1	m1.microtiny

Figura 6.8: Estado de las máquinas virtuales después de las pruebas de paralelismo en *OpenStack*.

Posteriormente, se consultó el detalle del error utilizando el comando `openstack server show <VM_ID>-c fault`. El resultado indicó errores de tipo **NoValidHost**, señalando que *OpenStack* no logró encontrar un host válido disponible durante la creación de algunas instancias. La figura 6.9 se observa el mensaje de error **NoValidHost**, generado por el servicio *Nova Scheduler*. Este mensaje indica que la plataforma no encontró un host disponible que cumpliera con los requisitos necesarios

para alojar las instancias solicitadas durante la ejecución de las pruebas.

```
| fault | ('code': 500, 'created': '2026-05-28T05:45:55Z', 'message': 'No se ha encontrado ningún host válido.', 'details': 'Traceback (most recent call last):\n File "/usr/lib/python3.9/site-packages/nova/conductor/manager.py", line 1654, in schedule_and_build_instances\n host_lists = self._schedule_instances(context, request_specs[0])\n File "/usr/lib/python3.9/site-packages/nova/conductor/manager.py", line 942, in _schedule_instances\n host_lists = self._query_client.select_destinations(context, spec_obj)\n File "/usr/lib/python3.9/site-packages/nova/scheduler/client/query.py", line 41, in select_destinations\n return self.scheduler_rpcapi.select_destinations(context, spec_obj)\n File "/usr/lib/python3.9/site-packages/nova/scheduler/rpcapi.py", line 160, in select_destinations\n return cctx.call(ctxt, 'select_destinations', **msg_args)\n File "/usr/lib/python3.9/site-packages/oslo_messaging/rpc/client.py", line 190, in call\n result = self.transport._send(context, ctxt, 'select_destinations', **msg_args)\n File "/usr/lib/python3.9/site-packages/oslo_messaging/transport.py", line 123, in _send\n return self._driver.send(target, ctxt, message)\n File "/usr/lib/python3.9/site-packages/oslo_messaging/drivers/amqpdriver.py", line 799, in send\n return self._send(target, ctxt, message, wait_for_reply, timeout)\n File "/usr/lib/python3.9/site-packages/oslo_messaging/drivers/amqpdriver.py", line 791, in _send\n raise result\nnova.exception.RemoteNoValidHost: Remote: No se ha encontrado ningún host válido.\n\nTraceback (most recent call last):\n File "/usr/lib/python3.9/site-packages/oslo_messaging/rpc/server.py", line 252, in inner\n return func(sargs, **kwargs)\n File "/usr/lib/python3.9/site-packages/nova/scheduler/manager.py", line 243, in select_destinations\n raise exception.NoValidHost(reason="")\n\nnova.exception.NoValidHost: No se ha encontrado ningún host válido.\n\n') |
```

Figura 6.9: Error NoValidHost durante las pruebas de paralelismo en *OpenStack*.

Después, se revisaron las cuotas y estadísticas generales del hipervisor utilizando los comandos `openstack quota show admin` y `openstack hypervisor stats show`. Las cuotas disponibles indicaban un límite de 80 instancias, 80 *cores* y 81920 *MB* de memoria *RAM*. Asimismo, las estadísticas mostraban los siguientes valores:

```
running_vms = 72
vcpus_used = 72
memory_mb_used = 38400
local_gb_used = 72
```

La figura 6.10 muestra las estadísticas del hipervisor y las cuotas observadas durante las pruebas. En la figura se puede observar las cuotas asignadas al proyecto así como las estadísticas generales del hipervisor. Puede apreciarse que el límite configurado era de 80 instancias y 80 *vCPU*. Asimismo, las estadísticas muestran 72 máquinas virtuales en ejecución y 72 *vCPU* utilizados al momento de la consulta, valores que permitieron identificar una inconsistencia en el registro de recursos dentro del entorno.

```
[root@controller terraform-openstack]# openstack quota show admin
+-----+
| Resource | Limit |
+-----+
| cores    | 80    |
| instances | 80    |
| ram      | 81920 |
| fixed_ips | None  |
| networks | 100   |
| volumes  | 10    |
| snapshots | 10    |
| gigabytes | 1000  |
| backups  | 10    |
| volumes__DEFAULT__ | -1 |
| gigabytes__DEFAULT__ | -1 |
| snapshots__DEFAULT__ | -1 |
| groups   | 10    |
| ports    | 500   |
| rbac_policies | 10    |
| routers  | 10    |
| subnets | 100   |
| subnet_pools | -1    |
| injected-file-size | 10240 |
| injected-path-size | 256   |
| injected-files | 5     |
| key-pairs | 100   |
| properties | 128   |
| server-group-members | 10    |
| server-groups | 10    |
| floating-ips | 50    |
| secgroup-rules | 100   |
| secgroups | 10    |
| backup-gigabytes | 1000  |
| per-volume-gigabytes | -1    |
+-----+

[root@controller terraform-openstack]# openstack hypervisor stats show
This command is deprecated.
+-----+
| Field | Value |
+-----+
| count | 3      |
| current_workload | 0      |
| disk_available_least | 3      |
| free_disk_gb | 57     |
| free_ram_mb | 539757 |
| local_gb | 129    |
| local_gb_used | 72     |
| memory_mb | 578157 |
| memory_mb_used | 38400  |
| running_vms | 72     |
| vcpus | 18     |
| vcpus_used | 72     |
+-----+
```

Figura 6.10: Cuotas y estadísticas del hipervisor durante las pruebas de paralelismo en *OpenStack*.

Aunque durante la prueba se solicitó el despliegue de 30 máquinas virtuales, las estadísticas del hipervisor reportaban 72 instancias en ejecución. Esta diferencia permitió identificar una inconsistencia relacionada con el registro de los recursos dentro de la plataforma. Esto ocurrió debido a una inconsistencia relacionada con el registro duplicado del hipervisor dentro de los servicios de cómputo, provocando que algunas estadísticas fueran contabilizadas múltiples veces. En conjunto, las pruebas permitieron comprobar que el incremento del paralelismo mejora considerablemente los tiempos de despliegue en *OpenStack*. Sin embargo, bajo cargas elevadas también pueden presentarse errores relacionados con la administración interna de recursos dentro de la plataforma.

6.1.1.2. Prueba de paralelismo en *Azure Portal*

Las pruebas de paralelismo en *Azure* tuvieron como objetivo evaluar el comportamiento de *Terraform* al desplegar múltiples máquinas virtuales utilizando distintos niveles de paralelismo. Para ello, se modificó tanto la cantidad de máquinas virtuales como el parámetro `-parallelism`, permitiendo comparar despliegues secuenciales y concurrentes. En todas las pruebas se utilizaron los siguientes comandos base:

```
terraform fmt
terraform init
terraform plan -var="num_vms=5"

time terraform apply \
-var="num_vms=5" \
-auto-approve \
-parallelism=1
```

El valor definido en `num_vms` y el parámetro `-parallelism` fueron modificados dependiendo del escenario evaluado.

Con el propósito de evaluar la consistencia de los resultados obtenidos, cada escenario fue ejecutado en 20 ocasiones bajo las mismas condiciones de operación. A partir de estas repeticiones se calcularon la media aritmética, la desviación estándar muestral, la variación típica y el coeficiente de variación (*CV*). Los resultados se presentan en la tabla 6.2.

Escenario	Repeticiones	Media (s)	Desv. estándar	Variación típica	CV (%)	Interpretación
Azure P1/1VM	20	49.284	1.121	±1,121	2.27	Excelente consistencia
Azure P1/5VM	20	223.941	5.412	±5,412	2.42	Excelente consistencia
Azure P5/5VM	20	165.208	3.587	±3,587	2.17	Excelente consistencia
Azure P1/10VM	20	257.104	6.982	±6,982	2.72	Excelente consistencia

Tabla 6.2: Resumen estadístico de las pruebas de paralelismo en *Azure*.

Los resultados muestran coeficientes de variación inferiores al 5% en todos los escenarios evaluados, lo que indica una excelente consistencia entre repeticiones. Las 20 ejecuciones realizadas para cada caso permitieron comprobar que los tiempos obtenidos fueron estables y reproducibles. Las pequeñas variaciones observadas pueden atribuirse a la asignación dinámica de recursos y a las operaciones internas de aprovisionamiento de *Azure*, aunque su impacto sobre los tiempos de ejecución fue mínimo.

El comando `time` permitió medir el tiempo total de ejecución del despliegue. En esta prueba, el valor más importante fue `real`, ya que representa el tiempo total percibido por el usuario durante el aprovisionamiento de las máquinas virtuales.

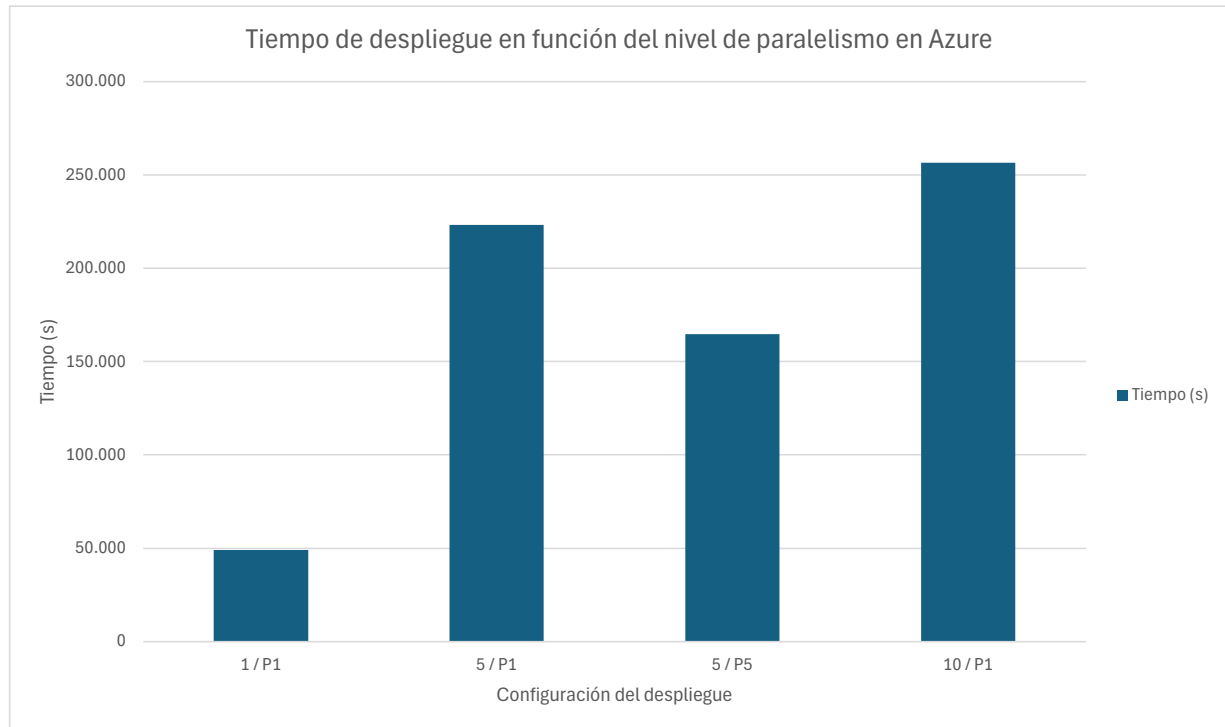


Figura 6.11: Tiempo de despliegue en función del nivel de paralelismo en *Azure*.

La figura 6.11 muestra una comparación de los tiempos de despliegue obtenidos durante las pruebas realizadas en *Azure*. En el escenario de cinco máquinas virtuales, el uso de paralelismo permitió reducir el tiempo de aprovisionamiento de aproximadamente 223 segundos a cerca de 165 segundos. Por otro lado, el escenario correspondiente a diez máquinas virtuales presentó el mayor tiempo promedio de ejecución, con aproximadamente 257 segundos, debido a restricciones de cuota regional impuestas por la plataforma.

Durante las pruebas de creación de diez máquinas virtuales se presentó una limitación relacionada con las cuotas regionales de procesamiento asignadas a la suscripción utilizada. Las instancias configuradas correspondían al tipo `Standard_D2s_v3`, donde cada máquina virtual consume 2 *vCPU*. La suscripción utilizada dentro de la región `southcentralus` contaba con un límite máximo de 10 *vCPU* disponibles, por lo que únicamente fue posible desplegar simultáneamente cinco máquinas virtuales.

Debido a esta restricción, *Azure* rechazó automáticamente la creación de nuevas instancias mediante errores de tipo `409 Conflict`, indicando que la cuota regional había sido excedida. En la figura 6.12 se puede observar el error `409 Conflict` generado durante el aprovisionamiento de las máquinas virtuales. El mensaje indica que la operación no pudo completarse debido a que se excedió la cuota regional de *vCPU* disponible en la región `southcentralus`. Asimismo, se muestra que el límite configurado era de 10 *vCPU* y que la creación de nuevas instancias requería recursos adicionales no disponibles dentro de la suscripción.


```

Error: creating Linux Virtual Machine (Subscription: "3fe13c5e-280c-49a7-a9f6-15ae1fbfdd3f"
Resource Group Name: "rg-terraform-azure-test"
Virtual Machine Name: "vm-terraform-azure-test-2"): performing CreateOrUpdate: unexpected status 409 (409 Conflict) with error: OperationNotAllowed
: Operation could not be completed as it results in exceeding approved Total Regional Cores quota. Additional details - Deployment Model: Resource Ma
nager, Location: southcentralus, Current Limit: 10, Current Usage: 10, Additional Required: 2, (Minimum) New Limit Required: 12. Setup Alerts when Qu
ota reaches threshold. Learn more at https://aka.ms/quotamonitoringalerting . Submit a request for Quota increase at https://aka.ms/ProdportalCRP/#bl
ade/Microsoft_Azure_Capacity/UsageAndQuota.ReactView/Parameters/%7B%22subscriptionId%22:%223fe13c5e-280c-49a7-a9f6-15ae1fbfdd3f%22,%22command%22:%22o
penQuotaApprovalBlade%22,%22quotas%22:%7B%22location%22:%22southcentralus%22,%22providerId%22:%22Microsoft.Compute%22,%22resourceName%22:%22cores%22
,%22quotaRequest%22:%7B%22properties%22:%7B%22limit%22:12,%22unit%22:%22Count%22,%22name%22:%7B%22value%22:%22cores%22%7D%7D%7D%7D%7D by specifying
parameters listed in the 'Details' section for deployment to succeed. Please read more about quota limits at https://docs.microsoft.com/en-us/azure/a
zure-supportability/regional-quota-requests

    with azurerm_linux_virtual_machine.vm[1],
      on main.tf line 117, in resource "azurerm_linux_virtual_machine" "vm":
    117: resource "azurerm_linux_virtual_machine" "vm" {}

Error: creating Linux Virtual Machine (Subscription: "3fe13c5e-280c-49a7-a9f6-15ae1fbfdd3f"
Resource Group Name: "rg-terraform-azure-test"
Virtual Machine Name: "vm-terraform-azure-test-4"): performing CreateOrUpdate: unexpected status 409 (409 Conflict) with error: OperationNotAllowed
: Operation could not be completed as it results in exceeding approved Total Regional Cores quota. Additional details - Deployment Model: Resource Ma
nager, Location: southcentralus, Current Limit: 10, Current Usage: 10, Additional Required: 2, (Minimum) New Limit Required: 12. Setup Alerts when Qu
ota reaches threshold. Learn more at https://aka.ms/quotamonitoringalerting . Submit a request for Quota increase at https://aka.ms/ProdportalCRP/#bl
ade/Microsoft_Azure_Capacity/UsageAndQuota.ReactView/Parameters/%7B%22subscriptionId%22:%223fe13c5e-280c-49a7-a9f6-15ae1fbfdd3f%22,%22command%22:%22o
penQuotaApprovalBlade%22,%22quotas%22:%7B%22location%22:%22southcentralus%22,%22providerId%22:%22Microsoft.Compute%22,%22resourceName%22:%22cores%22
,%22quotaRequest%22:%7B%22properties%22:%7B%22limit%22:12,%22unit%22:%22Count%22,%22name%22:%7B%22value%22:%22cores%22%7D%7D%7D%7D%7D by specifying
parameters listed in the 'Details' section for deployment to succeed. Please read more about quota limits at https://docs.microsoft.com/en-us/azure/a
zure-supportability/regional-quota-requests

    with azurerm_linux_virtual_machine.vm[3],
      on main.tf line 117, in resource "azurerm_linux_virtual_machine" "vm":
    117: resource "azurerm_linux_virtual_machine" "vm" {}

```

Figura 6.12: Errores de cuota regional durante el despliegue de máquinas virtuales en Azure.

La figura 6.13 presenta errores adicionales relacionados con el límite máximo de *vCPU* permitido por la suscripción.

```

Error: creating Linux Virtual Machine (Subscription: "3fe13c5e-280c-49a7-a9f6-15ae1fbfdd3f"
Resource Group Name: "rg-terraform-azure-test"
Virtual Machine Name: "vm-terraform-azure-test-6"): performing CreateOrUpdate: unexpected status 409 (409 Conflict) with error: OperationNotAllowed
: Operation could not be completed as it results in exceeding approved Total Regional Cores quota. Additional details - Deployment Model: Resource Ma
nager, Location: southcentralus, Current Limit: 10, Current Usage: 10, Additional Required: 2, (Minimum) New Limit Required: 12. Setup Alerts when Qu
ota reaches threshold. Learn more at https://aka.ms/quotamonitoringalerting . Submit a request for Quota increase at https://aka.ms/ProdportalCRP/#bl
ade/Microsoft_Azure_Capacity/UsageAndQuota.ReactView/Parameters/%7B%22subscriptionId%22:%223fe13c5e-280c-49a7-a9f6-15ae1fbfdd3f%22,%22command%22:%22o
penQuotaApprovalBlade%22,%22quotas%22:%7B%22location%22:%22southcentralus%22,%22providerId%22:%22Microsoft.Compute%22,%22resourceName%22:%22cores%22
,%22quotaRequest%22:%7B%22properties%22:%7B%22limit%22:12,%22unit%22:%22Count%22,%22name%22:%7B%22value%22:%22cores%22%7D%7D%7D%7D%7D by specifying
parameters listed in the 'Details' section for deployment to succeed. Please read more about quota limits at https://docs.microsoft.com/en-us/azure/a
zure-supportability/regional-quota-requests

    with azurerm_linux_virtual_machine.vm[5],
      on main.tf line 117, in resource "azurerm_linux_virtual_machine" "vm":
    117: resource "azurerm_linux_virtual_machine" "vm" {}

Error: creating Linux Virtual Machine (Subscription: "3fe13c5e-280c-49a7-a9f6-15ae1fbfdd3f"
Resource Group Name: "rg-terraform-azure-test"
Virtual Machine Name: "vm-terraform-azure-test-8"): performing CreateOrUpdate: unexpected status 409 (409 Conflict) with error: OperationNotAllowed
: Operation could not be completed as it results in exceeding approved Total Regional Cores quota. Additional details - Deployment Model: Resource Ma
nager, Location: southcentralus, Current Limit: 10, Current Usage: 10, Additional Required: 2, (Minimum) New Limit Required: 12. Setup Alerts when Qu
ota reaches threshold. Learn more at https://aka.ms/quotamonitoringalerting . Submit a request for Quota increase at https://aka.ms/ProdportalCRP/#bl
ade/Microsoft_Azure_Capacity/UsageAndQuota.ReactView/Parameters/%7B%22subscriptionId%22:%223fe13c5e-280c-49a7-a9f6-15ae1fbfdd3f%22,%22command%22:%22o
penQuotaApprovalBlade%22,%22quotas%22:%7B%22location%22:%22southcentralus%22,%22providerId%22:%22Microsoft.Compute%22,%22resourceName%22:%22cores%22
,%22quotaRequest%22:%7B%22properties%22:%7B%22limit%22:12,%22unit%22:%22Count%22,%22name%22:%7B%22value%22:%22cores%22%7D%7D%7D%7D%7D by specifying
parameters listed in the 'Details' section for deployment to succeed. Please read more about quota limits at https://docs.microsoft.com/en-us/azure/a
zure-supportability/regional-quota-requests

    with azurerm_linux_virtual_machine.vm[7],
      on main.tf line 117, in resource "azurerm_linux_virtual_machine" "vm":
    117: resource "azurerm_linux_virtual_machine" "vm" {}

```

Figura 6.13: Errores adicionales de aprovisionamiento ocasionados por límites regionales de *vCPU* en Azure.

En la figura 6.13 se observan errores adicionales asociados a distintas máquinas virtuales que no lograron aprovisionarse. Esto confirma que la restricción de cuota regional afectó a múltiples instancias durante la misma ejecución y no únicamente a un recurso aislado. Posteriormente, la figura 6.14 muestra la finalización parcial del despliegue y el tiempo total registrado por el comando `time terraform apply`. Aunque algunas máquinas virtuales fueron creadas correctamente, la ejecución concluyó con errores debido a las restricciones de cuota impuestas por la plataforma. Asimismo, se observa el tiempo total registrado por el comando `time`, utilizado para medir la duración de la prueba.

```

Error: creating Linux Virtual Machine (Subscription: "3fe13c5e-280c-49a7-a9f6-15ae1fbfdd3f"
Resource Group Name: "rg-terraform-azure-test"
Virtual Machine Name: "vm-terraform-azure-test-3"): performing CreateOrUpdate: unexpected status 409 (409 Conflict) with error: OperationNotAllowed
: Operation could not be completed as it results in exceeding approved Total Regional Cores quota. Additional details - Deployment Model: Resource Ma
nager, Location: southcentralus, Current Limit: 10, Current Usage: 10, Additional Required: 2, (Minimum) New Limit Required: 12. Setup Alerts when Qu
ota reaches threshold. Learn more at https://aka.ms/quotamonitoringalerting . Submit a request for Quota increase at https://aka.ms/ProdportalCRP/#bl
ade/Microsoft_Azure_Capacity/UsageAndQuota.ReactView/Parameters/%7B%22subscriptionId%22:%223fe13c5e-280c-49a7-a9f6-15ae1fbfdd3f%22,%22command%22:%22o
penQuotaApprovalBlade%22,%22quotas%22:[%7B%22location%22:%22southcentralus%22,%22providerId%22:%22Microsoft.Compute%22,%22resourceName%22:%22cores%22
,%22quotaRequest%22:%7B%22properties%22:%7B%22limit%22:12,%22unit%22:%22Count%22,%22name%22:%7B%22value%22:%22cores%22%7D%7D%7D%7D by specifying
parameters listed in the 'Details' section for deployment to succeed. Please read more about quota limits at https://docs.microsoft.com/en-us/azure/a
zure-supportability/regional-quota-requests

    with azurerm_linux_virtual_machine.vm[2],
    on main.tf line 117, in resource "azurerm_linux_virtual_machine" "vm":
117: resource "azurerm_linux_virtual_machine" "vm" {

real    4m16.593s
user    0m13.536s
sys     0m2.072s

```

Figura 6.14: Finalización parcial del despliegue debido a restricciones de cuota en *Azure*.

Finalmente, la figura 6.15 muestra las máquinas virtuales que lograron aprovisionarse correctamente dentro del límite permitido por la plataforma. En la figura se observan las máquinas virtuales que fueron creadas exitosamente dentro del grupo de recursos *RG-TERRAFORM-AZURE-TEST*. Asimismo, puede verificarse que todas las instancias se encuentran ubicadas en la región *southcentralus*. El resultado confirma que únicamente una parte de las máquinas virtuales solicitadas logró aprovisionarse antes de alcanzar el límite regional de *vCPU* impuesto por la suscripción.

```

[root@controller terraform-azure]# az vm list -o table
Name                                ResourceGroup                       Location                            Zones
-----
vm-terraform-azure-test-1           RG-TERRAFORM-AZURE-TEST            southcentralus                      [
vm-terraform-azure-test-10          RG-TERRAFORM-AZURE-TEST            southcentralus                      [
vm-terraform-azure-test-5           RG-TERRAFORM-AZURE-TEST            southcentralus                      [
vm-terraform-azure-test-7           RG-TERRAFORM-AZURE-TEST            southcentralus                      [
vm-terraform-azure-test-9           RG-TERRAFORM-AZURE-TEST            southcentralus                      [

```

Figura 6.15: Máquinas virtuales creadas exitosamente en *Azure*.

Los resultados obtenidos permitieron observar que el incremento del paralelismo redujo el tiempo total de despliegue cuando los recursos disponibles eran suficientes. Sin embargo, el principal factor limitante del entorno experimental correspondió a las cuotas regionales de procesamiento establecidas por *Azure*, las cuales restringieron el número máximo de máquinas virtuales que podían coexistir simultáneamente dentro de la suscripción utilizada.

6.1.2. Prueba de destrucción de infraestructura

6.1.2.1. Prueba de destrucción en *OpenStack*

La prueba de destrucción de infraestructura en *OpenStack* tuvo como objetivo analizar el comportamiento del proceso automatizado durante la eliminación de máquinas virtuales mediante *Terraform*. Para ello, se utilizó el comando `terraform destroy`, el cual permite eliminar los recursos previamente creados y registrados dentro del archivo de estado. De manera similar a la prueba de creación de infraestructura, se modificó el parámetro `-parallelism` con el propósito de evaluar el efecto de las operaciones concurrentes durante la destrucción de recursos. Este parámetro controla el número máximo de operaciones que *Terraform* puede ejecutar simultáneamente al momento de eliminar las máquinas virtuales. El comando utilizado para las pruebas fue el siguiente:

```
time terraform destroy -auto-approve -parallelism=1
```

En cada escenario se modificó el valor de `-parallelism` de acuerdo con la cantidad de operaciones concurrentes evaluadas.

Con el propósito de evaluar la consistencia de los resultados obtenidos, cada escenario fue ejecutado en 20 ocasiones independientes bajo las mismas condiciones de operación. A partir de estas repeticiones se calcularon la media aritmética, la desviación estándar muestral, la variación típica y el coeficiente de variación (*CV*). Los resultados se presentan en la tabla 6.3.

Escenario	Repeticiones	Media (s)	Desv. estándar	Variación típica	CV (%)	Interpretación
OpenStack P1/1VM	20	12.719	2.145	±2,145	16.87	Consistencia aceptable
OpenStack P1/5VM	20	51.581	4.949	±4,949	9.59	Buena consistencia
OpenStack P5/5VM	20	22.424	3.882	±3,882	17.31	Consistencia aceptable
OpenStack P1/10VM	20	105.814	5.913	±5,913	5.59	Buena consistencia
OpenStack P5/10VM	20	37.596	6.296	±6,296	16.75	Consistencia aceptable
OpenStack P10/10VM	20	32.621	4.265	±4,265	13.08	Consistencia aceptable
OpenStack P1/15VM	20	165.059	3.889	±3,889	2.36	Excelente consistencia
OpenStack P5/15VM	20	53.341	4.678	±4,678	8.77	Buena consistencia
OpenStack P15/15VM	20	43.763	4.412	±4,412	10.08	Consistencia aceptable
OpenStack P1/30VM	20	277.356	5.225	±5,225	1.88	Excelente consistencia
OpenStack P5/30VM	20	100.652	2.451	±2,451	2.44	Excelente consistencia
OpenStack P10/30VM	20	85.618	1.939	±1,939	2.26	Excelente consistencia
OpenStack P15/30VM	20	47.594	1.284	±1,284	2.70	Excelente consistencia
OpenStack P30/30VM	20	56.791	1.513	±1,513	2.66	Excelente consistencia

Tabla 6.3: Resumen estadístico de las pruebas de destrucción de infraestructura en *OpenStack*.

A partir de los resultados mostrados en la Tabla 6.3, se observa que los escenarios con mayor carga de trabajo presentaron coeficientes de variación inferiores al 5 %, lo que indica una excelente consistencia entre repeticiones. Por otro lado, algunos escenarios de menor duración registraron coeficientes de variación entre 10 % y 20 %, clasificados como consistencia aceptable. En conjunto, ninguno de los escenarios superó el umbral de 20 %, por lo que todos los resultados pueden considerarse estadísticamente consistentes. Estas variaciones pueden atribuirse al tiempo de respuesta de los servicios internos de *OpenStack*, la sincronización de las operaciones de eliminación y la carga existente en el entorno durante cada ejecución.

En conjunto, las 20 repeticiones realizadas para cada escenario permitieron comprobar que los resultados obtenidos fueron consistentes y reproducibles. El análisis estadístico confirmó que las diferencias observadas en los tiempos de destrucción estuvieron relacionadas principalmente con el nivel de paralelismo utilizado y con la duración total de cada operación, más que con comportamientos aleatorios del sistema.

El comando `time` permite medir el tiempo total empleado durante la destrucción de la infraestructura. En esta prueba, el valor más importante fue `real`, ya que representa el tiempo total percibido por el usuario durante la eliminación de las máquinas virtuales.

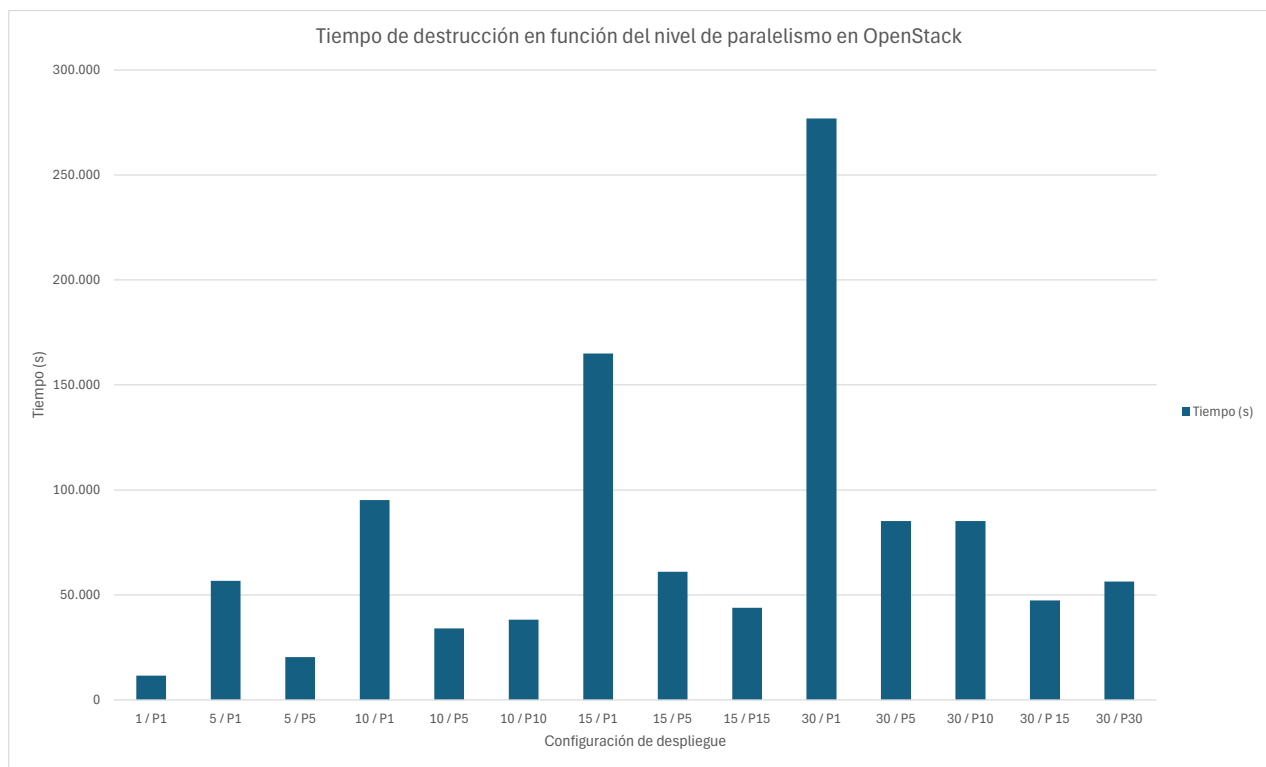


Figura 6.16: Tiempo de destrucción de infraestructura en función del nivel de paralelismo en *OpenStack*.

La figura 6.16 muestra los tiempos obtenidos durante las pruebas de destrucción de infraestructura en *OpenStack*. En el eje horizontal se representan los distintos escenarios evaluados, donde la notación P1 corresponde a paralelismo 1, P5 a paralelismo 5, P10 a paralelismo 10, P15 a paralelismo 15 y P30 a paralelismo 30. Los resultados muestran que el tiempo de destrucción aumenta conforme incrementa la cantidad de máquinas virtuales eliminadas; sin embargo, el uso de paralelismo permitió reducir considerablemente los tiempos totales de ejecución. El escenario correspondiente a una máquina virtual utilizando P1 presentó el menor tiempo total, con aproximadamente 13 segundos.

Posteriormente, en las pruebas correspondientes a cinco máquinas virtuales, el tiempo promedio disminuyó de aproximadamente 52 segundos utilizando P1 a cerca de 22 segundos utilizando P5. En los escenarios correspondientes a diez máquinas virtuales también se observó una reducción progresiva del tiempo conforme aumentó el paralelismo. Utilizando P1, el proceso presentó un tiempo promedio de aproximadamente 106 segundos, mientras que utilizando P5 y P10 los tiempos promedio disminuyeron a cerca de 38 y 33 segundos, respectivamente. Para las pruebas correspondientes a quince máquinas virtuales, el escenario secuencial con P1 presentó un tiempo promedio cercano a 165 segundos. Utilizando P5, el tiempo promedio disminuyó a aproximadamente 53 segundos, mientras que utilizando P15 se redujo hasta cerca de 44 segundos.

Finalmente, en las pruebas correspondientes a treinta máquinas virtuales, el escenario con P1 presentó el mayor tiempo promedio de destrucción, con aproximadamente 277 segundos. Al incrementar el paralelismo, el tiempo promedio disminuyó considerablemente, alcanzando cerca de 101 segundos con P5, 86 segundos con P10, 48 segundos con P15 y aproximadamente 57 segundos con P30.

A diferencia de las pruebas de creación de infraestructura, durante la destrucción de máquinas

virtuales no se presentaron errores asociados a saturación de recursos o limitaciones de memoria *RAM*. Esto indica que las operaciones de eliminación requieren una menor carga computacional respecto al aprovisionamiento simultáneo de nuevas instancias.

En términos generales, los resultados obtenidos permitieron comprobar que el incremento del paralelismo mejora considerablemente la velocidad de destrucción de infraestructura en *OpenStack*, reduciendo el tiempo total requerido para eliminar múltiples recursos virtuales de manera automatizada.

6.1.2.2. Prueba de destrucción en *Azure Portal*

Las pruebas de destrucción en *Azure* tuvieron como objetivo analizar el comportamiento de *Terraform* durante la eliminación automatizada de infraestructura utilizando distintos niveles de paralelismo. Para ello, se evaluó el tiempo requerido para destruir múltiples máquinas virtuales de manera secuencial y concurrente dentro del entorno experimental.

En todas las pruebas se utilizaron los siguientes comandos base:

```
terraform fmt
terraform init

time terraform destroy \
-var="num_vms=5" \
-auto-approve \
-parallelism=1
```

El valor definido en `num_vms` y el parámetro `-parallelism` fueron modificados dependiendo del escenario evaluado.

Con el propósito de evaluar la consistencia de los resultados obtenidos, cada escenario fue ejecutado en 20 ocasiones bajo las mismas condiciones de operación. A partir de estas repeticiones se calcularon la media aritmética, la desviación estándar muestral, la variación típica y el coeficiente de variación (*CV*). Los resultados se presentan en la Tabla 6.4.

Escenario	Repeticiones	Media (s)	Desv. estándar	Variación típica	CV (%)	Interpretación
Azure P1/1VM	20	80.498	1.704	±1,704	2.12	Excelente consistencia
Azure P1/5VM	20	300.084	7.057	±7,057	2.35	Excelente consistencia
Azure P5/5VM	20	167.680	3.396	±3,396	2.03	Excelente consistencia
Azure P1/10VM	20	356.083	9.682	±9,682	2.72	Excelente consistencia

Tabla 6.4: Resumen estadístico de las pruebas de destrucción de infraestructura en *Azure*.

Los resultados muestran coeficientes de variación inferiores al 5% en todos los escenarios evaluados, lo que indica una excelente consistencia entre repeticiones. Asimismo, las 20 ejecuciones realizadas para cada caso permitieron comprobar que los tiempos obtenidos fueron estables y reproducibles bajo las mismas condiciones de operación.

Las pequeñas variaciones observadas pueden atribuirse a factores propios de la infraestructura de *Azure*, como los tiempos de respuesta de los servicios internos de la plataforma y las operaciones de liberación de recursos ejecutadas durante cada destrucción.

En conjunto, el análisis estadístico confirmó que las diferencias observadas en los tiempos de destrucción estuvieron relacionadas principalmente con la cantidad de recursos eliminados y el nivel de paralelismo utilizado, más que con comportamientos aleatorios del sistema.

El comando `time` permite medir el tiempo total empleado durante la destrucción de la infraestructura. En esta prueba, el valor más importante fue `real`, ya que representa el tiempo total percibido por el usuario durante la eliminación de las máquinas virtuales.

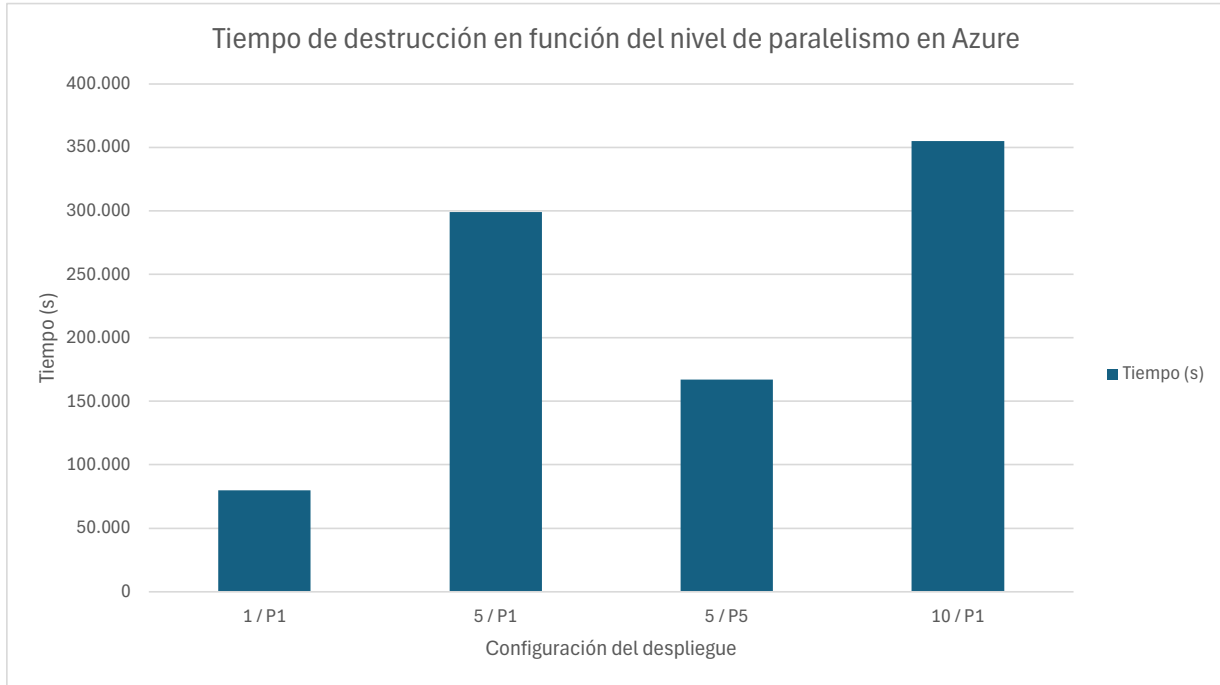


Figura 6.17: Tiempo de destrucción en función del nivel de paralelismo en *Azure*.

La figura 6.17 muestra los tiempos obtenidos durante las pruebas de destrucción de infraestructura en *Azure*. En general, se observó que el incremento del paralelismo permitió reducir considerablemente el tiempo total requerido para eliminar múltiples máquinas virtuales.

En el caso de cinco máquinas virtuales, el uso de paralelismo de 5 redujo el tiempo de destrucción de aproximadamente 299 segundos a cerca de 167 segundos respecto al proceso secuencial utilizando paralelismo de 1.

Asimismo, el escenario correspondiente a diez máquinas virtuales utilizando paralelismo de 1 presentó el mayor tiempo promedio de destrucción, con aproximadamente 356 segundos, debido a la cantidad total de recursos que debían eliminarse dentro de la plataforma. Sin embargo, el proceso logró completarse correctamente sin errores asociados a cuotas regionales, ya que durante la destrucción únicamente se liberan recursos previamente aprovisionados.

A partir de los resultados obtenidos, se observó que el paralelismo no solamente beneficia el tiempo de creación de infraestructura, sino también las tareas de eliminación de recursos. Esto permitió reducir los tiempos totales de administración de infraestructura dentro del entorno experimental evaluado.

6.1.3. Prueba de actualización de infraestructura

6.1.3.1. Prueba de actualización en *OpenStack*

La prueba de actualización de infraestructura tuvo como objetivo analizar la capacidad del entorno *OpenStack* para modificar características de máquinas virtuales ya desplegadas, específicamente mediante operaciones de cambio de nombre y redimensionamiento de recursos utilizando

distintos *flavors*. Esta prueba permitió evaluar el comportamiento del entorno durante operaciones de escalabilidad vertical y verificar la respuesta de los servicios internos de *OpenStack* durante el proceso de actualización.

Como preparación para las pruebas de actualización, fue necesario modificar el archivo de configuración `/etc/nova/nova.conf` tanto en el nodo *controller* como en el nodo *compute*, con el propósito de permitir operaciones de *resize* dentro del mismo nodo *compute*. Dentro de la sección `[DEFAULT]` se agregó la siguiente línea:

```
[DEFAULT]
allow_resize_to_same_host = True
```

Después de realizar la modificación, fue necesario reiniciar los servicios correspondientes y cargar nuevamente las credenciales administrativas de *OpenStack*. Como preparación para las pruebas, se crearon dos nuevos *flavors* denominados `m1.medium` y `m1.big`. El primero fue configurado con 4096 MB de memoria *RAM*, 40 GB de almacenamiento y 2 *vCPU*, mientras que el segundo fue configurado con 8192 MB de memoria *RAM*, 80 GB de almacenamiento y 4 *vCPU*. Las figuras 6.18 y 6.19 muestran la creación de ambos *flavors* dentro del entorno *OpenStack*.

```
[root@controller terraform-openstack]# openstack flavor create m1.medium \
--ram 4096 \
--disk 40 \
--vcpus 2
```

Field	Value
OS-FLV-DISABLED:disabled	False
OS-FLV-EXT-DATA:ephemeral	0
description	None
disk	40
id	324aa68a-d21f-40b4-90b1-0dd226cd9f66
name	m1.medium
os-flavor-access:is_public	True
properties	
ram	4096
rxtx_factor	1.0
swap	0
vcpus	2

Figura 6.18: Creación del *flavor* `m1.medium` en *OpenStack*.

En la figura 6.18 se observa la creación del *flavor* `m1.medium`. La configuración asigna 4096 MB de memoria *RAM*, 40 GB de almacenamiento y 2 *vCPU*, recursos utilizados para evaluar operaciones de redimensionamiento durante la prueba de actualización.

```
[root@controller terraform-openstack]# openstack flavor create m1.big \
--ram 8192 \
--disk 80 \
--vcpus 4
```

Field	Value
OS-FLV-DISABLED:disabled	False
OS-FLV-EXT-DATA:ephemeral	0
description	None
disk	80
id	4bc851d6-744f-43cb-99f7-d3c697ddd8c8
name	m1.big
os-flavor-access:is_public	True
properties	
ram	8192
rxtx_factor	1.0
swap	0
vcpus	4

Figura 6.19: Creación del *flavor* `m1.big` en *OpenStack*.

En la figura 6.19 se muestra la creación del *flavor* `m1.big`. Este *flavor* fue configurado con 8192 MB de memoria *RAM*, 80 GB de almacenamiento y 4 *vCPU*, permitiendo disponer de una configuración con mayores recursos para futuras pruebas de escalabilidad vertical. Posteriormente, se desplegaron 3 máquinas virtuales para realizar las pruebas de actualización. Una vez creadas

las instancias, se generó una carpeta destinada al almacenamiento de evidencias y estados previos. Los comandos utilizados para almacenar las evidencias iniciales fueron los siguientes:

```
mkdir evidencia_actualizacion_2
mkdir evidencia_tfstate_2

openstack server list --long

openstack server show vm-terraform-1 > evidencia_actualizacion_2/antes_vm1.txt

openstack server show vm-terraform-2 > evidencia_actualizacion_2/antes_vm2.txt

openstack server show vm-terraform-3 > evidencia_actualizacion_2/antes_vm3.txt
```

Seguido, se usaron los siguientes comandos para verificar los *flavors* actuales de las 3 máquinas virtuales creadas:

```
openstack server show vm-terraform-1 -c name -c flavor
openstack server show vm-terraform-2 -c name -c flavor
openstack server show vm-terraform-3 -c name -c flavor
```

La figura 6.20 muestra la generación de evidencias antes de realizar modificaciones dentro de la infraestructura, en esta figura se observa el registro de las máquinas virtuales existentes antes de realizar cualquier modificación. Asimismo, se muestra la generación de archivos de evidencia para cada instancia, los cuales fueron utilizados posteriormente para comparar el estado de la infraestructura antes y después de la actualización.

```
[root@controller terraform-openstack]# mkdir evidencia_actualizacion_1
mkdir evidencia_tfstate_1
[root@controller terraform-openstack]# openstack server list --long
```

ID	Flavor	Availability Zone	Name	Status	Task State	Power State	Networks	Image Name	Image ID
33aa855d-dfa9-4414-accf-33c1dd1e07b7	m1.nanotiny nova		vm-terraform-1	ACTIVE	None	Running	private=192.168.100.212	cirros	9681ce2d-b638-43b5-ae1a-4df0d220
000f982e-6061-4ca8-afb8-4da742264df9	m1.nanotiny nova		vm-terraform-3	ACTIVE	None	Running	private=192.168.100.58	cirros	9681ce2d-b638-43b5-ae1a-4df0d220
d082005e-b300-4ad1-b1cf-dff7d29c3bb7	m1.nanotiny nova		vm-terraform-2	ACTIVE	None	Running	private=192.168.100.38	cirros	9681ce2d-b638-43b5-ae1a-4df0d220

Figura 6.20: Generación de evidencias previas a la actualización de infraestructura.

```
[root@controller terraform-openstack]# openstack server show vm-terraform-1 -c name -c flavor
openstack server show vm-terraform-2 -c name -c flavor
openstack server show vm-terraform-3 -c name -c flavor
```

Field	Value
flavor	m1.nanotiny
name	vm-terraform-1

Field	Value
flavor	m1.nanotiny
name	vm-terraform-2

Field	Value
flavor	m1.nanotiny
name	vm-terraform-3

Figura 6.21: Verificación de *flavors* actuales.

En la figura 6.21 puede observarse que las tres máquinas virtuales utilizaban inicialmente el *flavor* `m1.nanotiny`. La información mostrada confirma que cada instancia disponía de 1 *vCPU*, 256 MB de memoria *RAM* y 1 GB de almacenamiento antes de ejecutar las operaciones de actualización. Después de almacenar las evidencias iniciales, se realizaron operaciones de actualización sobre distintas máquinas virtuales. Primero, se modificó el nombre de una instancia utilizando el comando `openstack server set`. Posteriormente, se redimensionó la segunda máquina virtual mediante operaciones de *resize* hacia el *flavor* `m1.medium`. Los comandos ejecutados fueron los siguientes:

```
openstack server set --name vm-terraform-1 \
UNAM-TELE-FI-MX

openstack server resize --flavor m1.medium \
vm-terraform-2

openstack server resize confirm vm-terraform-2

openstack server resize --flavor m1.medium \
vm-terraform-3

openstack server resize confirm vm-terraform-3
```

Finalmente, después de realizar las operaciones de actualización, se almacenaron nuevamente las evidencias correspondientes al estado actual de las máquinas virtuales con el propósito de comparar los cambios realizados sobre la infraestructura. Los comandos utilizados fueron los siguientes:

```

openstack server show UNAM-TELE-FI-MX \
> evidencia_actualizacion_1/despues_vm1.txt

openstack server show vm-terraform-2 \
> evidencia_actualizacion_1/despues_vm2.txt

openstack server show UNAM-TELE-FI-MX-3 \
> evidencia_actualizacion_1/despues_vm3.txt

```

La figura 6.22 muestra el almacenamiento de evidencias después de realizar las operaciones de actualización sobre las máquinas virtuales. Estos registros permitieron verificar los cambios aplicados sobre las máquinas virtuales y comparar el estado final de la infraestructura con las evidencias obtenidas antes de realizar las modificaciones.

```

[root@controller terraform-openstack]# openstack server show UNAM-TELE-FI-MX > evidencia_actualizacion_1/despues_vm1.txt
openstack server show vm-terraform-2 > evidencia_actualizacion_1/despues_vm2.txt
openstack server show UNAM-TELE-FI-MX-3 > evidencia_actualizacion_1/despues_vm3.txt

```

Figura 6.22: Generación de evidencias posteriores a la actualización de infraestructura.

6.1.3.2. Prueba de actualización en *Azure Portal*

La prueba de actualización de infraestructura en *Azure* tuvo como objetivo evaluar la capacidad de modificar recursos previamente desplegados mediante *Terraform* y comandos de *Azure CLI*. Esta prueba permitió analizar el comportamiento del entorno frente a cambios manuales realizados directamente sobre las máquinas virtuales y verificar la detección de diferencias entre el estado real de la infraestructura y el archivo de estado administrado por *Terraform*.

Inicialmente, se desplegaron tres máquinas virtuales dentro del entorno experimental. La figura 6.23 muestra el resultado del despliegue inicial y los nombres originales asignados a las instancias creadas. En la figura se observa la creación inicial de tres máquinas virtuales denominadas *vm-terraform-azure-test-1*, *vm-terraform-azure-test-2* y *vm-terraform-azure-test-3*. Asimismo, se muestra la generación de evidencias correspondientes al estado inicial de la infraestructura y al archivo de estado administrado por *Terraform*, información utilizada posteriormente para comparar los cambios realizados durante la prueba de actualización.

```

Apply complete! Resources: 9 added, 0 changed, 0 destroyed.

Outputs:

vm_ids = [
  "/subscriptions/3fe13c5e-280c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terrafor
m-azure-test-1",
  "/subscriptions/3fe13c5e-280c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terrafor
m-azure-test-2",
  "/subscriptions/3fe13c5e-280c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terrafor
m-azure-test-3",
]

real    1m51.650s
user    0m11.884s
sys     0m1.737s
[root@controller terraform-azure]# mkdir -p evidencia_actualizacion_azure_cli
[root@controller terraform-azure]# mkdir -p evidencia_actualizacion_azure_cli
az vm list -o table
az vm list --show-details -o table > evidencia_actualizacion_azure_cli/vms_antes.txt
terraform state list > evidencia_actualizacion_azure_cli/state_antes.txt
Name                                ResourceGroup      Location            Zones
-----
vm-terraform-azure-test-1           RG-TERRAFORM-AZURE-TEST  southcentralus
vm-terraform-azure-test-2           RG-TERRAFORM-AZURE-TEST  southcentralus
vm-terraform-azure-test-3           RG-TERRAFORM-AZURE-TEST  southcentralus

```

Figura 6.23: Creación inicial de máquinas virtuales en *Azure*.

Después, se realizaron modificaciones manuales sobre las máquinas virtuales utilizando comandos de *Azure CLI*. En este caso, se agregaron etiquetas (*tags*) con nombres personalizados a cada instancia mediante los siguientes comandos:

```
az vm update \
  --resource-group rg-terraform-azure-test \
  --name vm-terraform-azure-test-1 \
  --set tags.NombrePrueba=UNAM-FI-TELE-MX

az vm update \
  --resource-group rg-terraform-azure-test \
  --name vm-terraform-azure-test-2 \
  --set tags.NombrePrueba=UNAM-FI-MX

az vm update \
  --resource-group rg-terraform-azure-test \
  --name vm-terraform-azure-test-3 \
  --set tags.NombrePrueba=UNAM-FI-CLOUD
```

Una vez aplicados los cambios, se almacenaron evidencias del nuevo estado de la infraestructura y del archivo administrado por *Terraform*.

```
mkdir -p evidencia_actualizacion_azure_cli

az vm list --show-details -o table \
> evidencia_actualizacion_azure_cli/vms_despues_nombre.txt

terraform state list \
> evidencia_actualizacion_azure_cli/state_despues_nombre.txt
```

A continuación, se verificó la disponibilidad de tamaños de máquina virtual permitidos dentro de la región *southcentralus*. Para ello, se utilizó el siguiente comando:

```
az vm list-skus \
  --location southcentralus \
  --size Standard_D \
  --all -o table
```

En la columna **Restrictions** debía aparecer el valor **None** para indicar que el tamaño seleccionado no presentaba restricciones regionales. La figura 6.24 muestra la verificación de tamaños de máquina virtual disponibles dentro de la región utilizada. Se muestran los tamaños de máquina virtual disponibles dentro de la región *southcentralus*. La columna **Restrictions** permitió identificar qué configuraciones podían utilizarse para operaciones de actualización y cuáles presentaban limitaciones relacionadas con disponibilidad regional o compatibilidad de recursos.

ResourceType	Locations	Name	Zones	Restrictions
virtualMachines	southcentralus	Standard_D1	3	['NotAvailableForSubscription, type: Location, locations: southcentralus', 'NotAvailableForSubscription, type: Zone, locations: southcentralus, zon
es: 1,2,3']				
virtualMachines	southcentralus	Standard_D11	3	['NotAvailableForSubscription, type: Location, locations: southcentralus', 'NotAvailableForSubscription, type: Zone, locations: southcentralus, zon
es: 1,2,3']				
virtualMachines	southcentralus	Standard_D11_v2	1,2,3	['NotAvailableForSubscription, type: Location, locations: southcentralus', 'NotAvailableForSubscription, type: Zone, locations: southcentralus, zon
es: 1,2,3']				
virtualMachines	southcentralus	Standard_D11_v2_Promo	1,2,3	['NotAvailableForSubscription, type: Location, locations: southcentralus', 'NotAvailableForSubscription, type: Zone, locations: southcentralus, zon
es: 1,2,3']				
virtualMachines	southcentralus	Standard_D12	3	['NotAvailableForSubscription, type: Location, locations: southcentralus', 'NotAvailableForSubscription, type: Zone, locations: southcentralus, zon
es: 1,2,3']				
virtualMachines	southcentralus	Standard_D128ads_v7	2,3	None
virtualMachines	southcentralus	Standard_D128ads_v7	2,3	None
virtualMachines	southcentralus	Standard_D128als_v7	2,3	None
virtualMachines	southcentralus	Standard_D128as_v7	2,3	None
virtualMachines	southcentralus	Standard_D128ds_v6	1,2,3	NotAvailableForSubscription, type: Zone, locations: southcentralus, zones: 1
virtualMachines	southcentralus	Standard_D128ds_v6	1,2,3	NotAvailableForSubscription, type: Zone, locations: southcentralus, zones: 1
virtualMachines	southcentralus	Standard_D128ls_v6	1,2,3	NotAvailableForSubscription, type: Zone, locations: southcentralus, zones: 1
virtualMachines	southcentralus	Standard_D128s_v6	1,2,3	NotAvailableForSubscription, type: Zone, locations: southcentralus, zones: 1
virtualMachines	southcentralus	Standard_D12_v2	1,2,3	['NotAvailableForSubscription, type: Location, locations: southcentralus', 'NotAvailableForSubscription, type: Zone, locations: southcentralus, zon
es: 1,2,3']				

Figura 6.24: Verificación de tamaños de máquina virtual disponibles en *Azure*.

Más adelante, se realizó una prueba de actualización del tamaño de una máquina virtual. Primero, la instancia fue apagada utilizando el siguiente comando:

```
az vm deallocate \
-g rg-terraform-azure-test \
-n vm-terraform-azure-test-1
```

Después, se intentó modificar el tamaño de la máquina virtual mediante:

```
az vm resize \
-g rg-terraform-azure-test \
-n vm-terraform-azure-test-1 \
--size Standard_D1
```

Durante esta prueba, *Azure* presentó restricciones relacionadas con disponibilidad regional de tamaños de máquina virtual, compatibilidad de generación de hipervisor y cuotas de recursos, lo cual impactó directamente en la capacidad de actualización dinámica de infraestructura. La figura 6.25 muestra las restricciones relacionadas con la región, compatibilidad del hipervisor y cuotas de recursos disponibles. Asimismo, se observan los errores generados durante el intento de redimensionamiento de la máquina virtual. Los mensajes indican que los tamaños **Standard_D1** y **Standard_D12_v2** no se encontraban disponibles dentro de la región **southcentralus** al momento de la prueba, por lo que la operación de actualización no pudo completarse exitosamente. Esta situación evidenció que la capacidad de actualización de infraestructura en *Azure* depende no solamente de la configuración definida por el usuario, sino también de la disponibilidad de recursos dentro de la región seleccionada.

```
ls_v2,Standard_B4ps_v2,Standard_B8ps_v2,Stand[roo@controller terraform-azure]# az vm deallocate \
-g rg-terraform-azure-test \
-n vm-terraform-azure-test-1
[roo@controller terraform-azure]# az vm resize \
-g rg-terraform-azure-test \
-n vm-terraform-azure-test-1 \
--size Standard_D1
[SkNotAvailable] The requested VM size for resource 'Following SKUs have failed for Capacity Restrictions: Standard_D1' is currently not available in location 'southcentralus'. Please try another size or deploy t
to a different location or different zone. See https://aka.ms/azuresknotavailable for details.
Code: SkNotAvailable
Message: The requested VM size for resource 'Following SKUs have failed for Capacity Restrictions: Standard_D1' is currently not available in location 'southcentralus'. Please try another size or deploy to a diffe
rent location or different zone. See https://aka.ms/azuresknotavailable for details.
Target: vmSize
[roo@controller terraform-azure]# az vm resize \
-g rg-terraform-azure-test \
-n vm-terraform-azure-test-1 \
--size Standard_D12_v2
[SkNotAvailable] The requested VM size for resource 'Following SKUs have failed for Capacity Restrictions: Standard_D12_v2' is currently not available in location 'southcentralus'. Please try another size or depl
oy to a different location or different zone. See https://aka.ms/azuresknotavailable for details.
Code: SkNotAvailable
Message: The requested VM size for resource 'Following SKUs have failed for Capacity Restrictions: Standard_D12_v2' is currently not available in location 'southcentralus'. Please try another size or deploy to a d
ifferent location or different zone. See https://aka.ms/azuresknotavailable for details.
Target: vmSize
[roo@controller terraform-azure]# az vm start \
-g rg-terraform-azure-test \
-n vm-terraform-azure-test-1
[roo@controller terraform-azure]#
```

Figura 6.25: Restricciones regionales durante la actualización de infraestructura en *Azure*.

Posteriormente, la máquina virtual fue iniciada nuevamente mediante:

```
az vm start \  
-g rg-terraform-azure-test \  
-n vm-terraform-azure-test-1
```

Después del reinicio, se verificó nuevamente el estado de la infraestructura utilizando el comando:

```
az vm list -o table
```

Además, se almacenaron nuevas evidencias relacionadas con el estado actualizado del entorno:

```
mkdir -p evidencia_actualizacion_azure_cli  
  
az vm list --show-details -o table \  
> evidencia_actualizacion_azure_cli/vms_despues.txt  
  
terraform state list \  
> evidencia_actualizacion_azure_cli/state_despues.txt  
  
terraform show \  
> evidencia_actualizacion_azure_cli/show_despues.txt
```

Finalmente, se ejecutó el siguiente comando:

```
terraform plan -var="num_vms=3"
```

El objetivo fue detectar las diferencias entre el estado real de la infraestructura y el archivo `.tfstate` administrado por *Terraform*. La figura 6.26 muestra cómo *Terraform* detectó automáticamente las modificaciones manuales realizadas sobre las etiquetas de las máquinas virtuales. En particular, las etiquetas `NombrePrueba` agregadas mediante *Azure CLI* aparecen como diferencias respecto al estado almacenado en el archivo `.tfstate`, evidenciando la capacidad de la herramienta para identificar situaciones de *drift*. Asimismo, el plan generado indica que las tres máquinas virtuales serían actualizadas para restaurar la configuración declarada originalmente en el código de infraestructura.

A partir de los resultados obtenidos, fue posible comprobar que *Terraform* fue capaz de identificar automáticamente modificaciones manuales realizadas directamente desde *Azure CLI*. Asimismo, las pruebas evidenciaron que la actualización dinámica de infraestructura en *Azure* puede verse limitada por restricciones regionales, cuotas de procesamiento y disponibilidad de tamaños de máquina virtual dentro de determinadas regiones geográficas.

```
[root@controller terraform-azure]# terraform plan -var="num_vms=3"
azurerm_resource_group.rg: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test]
azurerm_virtual_network.vnet: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Network/virtualNetworks/vnet-terraform-test]
azurerm_subnet.subnet: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Network/virtualNetworks/vnet-terraform-test/subnets/subnet-terraform-test]
azurerm_network_interface.nic[2]: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Network/networkInterfaces/vm-terraform-azure-test-3-nic]
azurerm_network_interface.nic[1]: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Network/networkInterfaces/vm-terraform-azure-test-2-nic]
azurerm_network_interface.nic[0]: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Network/networkInterfaces/vm-terraform-azure-test-1-nic]
azurerm_linux_virtual_machine.vm[1]: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-2]
azurerm_linux_virtual_machine.vm[2]: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-3]
azurerm_linux_virtual_machine.vm[0]: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-1]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
- update in-place

Terraform will perform the following actions:

# azurerm_linux_virtual_machine.vm[0] will be updated in-place
- resource "azurerm_linux_virtual_machine" "vm" {
  id = "/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-1"
  name = "vm-terraform-azure-test-1"
  tags = {
    "NombrePrueba" = "UNAM-FI-TELE-MX" -> null
  }
  # (41 unchanged attributes hidden)
}
# (2 unchanged blocks hidden)

# azurerm_linux_virtual_machine.vm[1] will be updated in-place
- resource "azurerm_linux_virtual_machine" "vm" {
  id = "/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-2"
  name = "vm-terraform-azure-test-2"
  tags = {
    "NombrePrueba" = "UNAM-FI-MX" -> null
  }
  # (41 unchanged attributes hidden)
}
# (2 unchanged blocks hidden)

# azurerm_linux_virtual_machine.vm[2] will be updated in-place
- resource "azurerm_linux_virtual_machine" "vm" {
  id = "/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-3"
  name = "vm-terraform-azure-test-3"
  tags = {
    "NombrePrueba" = "UNAM-FI-CLOUD" -> null
  }
  # (41 unchanged attributes hidden)
}
# (2 unchanged blocks hidden)

Plan: 0 to add, 3 to change, 0 to destroy.
```

Figura 6.26: Detección de *drift* mediante `terraform plan` en *Azure*.

6.1.4. Validación del archivo de estado

6.1.4.1. Prueba de validación del archivo de estado en *OpenStack*

La prueba de validación del archivo de estado tuvo como objetivo comprobar la capacidad de *Terraform* para detectar modificaciones manuales realizadas directamente sobre la infraestructura desplegada en *OpenStack*. Esta prueba permitió analizar el comportamiento del archivo `terraform.tfstate` y verificar la sincronización entre el estado declarado en el archivo `main.tf` y el estado real de las máquinas virtuales.

Inicialmente, se compararon las evidencias almacenadas antes y después de las modificaciones realizadas durante las pruebas de actualización de infraestructura. Para ello, se utilizaron los siguientes comandos:

```
diff evidencia_actualizacion_2/antes_vm1.txt \
evidencia_actualizacion_2/despues_vm1.txt

diff evidencia_actualizacion_2/antes_vm2.txt \
evidencia_actualizacion_2/despues_vm2.txt

diff evidencia_actualizacion_2/antes_vm3.txt \
evidencia_actualizacion_2/despues_vm3.txt
```

Las figuras 6.27, 6.28 y 6.29 muestran las diferencias detectadas entre los estados previos y posteriores de las máquinas virtuales después de realizar las modificaciones manuales.

```
[root@controller terraform-openstack]# diff evidencia_actualizacion_1/antes_vm1.txt evidencia_actualizacion_1/después_vm1.txt
diff evidencia_actualizacion_1/antes_vm2.txt evidencia_actualizacion_1/después_vm2.txt
diff evidencia_actualizacion_1/antes_vm3.txt evidencia_actualizacion_1/después_vm3.txt
36c36
< | name | vm-terraform-1
> | name | UNAM-TELE-FI-MX
48c48
< | updated | 2026-05-28T06:29:18Z
> | updated | 2026-05-28T06:44:44Z
1,48c1,48
<
>
< | Field | Value
<
< | OS-DCF:diskConfig | MANUAL
< | OS-EXT-AZ:availability_zone | nova
< | OS-EXT-SRV-ATTR:host | compute
< | OS-EXT-SRV-ATTR:host_name | vm-terraform-2
< | OS-EXT-SRV-ATTR:hypervisor_hostname | compute
< | OS-EXT-SRV-ATTR:instance_name | instance-00000305
< | OS-EXT-SRV-ATTR:kernel_id | 0
< | OS-EXT-SRV-ATTR:launch_index | 0
< | OS-EXT-SRV-ATTR:ramdisk_id |
< | OS-EXT-SRV-ATTR:reservation_id | r-6c0e011p
< | OS-EXT-SRV-ATTR:root_device_name | /dev/vda
< | OS-EXT-SRV-ATTR:user_data |
< | OS-EXT-STS:power_state | Running
< | OS-EXT-STS:task_state | None
< | OS-EXT-STS:vm_state | active
< | OS-SRV-USD:launched_at | 2026-05-28T06:29:03.000000
< | OS-SRV-USD:terminated_at | None
< | accessIPv4 |
< | accessIPv6 |
< | addresses | private=192.168.100.38
< | config_drive |
< | created | 2026-05-28T06:28:40Z
< | description | vm-terraform-2
< | flavor | description=, disk=1, ephemeral=0, , id=ml.nanotiny, is_disabled=, is_public=True, location=, name=ml.nanotiny, original_name=ml.nanotiny, ram=256, rxtx_factor=, swap=0, vcpus=1
< | hostId | 16124831a12599af9408e9fad62b1346c75d30f9eb33bccb66ad5b
< | hostStatus | UP
< | id | 0822805e-b300-4ad1-b1cf-dff7d29c3bb7
< | image | cirros (9681ce2d-b638-43b5-9e1a-4d70d2283df1)
< | key_name | None
< | locked | False
< | locked_reason | None
< | name | vm-terraform-2
< | pinned_availability_zone |
< | progress | 0
< | project_id | 5b9dfeb866ad42818d9b3781fedda7b
< | properties |
< | security_groups | name='default'
< | server_groups | None
< | status | ACTIVE
< | tags |
< | trusted_image_certificates | None
< | updated | 2026-05-28T06:29:03Z
< | user_id | 0733b232bb17472c0bf28b7e3c8eb309
< | volumes_attached |
```

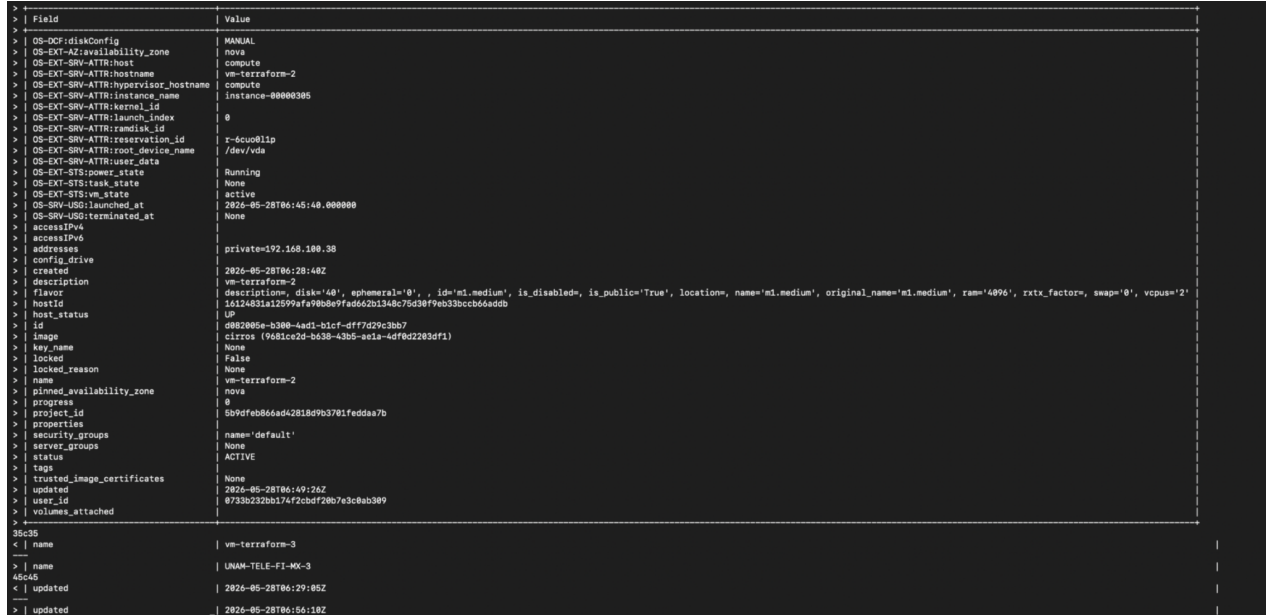
Figura 6.27: Comparación de evidencias antes y después de modificar la máquina virtual vm-terraform-1.

En la figura 6.27 se observa que la principal diferencia detectada corresponde al nombre de la máquina virtual. Inicialmente la instancia se encontraba registrada como vm-terraform-1, mientras que después de la modificación manual se denominó UNAM-TELE-FI-MX. Asimismo, se aprecia una actualización en la marca temporal (updated), lo que confirma que la infraestructura fue modificada correctamente.

```
> | Field | Value
[root@controller terraform-openstack]# diff evidencia_actualizacion_1/antes_vm1.txt evidencia_actualizacion_1/después_vm1.txt
diff evidencia_actualizacion_1/antes_vm2.txt evidencia_actualizacion_1/después_vm2.txt
diff evidencia_actualizacion_1/antes_vm3.txt evidencia_actualizacion_1/después_vm3.txt
36c36
< | name | vm-terraform-1
> | name | UNAM-TELE-FI-MX
48c48
< | updated | 2026-05-28T06:29:18Z
> | updated | 2026-05-28T06:44:44Z
1,48c1,48
<
>
< | Field | Value
<
< | OS-DCF:diskConfig | MANUAL
< | OS-EXT-AZ:availability_zone | nova
< | OS-EXT-SRV-ATTR:host | compute
< | OS-EXT-SRV-ATTR:host_name | vm-terraform-2
< | OS-EXT-SRV-ATTR:hypervisor_hostname | compute
< | OS-EXT-SRV-ATTR:instance_name | instance-00000305
< | OS-EXT-SRV-ATTR:kernel_id | 0
< | OS-EXT-SRV-ATTR:launch_index | 0
< | OS-EXT-SRV-ATTR:ramdisk_id |
< | OS-EXT-SRV-ATTR:reservation_id | r-6c0e011p
< | OS-EXT-SRV-ATTR:root_device_name | /dev/vda
< | OS-EXT-SRV-ATTR:user_data |
< | OS-EXT-STS:power_state | Running
< | OS-EXT-STS:task_state | None
< | OS-EXT-STS:vm_state | active
< | OS-SRV-USD:launched_at | 2026-05-28T06:29:03.000000
< | OS-SRV-USD:terminated_at | None
< | accessIPv4 |
< | accessIPv6 |
< | addresses | private=192.168.100.38
< | config_drive |
< | created | 2026-05-28T06:28:40Z
< | description | vm-terraform-2
< | flavor | description=, disk=1, ephemeral=0, , id=ml.nanotiny, is_disabled=, is_public=True, location=, name=ml.nanotiny, original_name=ml.nanotiny, ram=256, rxtx_factor=, swap=0, vcpus=1
< | hostId | 16124831a12599af9408e9fad62b1346c75d30f9eb33bccb66ad5b
< | hostStatus | UP
< | id | 0822805e-b300-4ad1-b1cf-dff7d29c3bb7
< | image | cirros (9681ce2d-b638-43b5-9e1a-4d70d2283df1)
< | key_name | None
< | locked | False
< | locked_reason | None
< | name | vm-terraform-2
< | pinned_availability_zone |
< | progress | 0
< | project_id | 5b9dfeb866ad42818d9b3781fedda7b
< | properties |
< | security_groups | name='default'
< | server_groups | None
< | status | ACTIVE
< | tags |
< | trusted_image_certificates | None
< | updated | 2026-05-28T06:29:03Z
< | user_id | 0733b232bb17472c0bf28b7e3c8eb309
< | volumes_attached |
```

Figura 6.28: Comparación de evidencias antes y después de realizar *resize* sobre la máquina virtual vm-terraform-2.

En la figura 6.28 se muestran las diferencias generadas después de realizar la operación de *resize* sobre la máquina virtual `vm-terraform-2`. Puede observarse que el *flavor* cambió de `m1.nanotiny` a `m1.medium`, incrementando la cantidad de memoria *RAM*, almacenamiento y *vCPU* asignados a la instancia. Estas diferencias evidencian que la actualización de recursos fue aplicada exitosamente dentro de *OpenStack*.



Field	Value
OS-DCF:diskConfig	MANUAL
OS-EXT-AZ:availability_zone	nova
OS-EXT-SRV-ATTR:host	compute
OS-EXT-SRV-ATTR:hostnames	vm-terraform-2
OS-EXT-SRV-ATTR:hypervisor_hostname	compute
OS-EXT-SRV-ATTR:instance_name	instance-00000395
OS-EXT-SRV-ATTR:kernel_id	0
OS-EXT-SRV-ATTR:launch_index	0
OS-EXT-SRV-ATTR:ramdisk_id	
OS-EXT-SRV-ATTR:reservation_id	r-6c0u0l1p
OS-EXT-SRV-ATTR:root_device_name	/dev/vda
OS-EXT-SRV-ATTR:source_data	
OS-EXT-STS:power_state	Running
OS-EXT-STS:task_state	None
OS-EXT-STS:vm_state	active
OS-SRV-USG:launched_at	2026-05-28T06:45:48.000000
OS-SRV-USG:terminated_at	None
accessIPv4	
accessIPv6	
addresses	private=192.168.100.30
config_drive	
created	2026-05-28T06:28:48Z
description	vm-terraform-2
flavor	description: disk=40', ephemeral=0', id='m1.medium', is_disabled=, is_public=True', location=, name='m1.medium', original_name='m1.medium', ram='4096', rxtx_factor=, swap=0', vcpus='2'
hostId	1612451a12599afa9808e9fad62b114bc76d38f9ab33bccb66addb
host_status	UP
id	d882080e-5300-4ad1-b1cf-dff7d29c30b7
image	cirros (9681cc28-b638-43b5-a01a-4c70d2283df1)
key_name	None
locked	False
locked_reason	None
name	vm-terraform-2
pinned_availability_zone	nova
progress	0
project_id	8b9dfdeb666ad42818d9b3701fedda57b
properties	
security_groups	name='default'
server_groups	None
status	ACTIVE
tags	
trusted_image_certificates	None
updated	2026-05-28T06:49:26Z
user_id	0733b232bd174f2cbd720b7e3c8ab389
volumes_attached	

35c35	vm-terraform-3
48cd5	UNAM-TELE-FI-MX-3
updated	2026-05-28T06:29:05Z
updated	2026-05-28T06:56:18Z

Figura 6.29: Comparación de evidencias antes y después de realizar *resize* sobre la máquina virtual `vm-terraform-3`.

En la figura 6.29 se observan las modificaciones realizadas sobre la máquina virtual `vm-terraform-3`. Además del cambio de nombre a `UNAM-TELE-FI-MX-3`, también puede apreciarse la actualización del *flavor* utilizado por la instancia, reflejando el incremento de recursos asignados. La comparación entre ambos estados permitió verificar que los cambios efectuados fueron registrados correctamente por la plataforma. Posteriormente, se ejecutó el siguiente comando para analizar las diferencias existentes entre la infraestructura declarada en el archivo `main.tf`, el archivo `terraform.tfstate` y el estado real de *OpenStack*:

```
terraform plan | tee \
evidencia_tfstate_2/plan_drift.txt
```

La figura 6.30 muestra el resultado obtenido después de ejecutar `terraform plan`, donde *Terraform* detectó automáticamente las modificaciones manuales realizadas sobre las máquinas virtuales. En particular, se identificaron cambios relacionados con el nombre de la máquina virtual `vm-terraform-1` y con el *flavor* asignado a `vm-terraform-2`. El resultado indica que dos recursos requerían actualización para volver al estado definido originalmente, evidenciando la capacidad de *Terraform* para detectar situaciones de *drift*. Aunque las diferencias observadas mediante el comando `diff` permitieron verificar manualmente los cambios realizados sobre las máquinas virtuales, también fue necesario comprobar si dichas modificaciones podían ser detectadas automáticamente por *Terraform*. Para ello, se utilizó el comando `terraform plan`.


```

[root@controller terraform-openstack]# terraform plan | tee evidencia_tfstate_2/plan_drift.txt
data.openstack_networking_network_v2.read: Reading...
data.openstack_networking_network_v2.read: Read complete after 1s [id=80eaa3e-a89f-4300-b225-4c348b34798]
openstack_compute_instance_v2.instance[0]: Refreshing state... [id=80ba34c-29cc-4875-afc3-36f762ed2c87]
openstack_compute_instance_v2.instance[2]: Refreshing state... [id=a56a2f91-7c96-44da-98b1-b5cd42998f6]
openstack_compute_instance_v2.instance[1]: Refreshing state... [id=693144fe-c6a8-432e-b015-34c1f97ed4b4]

Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  ~ update in-place

Terraform will perform the following actions:

# openstack_compute_instance_v2.instance[0] will be updated in-place
~ resource "openstack_compute_instance_v2" "instancia" {
  id           = "80ba34c-29cc-4875-afc3-36f762ed2c87"
  name         = "UNAM-TELE-FI-MX" -> "vm-terraform-1"
  tags         = {}
  # (18 unchanged attributes hidden)
  # (1 unchanged block hidden)
}

# openstack_compute_instance_v2.instance[1] will be updated in-place
~ resource "openstack_compute_instance_v2" "instancia" {
  flavor_name   = "m1.medium" -> "m1.nano.tiny"
  id           = "693144fe-c6a8-432e-b015-34c1f97ed4b4"
  name         = "vm-terraform-2"
  tags         = {}
  # (17 unchanged attributes hidden)
  # (1 unchanged block hidden)
}

Plan: 0 to add, 2 to change, 0 to destroy.

Note: You didn't use the -out option to save this plan, so Terraform can't

```

Figura 6.30: Detección de diferencias entre la infraestructura real y el archivo `terraform.tfstate` mediante `terraform plan`.

Asimismo, el resumen mostrado por `terraform plan` indica 0 to add, 2 to change, 0 to destroy, confirmando que únicamente dos recursos presentaban diferencias respecto al estado almacenado en el archivo `terraform.tfstate`.

El comando `terraform plan` permitió comparar la configuración definida en el archivo `main.tf`, la información almacenada en el archivo `terraform.tfstate` y el estado real de la infraestructura desplegada en *OpenStack*. Esta comparación permitió identificar automáticamente las modificaciones manuales realizadas durante las pruebas de actualización, demostrando la capacidad de *Terraform* para mantener la consistencia entre la infraestructura declarada y la infraestructura existente. En conjunto, esta prueba permitió comprobar que el archivo `terraform.tfstate` funciona como mecanismo principal de control y sincronización dentro de *Terraform*, lo que permite detectar automáticamente diferencias entre el estado deseado y el estado real de los recursos administrados.

6.1.4.2. Prueba de validación del archivo de estado en *Azure Portal*

La prueba de estado tuvo como objetivo evaluar la capacidad de *Terraform* para mantener sincronizado el archivo `terraform.tfstate` con el estado real de la infraestructura desplegada en *Azure*. Esta prueba permitió verificar el comportamiento del sistema frente a modificaciones manuales realizadas directamente sobre los recursos mediante *Azure CLI*, así como el proceso de reconciliación automática ejecutado posteriormente por *Terraform*. Inicialmente, se almacenaron evidencias correspondientes al estado actual de la infraestructura y del archivo administrado por *Terraform*. Para ello, se ejecutaron los siguientes comandos:

```
mkdir -p evidencia_tfstate_azure

terraform state list \
> evidencia_tfstate_azure/state_list_inicial.txt

terraform show \
> evidencia_tfstate_azure/show_inicial.txt

az vm list --show-details -o table \
> evidencia_tfstate_azure/vms_azure_inicial.txt

terraform plan -var="num_vms=3" \
> evidencia_tfstate_azure/plan_inicial.txt
```

Después, se verificó que *Terraform* no detectara diferencias entre el estado real de la infraestructura y el contenido del archivo `terraform.tfstate`. Para ello, se ejecutó el siguiente comando:

```
terraform plan -var="num_vms=3" \
> evidencia_tfstate_azure/plan_inicial_sin_cambios.txt
```

A continuación, se realizó una modificación manual directamente sobre una máquina virtual utilizando *Azure CLI*. En este caso, se agregó una etiqueta personalizada para generar un cambio externo al archivo administrado por *Terraform*:

```
az vm update \
-g rg-terraform-azure-test \
-n vm-terraform-azure-test-1 \
--set tags.PruebaTfstate=CambioExterno
```

Más adelante, se ejecutó nuevamente *Terraform* con el objetivo de reconciliar el estado de la infraestructura y eliminar las diferencias detectadas:

```
terraform apply -var="num_vms=3" -auto-approve
```

Las figuras 6.31 y 6.32 muestran cómo *Terraform* detectó automáticamente las modificaciones realizadas manualmente sobre las etiquetas de las máquinas virtuales y restauró el estado esperado definido dentro de la configuración original.

```
(root@controller terraform-azure)# terraform apply -var="num_vms=3" -auto-approve
terraform apply -var="num_vms=3" -auto-approve

azurerm_resource_group.rg: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test]
azurerm_virtual_network.vnet: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Network/virtualNetworks/vnet-terraform-test]
azurerm_subnet.subnet: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Network/virtualNetworks/vnet-terraform-test/subnets/subnet-terraform-test]
azurerm_network_interface.nic[0]: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Network/networkInterfaces/vm-terraform-azure-test-1-nic]
azurerm_network_interface.nic[1]: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Network/networkInterfaces/vm-terraform-azure-test-2-nic]
azurerm_network_interface.nic[2]: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Network/networkInterfaces/vm-terraform-azure-test-3-nic]
azurerm_linux_virtual_machine.vm[0]: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-1]
azurerm_linux_virtual_machine.vm[2]: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-3]
azurerm_linux_virtual_machine.vm[1]: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-2]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
- update in-place

Terraform will perform the following actions:

# azurerm_linux_virtual_machine.vm[0] will be updated in-place
- resource "azurerm_linux_virtual_machine" "vm" {
  id = "/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-1"
  name = "vm-terraform-azure-test-1"
  tags = {
    "NombrePrueba" = "UNAM-FI-TELE-MX" -> null
    "PruebaEstado" = "CambioEstado" -> null
  }
  # (41 unchanged attributes hidden)
  # (2 unchanged blocks hidden)
}

# azurerm_linux_virtual_machine.vm[1] will be updated in-place
- resource "azurerm_linux_virtual_machine" "vm" {
  id = "/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-2"
  name = "vm-terraform-azure-test-2"
  tags = {
    "NombrePrueba" = "UNAM-FI-MX" -> null
  }
  # (41 unchanged attributes hidden)
  # (2 unchanged blocks hidden)
}

# azurerm_linux_virtual_machine.vm[2] will be updated in-place
- resource "azurerm_linux_virtual_machine" "vm" {
  id = "/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-3"
  name = "vm-terraform-azure-test-3"
  tags = {
    "NombrePrueba" = "UNAM-FI-CLOUD" -> null
  }
  # (41 unchanged attributes hidden)
  # (2 unchanged blocks hidden)
}
```

Figura 6.31: Proceso de reconciliación del archivo `terraform.tfstate` en *Azure*.

En la figura 6.31 puede observarse que *Terraform* detectó automáticamente diferencias entre el estado real de las máquinas virtuales y la configuración almacenada en el archivo `terraform.tfstate`. En particular, se identificaron etiquetas (*tags*) agregadas manualmente mediante *Azure CLI*, las cuales no formaban parte de la configuración declarada originalmente. Como resultado, *Terraform* generó un plan de ejecución para restaurar el estado esperado de la infraestructura.

```
}

Plan: 0 to add, 3 to change, 0 to destroy.
azurerm_linux_virtual_machine.vm[2]: Modifying... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-3]
azurerm_linux_virtual_machine.vm[0]: Modifying... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-1]
azurerm_linux_virtual_machine.vm[1]: Modifying... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-2]
azurerm_linux_virtual_machine.vm[2]: Still modifying... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-3, 10s elapsed]
azurerm_linux_virtual_machine.vm[0]: Still modifying... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-1, 10s elapsed]
azurerm_linux_virtual_machine.vm[1]: Still modifying... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-2, 10s elapsed]
azurerm_linux_virtual_machine.vm[2]: Modifications complete after 12s [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-3]
azurerm_linux_virtual_machine.vm[1]: Modifications complete after 12s [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-2]
azurerm_linux_virtual_machine.vm[0]: Modifications complete after 12s [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-1]

Apply complete! Resources: 0 added, 3 changed, 0 destroyed.

Outputs:
vm_ids = [
  "/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-1",
  "/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-2",
  "/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-3",
]
```

Figura 6.32: Proceso de reconciliación del archivo `terraform.tfstate` en *Azure*.

En la figura 6.32 se muestra la ejecución del proceso de reconciliación. El resumen indica 0 added, 3 changed, 0 destroyed, confirmando que *Terraform* modificó tres recursos existentes sin crear ni eliminar máquinas virtuales. De esta manera, las etiquetas agregadas manualmente fueron eliminadas y la infraestructura volvió a coincidir con la configuración definida en los archivos administrados por la herramienta. Finalmente, se verificó que no existieran diferencias entre el

estado real de la infraestructura y el archivo administrado por *Terraform*. Para ello, se ejecutaron los siguientes comandos:

```
terraform plan -var="num_vms=3" \  
> evidencia_tfstate_azure/plan_sin_cambios_final.txt  
  
terraform state list \  
> evidencia_tfstate_azure/state_list_final.txt  
  
terraform show \  
> evidencia_tfstate_azure/show_final.txt
```

Una vez finalizado el proceso de reconciliación, se verificó nuevamente el estado de la infraestructura con el propósito de confirmar que no existieran diferencias pendientes entre la configuración declarada y los recursos desplegados en *Azure*. A partir de los resultados obtenidos, fue posible comprobar que *Terraform* fue capaz de detectar automáticamente cambios externos realizados directamente sobre los recursos desplegados en *Azure*. Asimismo, las pruebas demostraron que el archivo `terraform.tfstate` permitió mantener sincronizada la infraestructura mediante procesos de reconciliación automática, restaurando el estado esperado definido dentro de los archivos de configuración.

6.1.5. Prueba de *rollback* mediante control de versiones

6.1.5.1. Prueba de *rollback* en *OpenStack*

La prueba de *rollback* tuvo como objetivo validar la capacidad de *Terraform* para restaurar automáticamente la infraestructura hacia el estado definido dentro de los archivos de configuración `.tf`. Para ello, se utilizaron previamente las modificaciones realizadas durante las pruebas de actualización de infraestructura, donde se alteraron manualmente nombres y *flavors* de distintas máquinas virtuales dentro del entorno *OpenStack*.

Inicialmente, se ejecutó el comando `terraform apply -auto-approve` con el propósito de restaurar automáticamente la configuración definida dentro del archivo `main.tf`. Durante este proceso, *Terraform* detectó diferencias relacionadas con el nombre de la máquina virtual `vm-terraform-1` y con el *flavor* asignado a `vm-terraform-2`.

```
terraform apply --auto-approve
```

En la figura 6.33 puede observarse que *Terraform* detectó dos diferencias respecto al estado definido en el archivo `main.tf`. La primera corresponde al nombre de la máquina virtual `vm-terraform-1`, mientras que la segunda se relaciona con el *flavor* asignado a `vm-terraform-2`. Durante el proceso de aplicación, *Terraform* intentó restaurar ambos recursos al estado originalmente definido. Sin embargo, la operación asociada al cambio de *flavor* generó un error debido a restricciones internas del proceso de *resize* en *OpenStack*.

```

[root@controller terraform-openstack]# terraform apply -auto-approve
data.openstack_networking_network_v2.read: Reading...
data.openstack_networking_network_v2.read: Read complete after 1s [id=86eaa5a5-489f-4380-bd28-4a14eb34788]
openstack_compute_instance_v2.instance[2]: Refreshing state... [id=a8a2f93-7c96-44de-98d1-b6cd4298b6f6]
openstack_compute_instance_v2.instance[1]: Refreshing state... [id=693144fe-cda8-432e-b815-34c1f87ed4b4]
openstack_compute_instance_v2.instance[0]: Refreshing state... [id=88bba34c-29cc-4875-afc3-36f762ed2c87]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
- update in-place

Terraform will perform the following actions:

# openstack_compute_instance_v2.instance[0] will be updated in-place
- resource "openstack_compute_instance_v2" "instance" {
  id           = "88bba34c-29cc-4875-afc3-36f762ed2c87"
  name         = "VM-TERRAFORM-1"
  tags         = []
  # (18 unchanged attributes hidden)
}
# (1 unchanged block hidden)

# openstack_compute_instance_v2.instance[1] will be updated in-place
- resource "openstack_compute_instance_v2" "instance" {
  flavor_name   = "m1.medium" -> "m1.nano.tiny"
  id           = "693144fe-cda8-432e-b815-34c1f87ed4b4"
  name         = "vm-terraform-2"
  tags         = []
  # (17 unchanged attributes hidden)
}
# (1 unchanged block hidden)

Plan: 0 to add, 2 to change, 0 to destroy.
openstack_compute_instance_v2.instance[1]: Modifying... [id=693144fe-cda8-432e-b815-34c1f87ed4b4]
openstack_compute_instance_v2.instance[0]: Modifying... [id=88bba34c-29cc-4875-afc3-36f762ed2c87]
openstack_compute_instance_v2.instance[0]: Modifications complete after 6s [id=88bba34c-29cc-4875-afc3-36f762ed2c87]
openstack_compute_instance_v2.instance[1]: Still modifying... [id=693144fe-cda8-432e-b815-34c1f87ed4b4, 18s elapsed]
openstack_compute_instance_v2.instance[1]: Still modifying... [id=693144fe-cda8-432e-b815-34c1f87ed4b4, 28s elapsed]

Error: Error waiting for instance (693144fe-cda8-432e-b815-34c1f87ed4b4) to resize: unexpected state 'ACTIVE', wanted target 'VERIFY_RESIZE'. last error: Nis(enlis)
with openstack_compute_instance_v2.instance[1],
on main.tf line 33, in resource "openstack_compute_instance_v2" "instance":
33: resource "openstack_compute_instance_v2" "instance" {

```

Figura 6.33: Ejecución de `terraform apply -auto-approve` durante el proceso de *rollback*.

Posteriormente, se ejecutó nuevamente el comando `terraform plan` para validar el estado final de la infraestructura y detectar posibles diferencias restantes entre el entorno real y el archivo `terraform.tfstate`.

```

terraform plan | tee \
evidencia_tfstate_2/plan_final.txt

```

En la figura 6.34 se observa que, después de ejecutar el proceso de restauración, únicamente permanecía una diferencia pendiente dentro de la infraestructura. El resumen `0 to add, 1 to change, 0 to destroy` indica que el nombre de la máquina virtual había sido restaurado correctamente, mientras que el cambio de *flavor* aún requería intervención debido a las limitaciones del proceso de reversión.

```

data.openstack_networking_network_v2.read: Reading... [id=a8a2f93-7c96-44de-98d1-b6cd4298b6f6]
data.openstack_networking_network_v2.read: Read complete after 9s [id=86eaa5a5-489f-4380-bd28-4a14eb34788]
openstack_compute_instance_v2.instance[0]: Refreshing state... [id=88bba34c-29cc-4875-afc3-36f762ed2c87]
openstack_compute_instance_v2.instance[1]: Refreshing state... [id=693144fe-cda8-432e-b815-34c1f87ed4b4]
openstack_compute_instance_v2.instance[2]: Refreshing state... [id=a8a2f93-7c96-44de-98d1-b6cd4298b6f6]

Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
- update in-place

Terraform will perform the following actions:

# openstack_compute_instance_v2.instance[1] will be updated in-place
- resource "openstack_compute_instance_v2" "instance" {
  flavor_name   = "m1.medium" -> "m1.nano.tiny"
  id           = "693144fe-cda8-432e-b815-34c1f87ed4b4"
  name         = "vm-terraform-2"
  tags         = []
  # (17 unchanged attributes hidden)
}
# (1 unchanged block hidden)

Plan: 0 to add, 1 to change, 0 to destroy.

Note: You didn't use the -out option to save this plan, so Terraform can't
guarantee to take exactly these actions if you run "terraform apply" now.

```

Figura 6.34: Validación final del estado de la infraestructura mediante `terraform plan`.

Después, se consultaron los eventos asociados a cada máquina virtual mediante los siguientes comandos:

```

openstack server event list vm-terraform-1

openstack server event list vm-terraform-2

openstack server event list vm-terraform-3

```

En la figura 6.35 se muestran los eventos registrados por *OpenStack* para las tres máquinas virtuales analizadas. Estos eventos permiten reconstruir cronológicamente las operaciones ejecutadas sobre cada instancia, incluyendo procesos de creación, redimensionamiento (*resize*) y confirmación de cambios.

```
[root@controller terraform-openstack]# openstack server event list vm-terraform-1
openstack server event list vm-terraform-2
openstack server event list vm-terraform-3
```

Request ID	Server ID	Action	Start Time
req-c74fd671-82d0-4620-8922-b79d5bcd06ec	80bba34c-29cc-4075-a	create	2026-05-28T19:53:16.000000

Request ID	Server ID	Action	Start Time
req-c8924cb5-cef6-4122-bed6-7fb99e051d34	693144fe-c6a8-432e-b015-34c1f07ed404	resize	2026-05-28T20:07:11.000000
req-168700e0-9822-48e3-896b-6ad4de1c84b2	693144fe-c6a8-432e-b015-34c1f07ed404	confirmResize	2026-05-28T20:01:08.000000
req-dd48f2c7-c557-4472-8b0c-7f6a7bced774	693144fe-c6a8-432e-b015-34c1f07ed404	resize	2026-05-28T20:00:29.000000
req-866cfbea-54c1-48e2-ba3b-c8e23736b36b	693144fe-c6a8-432e-b015-34c1f07ed404	create	2026-05-28T19:53:16.000000

Request ID	Server ID	Action	Start Time
req-6061d463-d652-4923-b301-609eca0ef0d1	a56a2f91-7c96-44da-98b1-b5cd42998bfe	resize	2026-05-28T20:05:30.000000
req-87acc4d5-7067-4807-a824-c93009c15d7	a56a2f91-7c96-44da-98b1-b5cd42998bfe	resize	2026-05-28T20:01:22.000000
req-d0d4f2b0-0ccc-4664-8fec-e00f9097d77	a56a2f91-7c96-44da-98b1-b5cd42998bfe	create	2026-05-28T19:53:16.000000

Figura 6.35: Eventos registrados durante las operaciones de actualización y *rollback*.

Posteriormente, se verificó el estado actual de las máquinas virtuales utilizando los siguientes comandos:

```
openstack server show vm-terraform-1 \
-c name -c flavor -c status

openstack server show vm-terraform-2 \
-c name -c flavor -c status

openstack server show vm-terraform-3 \
-c name -c flavor -c status
```

En la figura 6.36 puede verificarse el estado final de las máquinas virtuales después del proceso de restauración. Todas las instancias permanecieron en estado **ACTIVE**; sin embargo, la máquina virtual *vm-terraform-2* conservó el *flavor* *m1.medium*, evidenciando que la reversión automática del redimensionamiento no pudo completarse exitosamente.

```
[root@controller terraform-openstack]# openstack server show vm-terraform-1 -c name -c flavor -c status
openstack server show vm-terraform-2 -c name -c flavor -c status
openstack server show vm-terraform-3 -c name -c flavor -c status
```

Field	Value
flavor	description=, disk='1', ephemeral='0', , id='m1.nano', is_disabled=, is_public='True', location=, name='m1.nano', original_name='m1.nano', ram='256', rxtx_factor=, swap='0', vcpus='1'
name	vm-terraform-1
status	ACTIVE

Field	Value
flavor	description=, disk='40', ephemeral='0', , id='m1.medium', is_disabled=, is_public='True', location=, name='m1.medium', original_name='m1.medium', ram='4096', rxtx_factor=, swap='0', vcpus='2'
name	vm-terraform-2
status	ACTIVE

Field	Value
flavor	description=, disk='1', ephemeral='0', , id='m1.nano', is_disabled=, is_public='True', location=, name='m1.nano', original_name='m1.nano', ram='256', rxtx_factor=, swap='0', vcpus='1'
name	vm-terraform-3
status	ACTIVE

Figura 6.36: Estado final de las máquinas virtuales después del proceso de *rollback*.

Con el propósito de analizar el comportamiento interno del proceso de *resize*, se consultó el historial de migraciones asociado a la máquina virtual **vm-terraform-2** mediante el siguiente comando:

```
openstack server migration list \
--server vm-terraform-2
```

En la figura 6.37 se presenta el historial de migraciones asociado a **vm-terraform-2**. El registro muestra tanto una operación de *resize* confirmada como un intento posterior de restauración que finalizó con error. Esta información permitió identificar que la diferencia pendiente detectada por *Terraform* estaba directamente relacionada con el proceso de redimensionamiento de la instancia.

Id	UUID	Source Node	Dest Node	Source Compute	Dest Compute	Dest Host	Status	Server UUID	Old Flavor	New Flavor	Type	Created At	Updated At
22	9e0f65e3-93e0-4fb7-819e-44ad6d9c8ccd	compute	compute	compute	compute	192.168.1.0.28	error	693144fe-c6a8-432e-b015-34c1f07ed404	3	6	resize	2026-05-28T20:07:11.000000	2026-05-28T20:07:26.000000
19	100d69d1-3143-419f-8d3d-f08adf8a799a	compute	compute	compute	compute	192.168.1.0.28	confirmed	693144fe-c6a8-432e-b015-34c1f07ed404	6	3	resize	2026-05-28T20:00:32.000000	2026-05-28T20:01:16.000000

Figura 6.37: Historial de migraciones y operaciones de *resize* sobre **vm-terraform-2**.

Posteriormente, se consultaron nuevamente los eventos asociados a **vm-terraform-2** para analizar el detalle de las operaciones realizadas durante el proceso de actualización y restauración.

```
openstack server event list vm-terraform-2
```

En la figura 6.38 se observan los eventos registrados para **vm-terraform-2**. La secuencia de eventos evidencia la ejecución de operaciones de *resize* y *confirmResize*, proporcionando información adicional para el análisis del proceso de restauración.

```
[root@controller terraform-openstack]# openstack server event list vm-terraform-2
```

Request ID	Server ID	Action	Start Time
req-c8924cb5-cef6-4122-bed6-7fb99e051d34	693144fe-c6a8-432e-b015-34c1f07ed404	resize	2026-05-28T20:07:11.000000
req-168700e0-9822-48e3-896b-6ad4de1c84b2	693144fe-c6a8-432e-b015-34c1f07ed404	confirmResize	2026-05-28T20:01:08.000000
req-dd48f2c7-c557-4472-8b0c-7f6a7bced774	693144fe-c6a8-432e-b015-34c1f07ed404	resize	2026-05-28T20:00:29.000000
req-866cfbea-54c1-48e2-ba3b-c8e23736b36b	693144fe-c6a8-432e-b015-34c1f07ed404	create	2026-05-28T19:53:16.000000

Figura 6.38: Eventos registrados para la máquina virtual `vm-terraform-2`.

Después, se inspeccionó el evento asociado al error generado durante el proceso de *resize* mediante el siguiente comando:

```
openstack server event show vm-terraform-2 \
req-c8924cb5-cef6-4122-bed6-7fb99e051d34
```

En la figura 6.39 se muestra el detalle del error generado durante el intento de reversión del *resize*. El mensaje indica que *OpenStack* no pudo completar el redimensionamiento solicitado debido a restricciones asociadas con la reducción de recursos previamente asignados a la instancia.

```
events details='Error de redimensionamiento: Incapaz de reducir el tamaño del disco.', event='compute_resize_instance', finish_time='2026-05-28T20:07:38.000000', host='compute', host_id='16124831a12999afa90b8e9fad62b1348c75d38f9eb33bccb66adbb', result='Error', start_time='2026-05-28T20:07:20.000000', traceback='File "/usr/lib/python3.9/site-packages/nova/compute/utils.py", line 1453, in decorated_function\n    return function(self, context, *args, **kwargs)\nFile "/usr/lib/python3.9/site-packages/nova/compute/manager.py", line 214, in decorated_function\n    compute_utils.add_instance_fault_from_exc(context, self.force_reraise())\nFile "/usr/lib/python3.9/site-packages/oslo_utils/excutils.py", line 227, in __exit__\n    self.force_reraise()\nFile "/usr/lib/python3.9/site-packages/oslo_utils/excutils.py", line 200, in force_reraise\n    raise self.value\nFile "/usr/lib/python3.9/site-packages/nova/compute/manager.py", line 203, in decorated_function\n    return function(self, context, *args, **kwargs)\nFile "/usr/lib/python3.9/site-packages/nova/compute/manager.py", line 6072, in resize_instance\n    self._revert_allocation(context, instance, migration)\nFile "/usr/lib/python3.9/site-packages/oslo_utils/excutils.py", line 227, in __exit__\n    self.force_reraise()\nFile "/usr/lib/python3.9/site-packages/oslo_utils/excutils.py", line 200, in force_reraise\n    raise self.value\nFile "/usr/lib/python3.9/site-packages/nova/compute/manager.py", line 6068, in resize_instance\n    self._resize_instance(context, instance, image, migration,\nFile "/usr/lib/python3.9/site-packages/nova/compute/manager.py", line 6139, in _resize_instance\n    self.compute_rpcapi.finish_resize(context, instance,\nFile "/usr/lib64/python3.9/contextlib.py", line 137, in __exit__\n    self.gen.throw(typ, value, traceback)\nFile "/usr/lib/python3.9/site-packages/nova/compute/manager.py", line 10878, in _error_out_instance_on_exception\n    raise error.inner_exception\n\n    details='compute_prep_resize',\n    finish_time='2026-05-28T20:07:20.000000', host='compute', host_id='16124831a12999afa90b8e9fad62b1348c75d38f9eb33bccb66adbb',\n    result='Success', start_time='2026-05-28T20:07:15.000000',\n    traceback=''\n    details='cold_migrate',\n    finish_time='2026-05-28T20:07:15.000000', host='controller', host_id='8b1ee66dabdad3609bbbc132e65f2fe1c7da98c8215e6bb65ca3a983',\n    result='Success', start_time='2026-05-28T20:07:11.000000',\n    traceback=''\n    details='conductor_migrate_server',\n    finish_time='2026-05-28T20:07:16.000000', host='controller', host_id='8b1ee66dabdad3609bbbc132e65f2fe1c7da98c8215e6bb65ca3a983',\n    result='Success', start_time='2026-05-28T20:07:11.000000',\n    traceback=''\n\nid req-c8924cb5-cef6-4122-bed6-7fb99e051d34
```

Figura 6.39: Detalle del error generado durante el proceso de *resize*.

Finalmente, se verificó el evento asociado a una operación exitosa de *resize* utilizando el siguiente comando:

```
openstack server event show vm-terraform-2 \
req-dd48f2c7-c557-4472-8b0c-7f6a7bced774
```


En la figura 6.40 puede observarse el detalle de una operación de *resize* ejecutada exitosamente. Este registro permitió contrastar el comportamiento de una operación correcta frente al evento que generó el error durante el intento de restauración, facilitando la identificación de la causa de la diferencia detectada por *Terraform*.

```
[root@controller terraform-openstack]# openstack server event show vm-terraform-2 req-dd48f2c7-c557-4472-8b8c-7f6a7bcd774
openstack server event show vm-terraform-2 req-dd48f2c7-c557-4472-8b8c-7f6a7bcd774
```

Field	Value
action	resize
events	<pre> details=, event='compute_finish_resize', finish_time='2026-05-28T20:01:01.000000', host='compute', host_ id='16124831a12599afa90b8e9fad62b134c75d38f9eb33bccb66addb', result='Success', start_time='2026-05-28T20:00:51.000000', traceback= details=, event='compute_resize_instance', finish_time='2026-05-28T20:00:51.000000', host='compute', host_ id='16124831a12599afa90b8e9fad62b134c75d38f9eb33bccb66addb', result='Success', start_time='2026-05-28T20:00:40.000000', traceback= details=, event='compute_prep_resize', finish_time='2026-05-28T20:00:40.000000', host='compute', host_ id='16124831a12599afa90b8e9fad62b134c75d38f9eb33bccb66addb', result='Success', start_time='2026-05-28T20:00:35.000000', traceback= details=, event='cold_migrate', finish_time='2026-05-28T20:00:35.000000', host='controller', ho st_id='0b1ee6dabdad3609bb132e65f2fe11c7da98c8215e6bb65ca3a983', result='Success', start_time='2026-05-28T20:00:30.000000', traceback= details=, event='conductor_migrate_server', finish_time='2026-05-28T20:00:35.000000', host='controller', ho st_id='0b1ee6dabdad3609bb132e65f2fe11c7da98c8215e6bb65ca3a983', result='Success', start_time='2026-05-28T20:00:30.000000', traceback= </pre>
id	req-dd48f2c7-c557-4472-8b8c-7f6a7bcd774
message	None
project_id	5b9dfeb866ad42818d9b701feddaa7b
request_id	req-dd48f2c7-c557-4472-8b8c-7f6a7bcd774
start_time	2026-05-28T20:00:29.000000
user_id	0733b232b0174f2cddf28b793c0ab309

Figura 6.40: Detalle de una operación exitosa de *resize* sobre *vm-terraform-2*.

Durante el proceso de restauración, *Terraform* logró corregir automáticamente el nombre de la máquina virtual *vm-terraform-1*; sin embargo, no fue posible revertir automáticamente el cambio de *flavor* aplicado sobre *vm-terraform-2*. Esto ocurrió debido a que *OpenStack* no permite reducir automáticamente ciertos recursos asociados a una máquina virtual una vez que el proceso de *resize* ha sido confirmado.

En particular, después de confirmar una operación de redimensionamiento hacia un *flavor* superior, el entorno conserva la nueva asignación de recursos físicos, almacenamiento y configuración interna de la instancia. Debido a ello, intentar regresar automáticamente hacia un *flavor* menor puede generar inconsistencias relacionadas con discos virtuales, particiones, asignación de memoria y recursos asociados al nodo *compute*. Como consecuencia, *Terraform* detecta la diferencia dentro del archivo *terraform.tfstate*, pero depende de las capacidades internas de *OpenStack* para completar correctamente el proceso de reversión.

En conjunto, la prueba permitió demostrar que *Terraform* puede detectar modificaciones manuales realizadas fuera de la herramienta y restaurar parcialmente el estado original de la infraestructura utilizando la información almacenada dentro del archivo *terraform.tfstate*. Por lo tanto, el proceso de *rollback* no depende únicamente de las capacidades de *Terraform*, sino también de las políticas y limitaciones implementadas por la plataforma de virtualización subyacente. Este comportamiento resulta especialmente relevante en operaciones de escalabilidad vertical, donde la reversión de recursos puede no estar completamente soportada una vez confirmados ciertos cambios en la infraestructura.

6.1.5.2. Prueba de *rollback* en *Azure Portal*

La prueba de *rollback* en *Azure* tuvo como objetivo comprobar la capacidad de *Terraform* para restaurar automáticamente la configuración original de la infraestructura después de realizar cambios manuales sobre las máquinas virtuales. Esta prueba permitió verificar cómo el archivo *terraform.tfstate* puede utilizarse para regresar la infraestructura a la configuración previamente definida.

Inicialmente, se creó un directorio para almacenar las evidencias relacionadas con la prueba. Además, se guardó información de las máquinas virtuales junto con las etiquetas modificadas manualmente.

```
mkdir -p evidencia_rollback_azure

az vm list \
  -g rg-terraform-azure-test \
  --query "[].{Nombre:name,TagNombre:tags.NombrePrueba}" \
  -o table > evidencia_rollback_azure/01_nombres_actuales.txt

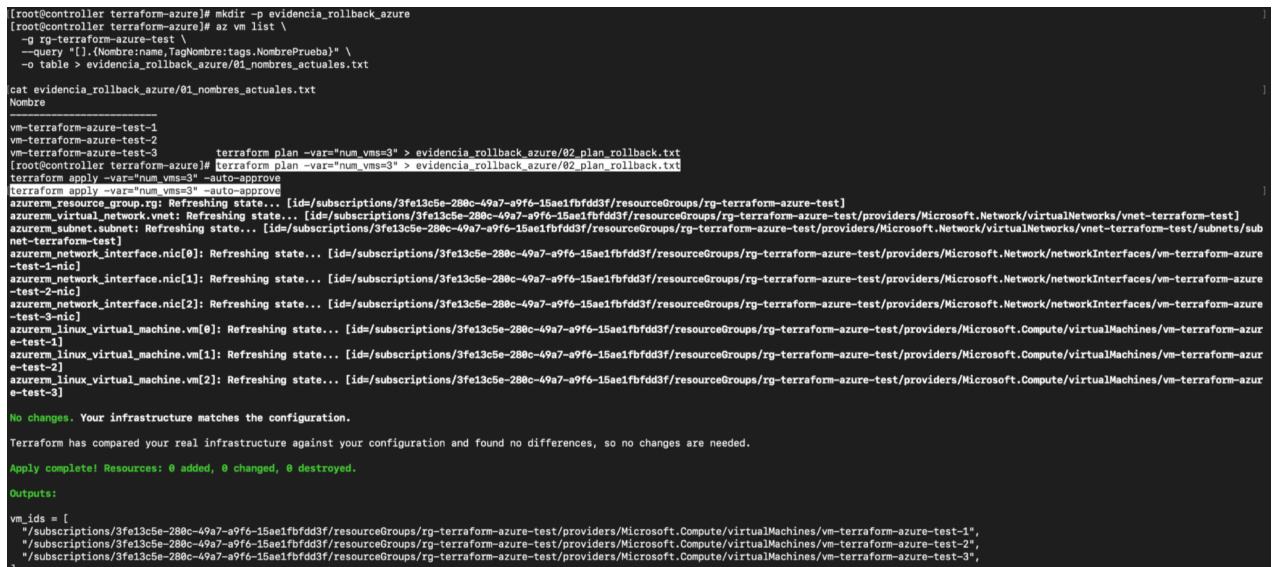
cat evidencia_rollback_azure/01_nombres_actuales.txt
```

A continuación, se ejecutó una validación mediante `terraform plan` y posteriormente se aplicó el proceso de restauración utilizando `terraform apply`.

```
terraform plan -var="num_vms=3" \
> evidencia_rollback_azure/02_plan_rollback.txt

terraform apply -var="num_vms=3" -auto-approve
```

En la figura 6.41 puede observarse que *Terraform* comparó el estado actual de la infraestructura con la configuración definida en los archivos `.tf`. El resultado indica que no existían diferencias pendientes entre la infraestructura desplegada y el estado esperado, por lo que no fue necesario realizar cambios adicionales. Esto confirma que el proceso de restauración se había completado correctamente y que la infraestructura se encontraba sincronizada con la información almacenada en el archivo `terraform.tfstate`.



```
[root@controller terraform-azure]# mkdir -p evidencia_rollback_azure
[root@controller terraform-azure]# az vm list \
  -g rg-terraform-azure-test \
  --query "[].{Nombre:name,TagNombre:tags.NombrePrueba}" \
  -o table > evidencia_rollback_azure/01_nombres_actuales.txt

cat evidencia_rollback_azure/01_nombres_actuales.txt
Nombre
vm-terraform-azure-test-1
vm-terraform-azure-test-2
vm-terraform-azure-test-3
[terraform plan -var="num_vms=3" > evidencia_rollback_azure/02_plan_rollback.txt]
[terraform plan -var="num_vms=3" > evidencia_rollback_azure/02_plan_rollback.txt]
[terraform apply -var="num_vms=3" -auto-approve]
terraform apply -var="num_vms=3" -auto-approve
azure_resource_group: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test]
azurerm_virtual_network.vnet: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Network/virtualNetworks/vnet-terraform-test]
azurerm_subnet.subnet: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Network/virtualNetworks/vnet-terraform-test/subnets/subnet-terraform-test]
azurerm_network_interface.nic[0]: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Network/networkInterfaces/vm-terraform-azure-test-1-nic]
azurerm_network_interface.nic[1]: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Network/networkInterfaces/vm-terraform-azure-test-2-nic]
azurerm_network_interface.nic[2]: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Network/networkInterfaces/vm-terraform-azure-test-3-nic]
azurerm_linux_virtual_machine.vm[0]: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-1]
azurerm_linux_virtual_machine.vm[1]: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-2]
azurerm_linux_virtual_machine.vm[2]: Refreshing state... [id=/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-3]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:
vm_ids = [
  "/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-1",
  "/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-2",
  "/subscriptions/3fe13c5e-288c-49a7-a9f6-15ae1fbfdd3f/resourceGroups/rg-terraform-azure-test/providers/Microsoft.Compute/virtualMachines/vm-terraform-azure-test-3",
]
```

Figura 6.41: Proceso de restauración de infraestructura mediante *Terraform* en *Azure*.

Una vez finalizado el proceso de restauración, se almacenó nuevamente la información de las máquinas virtuales para verificar que las etiquetas modificadas habían sido eliminadas correctamente.

```
az vm list \
  -g rg-terraform-azure-test \
  --query "[].{Nombre:name,TagNombre:tags.NombrePrueba}" \
  -o table > evidencia_rollback_azure/03_nombres_restaurados.txt

terraform plan -var="num_vms=3" \
> evidencia_rollback_azure/04_plan_final.txt
```

Finalmente, se mostró la secuencia completa del proceso de *rollback*, comparando la configuración antes y después de la restauración.

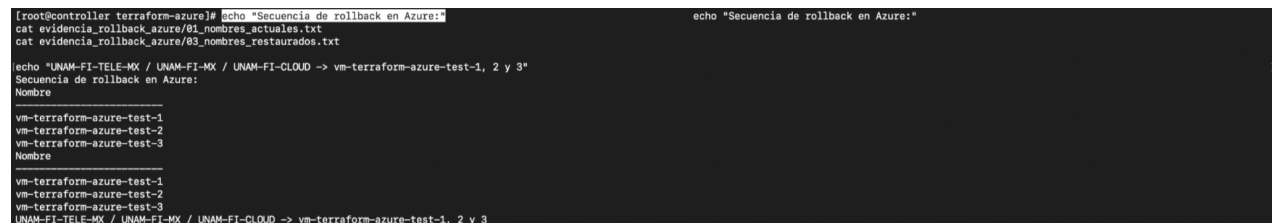
```
echo "Secuencia de rollback en Azure:"

cat evidencia_rollback_azure/01_nombres_actuales.txt

cat evidencia_rollback_azure/03_nombres_restaurados.txt

echo "UNAM-FI-TELE-MX / UNAM-FI-MX / UNAM-FI-CLOUD \
-> vm-terraform-azure-test-1, 2 y 3"
```

En la figura 6.42 se muestra la comparación entre el estado previo y posterior al proceso de *rollback*. Inicialmente, las máquinas virtuales contenían las etiquetas UNAM-FI-TELE-MX, UNAM-FI-MX y UNAM-FI-CLOUD, incorporadas manualmente durante las pruebas anteriores. Después de ejecutar el proceso de restauración, dichas etiquetas fueron eliminadas y la infraestructura regresó al estado definido originalmente dentro de los archivos de configuración administrados por *Terraform*.



```
[root@controller terraform-azure]# echo "Secuencia de rollback en Azure:"
echo "Secuencia de rollback en Azure:"
cat evidencia_rollback_azure/01_nombres_actuales.txt
cat evidencia_rollback_azure/03_nombres_restaurados.txt

echo "UNAM-FI-TELE-MX / UNAM-FI-MX / UNAM-FI-CLOUD -> vm-terraform-azure-test-1, 2 y 3"
Secuencia de rollback en Azure:
Nombre
vm-terraform-azure-test-1
vm-terraform-azure-test-2
vm-terraform-azure-test-3
Nombre
vm-terraform-azure-test-1
vm-terraform-azure-test-2
vm-terraform-azure-test-3
UNAM-FI-TELE-MX / UNAM-FI-MX / UNAM-FI-CLOUD -> vm-terraform-azure-test-1, 2 y 3
```

Figura 6.42: Comparación antes y después del proceso de *rollback* en *Azure*.

Los resultados obtenidos permitieron comprobar que *Terraform* fue capaz de detectar automáticamente las diferencias existentes entre el entorno actual y la configuración definida originalmente, restaurando los valores esperados mediante el archivo `terraform.tfstate`.

Asimismo, a diferencia de lo observado durante la prueba de *rollback* en *OpenStack*, donde existieron limitaciones asociadas al proceso de *resize*, en *Azure* la restauración se completó exitosamente debido a que las modificaciones realizadas afectaban únicamente metadatos de configuración (*tags*) y no implicaban cambios sobre recursos físicos o características de hardware de las máquinas virtuales.

6.1.6. Comparación entre *OpenStack* y *Microsoft Azure Portal*

Con el propósito de comparar el comportamiento de una nube privada y una nube pública utilizando *Terraform*, se realizaron pruebas equivalentes tanto en *OpenStack* como en *Microsoft*

Azure. Las pruebas se enfocaron en operaciones de creación y destrucción de máquinas virtuales bajo distintos niveles de paralelismo, con el objetivo de analizar tiempos de ejecución, capacidad de aprovisionamiento, estabilidad del entorno y limitaciones operativas presentes en cada plataforma.

La tabla 6.5 muestra los resultados obtenidos durante las pruebas de creación de máquinas virtuales en ambos entornos. En estas pruebas se evaluó el comportamiento de *Terraform* al incrementar gradualmente el número de instancias y el nivel de paralelismo utilizado durante el aprovisionamiento.

Prueba	OpenStack (s)	VMs creadas	Azure (s)	VMs creadas
Crear 1 VM	35.603	1/1	49.000	1/1
Crear 5 VM / P1	166.865	5/5	223.277	5/5
Crear 5 VM / P5	55.185	5/5	164.675	5/5
Crear 10 VM / P1	274.547	10/10	256.593	5/10
Crear 10 VM / P5	115.869	10/10	No se realizaron pruebas	–
Crear 10 VM / P10	112.824	10/10	No se realizaron pruebas	–
Crear 15 VM / P1	478.721	15/15	No se realizaron pruebas	–
Crear 15 VM / P5	182.736	15/15	No se realizaron pruebas	–
Crear 15 VM / P15	226.576	15/15	No se realizaron pruebas	–
Crear 30 VM / P1	908.354	24/30	No se realizaron pruebas	–
Crear 30 VM / P5	299.091	24/30	No se realizaron pruebas	–
Crear 30 VM / P10	298.091	24/30	No se realizaron pruebas	–
Crear 30 VM / P15	284.184	24/30	No se realizaron pruebas	–
Crear 30 VM / P30	425.281	24/30	No se realizaron pruebas	–

Tabla 6.5: Comparación de pruebas de creación entre *OpenStack* y *Microsoft Azure* en segundos.

A partir de los resultados obtenidos, fue posible observar que *OpenStack* presentó menores tiempos de aprovisionamiento en la mayoría de los escenarios evaluados. Por ejemplo, durante la creación de cinco máquinas virtuales utilizando un paralelismo de cinco procesos, *OpenStack* logró completar el despliegue en aproximadamente 55 segundos, mientras que *Microsoft Azure* requirió cerca de 2 minutos con 44 segundos.

Asimismo, conforme aumentó el número de máquinas virtuales y el nivel de paralelismo, se identificó que *OpenStack* permitió un mayor control sobre el comportamiento interno de la infraestructura, aunque también comenzaron a observarse limitaciones relacionadas con la capacidad física disponible dentro del laboratorio, principalmente asociadas al consumo de memoria *RAM*, almacenamiento y recursos de procesamiento.

En el caso de *Microsoft Azure*, las principales limitaciones estuvieron relacionadas con las cuotas regionales de *vCPU* y las restricciones de la suscripción utilizada. Debido a ello, únicamente fue posible mantener activas cinco máquinas virtuales del tipo **Standard_D2s_v3** dentro de la región **southcentralus**. Como consecuencia, no fue posible continuar con pruebas superiores a diez máquinas virtuales ni escenarios de mayor paralelismo.

La tabla 6.6 presenta los resultados correspondientes a las pruebas de destrucción de infraestructura, donde se evaluó el tiempo requerido para eliminar máquinas virtuales previamente desplegadas utilizando distintos niveles de paralelismo.

Prueba	OpenStack (s)	VMs destruidas	Azure (s)	VMs destruidas
Destruir 1 VM	11.637	1/1	80.033	1/1
Destruir 5 VM / P1	56.722	5/5	299.033	5/5
Destruir 5 VM / P5	20.450	5/5	167.124	5/5
Destruir 10 VM / P1	95.249	10/10	355.167	5/10
Destruir 10 VM / P5	34.077	10/10	No se realizaron pruebas	–
Destruir 10 VM / P10	38.231	10/10	No se realizaron pruebas	–
Destruir 15 VM / P1	164.946	15/15	No se realizaron pruebas	–
Destruir 15 VM / P5	61.117	15/15	No se realizaron pruebas	–
Destruir 15 VM / P15	43.895	15/15	No se realizaron pruebas	–
Destruir 30 VM / P1	276.901	30/30	No se realizaron pruebas	–
Destruir 30 VM / P5	100.178	30/30	No se realizaron pruebas	–
Destruir 30 VM / P10	85.178	30/30	No se realizaron pruebas	–
Destruir 30 VM / P15	47.426	30/30	No se realizaron pruebas	–
Destruir 30 VM / P30	56.455	30/30	No se realizaron pruebas	–

Tabla 6.6: Comparación de pruebas de destrucción entre *OpenStack* y *Microsoft Azure* en segundos.

Durante las pruebas de destrucción, *OpenStack* volvió a presentar menores tiempos de ejecución respecto a *Azure*. Por ejemplo, al destruir cinco máquinas virtuales utilizando paralelismo de cinco procesos, *OpenStack* completó la operación en aproximadamente 20 segundos, mientras que *Azure* requirió alrededor de 2 minutos con 47 segundos.

De igual forma, al incrementar el paralelismo dentro de *OpenStack*, se observó una reducción considerable en los tiempos de destrucción; sin embargo, también comenzaron a presentarse escenarios donde algunas máquinas virtuales no podían completarse correctamente debido a saturación de recursos dentro del entorno de laboratorio.

En términos de administración y configuración, ambas plataformas presentaron diferencias importantes. *Azure* ofreció una integración más sencilla mediante *Azure CLI*, *Azure Portal* y el proveedor *AzureRM*, permitiendo automatizar infraestructura con una menor cantidad de configuraciones manuales. Por otro lado, *OpenStack* requiere configuraciones adicionales relacionadas con redes, imágenes, conectividad entre nodos y administración de servicios internos de la nube.

También se identificaron diferencias en el nivel de control ofrecido por cada plataforma. *OpenStack*, al tratarse de una nube privada, permitió visualizar y administrar directamente servicios internos como *Nova*, *Neutron*, *Keystone* y los nodos *compute*, proporcionando un mayor nivel de control sobre la infraestructura. En contraste, *Microsoft Azure* ofreció una administración más abstraída y simplificada orientada al usuario final.

6.2. Análisis de ventajas de la automatización

El uso de *Terraform* como herramienta de Infraestructura como Código permitió identificar diversas ventajas frente al procedimiento manual de creación y administración de máquinas virtuales.

Una de las principales ventajas observadas fue la reducción de la intervención humana durante el despliegue. Mientras que el proceso manual requiere configurar múltiples parámetros en cada ejecución, el uso de *scripts* permitió reutilizar configuraciones previamente definidas, disminuyendo la repetición de tareas y la posibilidad de errores.

Asimismo, las pruebas estadísticas realizadas mediante 20 repeticiones por escenario demostraron que la ejecución repetida del mismo código produjo resultados consistentes y reproducibles en ambos entornos. Los coeficientes de variación obtenidos permitieron validar la estabilidad del proceso de aprovisionamiento incluso al utilizar distintos niveles de paralelismo.

Otra ventaja relevante fue la reducción de tiempos de operación y la capacidad de escalar el despliegue de recursos. Las pruebas de creación y destrucción mostraron que el uso de paralelismo disminuye considerablemente los tiempos de despliegue respecto a los escenarios secuenciales. Asimismo, fue posible administrar simultáneamente múltiples máquinas virtuales y evaluar el comportamiento del entorno bajo distintos niveles de carga. Las pruebas realizadas permitieron desplegar correctamente hasta 24 instancias simultáneas en *OpenStack* y 5 instancias en *Azure*, considerando las restricciones de recursos y cuotas disponibles en cada plataforma.

También se comprobó que *Terraform* facilita tareas de actualización, validación del archivo de estado y recuperación automática de configuraciones mediante procesos de *rollback*. Esto permitió detectar cambios externos, restaurar configuraciones previamente definidas y mantener sincronizada la infraestructura administrada. Asimismo, las pruebas demostraron la capacidad de *Terraform* para identificar automáticamente cambios realizados fuera del código mediante el archivo `terraform.tfstate`, facilitando la detección de desviaciones entre la infraestructura declarada y la infraestructura real, así como la recuperación automatizada del estado esperado.

Finalmente, el uso de archivos versionables permitió mantener un mayor control sobre la infraestructura desplegada, facilitando la reutilización, mantenimiento y adaptación del código entre distintos entornos *cloud*.

6.3. Limitaciones identificadas en la implementación

Durante el desarrollo e implementación del proceso automatizado se identificaron diversas limitaciones asociadas tanto a la infraestructura disponible como a las restricciones propias de cada plataforma.

En *Microsoft Azure*, las principales limitaciones estuvieron relacionadas con las cuotas regionales de *vCPU*, la disponibilidad de ciertos tamaños de máquina virtual y las restricciones impuestas por la suscripción utilizada. Estas condiciones limitaron la cantidad de recursos que podían desplegarse simultáneamente dentro de la región seleccionada.

Por otra parte, en *OpenStack* las limitaciones estuvieron asociadas principalmente a los recursos físicos disponibles dentro del laboratorio, así como a la capacidad de concurrencia de servicios como *MariaDB*, *RabbitMQ* y *Nova Scheduler*. A pesar de las optimizaciones realizadas sobre la infraestructura, durante las pruebas de saturación únicamente fue posible desplegar correctamente 24 de las 30 máquinas virtuales solicitadas.

Asimismo, se identificaron restricciones relacionadas con operaciones de *resize* y con la planificación de instancias, observándose errores del tipo *NoValidHost* en escenarios de alta concurrencia.

Otra limitación importante estuvo relacionada con el archivo `terraform.tfstate`. Aunque este mecanismo permitió mantener sincronizada la infraestructura y facilitar procesos de recuperación mediante *rollback*, también requirió un manejo cuidadoso para evitar inconsistencias entre el estado real y el estado administrado por *Terraform*.

En conjunto, estas limitaciones no afectan la viabilidad del enfoque propuesto, pero evidencian la necesidad de considerar tanto las restricciones físicas de la infraestructura como las políticas y cuotas definidas por cada proveedor al implementar soluciones basadas en Infraestructura como Código.

Capítulo 7

Conclusiones y trabajo futuro

7.1. Conclusiones

En esta tesis se implementó un esquema de automatización para la creación y administración de máquinas virtuales mediante Infraestructura como Código utilizando *Terraform* en una nube privada basada en *OpenStack* y una nube pública basada en *Microsoft Azure*.

Los resultados obtenidos validan la hipótesis planteada, confirmando que el uso de *Terraform* no solo estandariza y garantiza la reproducibilidad de los despliegues, sino que también ofrece un marco robusto para gestionar la infraestructura ante condiciones variables. A través de las pruebas experimentales, se demostró que este enfoque permite cuantificar el comportamiento operativo y la resiliencia del sistema frente a fallos y alta concurrencia. Asimismo, la consistencia estadística observada en las 20 repeticiones por escenario valida que el modelo de Infraestructura como Código (IaC) aquí desarrollado constituye una solución predecible y eficiente para la administración de recursos en plataformas heterogéneas. Las pruebas de paralelismo demostraron que el incremento de operaciones concurrentes reduce significativamente los tiempos de despliegue. Tanto en *OpenStack* como en *Microsoft Azure*. En particular, para los escenarios de mayor carga en *OpenStack*, los mejores resultados se registraron con paralelismo de 30, si bien el incremento de operaciones concurrentes reduce los tiempos de despliegue, la mejora en la eficiencia no es lineal. Además, se observó un punto de saturación donde el aumento en el número de hilos de ejecución (*parallel*) genera rendimientos decrecientes debido a la contención de recursos y las limitaciones de las API de los proveedores. En *OpenStack*, por ejemplo, se evidenció que la infraestructura alcanza un límite operativo donde el aumento excesivo de la concurrencia, a pesar de reducir tiempos, eleva la tasa de fallos, impidiendo el aprovisionamiento exitoso de la totalidad de las máquinas virtuales solicitadas (24 de 30). Este comportamiento subraya que la automatización debe equilibrarse con la capacidad real de los servicios internos de la plataforma.

Las pruebas también permitieron diagnosticar limitaciones específicas de cada entorno. En *OpenStack*, el cuello de botella residió en la disponibilidad de recursos físicos y la capacidad de gestión de sus servicios internos. Por su parte, en *Microsoft Azure*, las restricciones estuvieron dictadas por las cuotas de suscripción y la disponibilidad regional, lo que limitó la concurrencia a cinco instancias simultáneas y restringió operaciones de escalado. Esto demuestra que un esquema de *IaC* es efectivo, pero su éxito está condicionado por la arquitectura subyacente de la nube.

En conclusión, la comparación entre ambas plataformas permitió observar que, aunque *OpenStack* ofrece tiempos de respuesta menores y una mayor capacidad de carga en este entorno experimental, *Microsoft Azure* brinda una gestión más abstracta, aunque más restringida por políticas

comerciales. Finalmente, se comprobó que el flujo de trabajo basado en *Terraform* es universalmente aplicable; si bien la configuración es específica por proveedor, la lógica de automatización y gestión de estado es altamente reutilizable, validando que el uso de *IaC* es una estrategia robusta para transitar hacia entornos de nube híbrida, siempre que se comprendan y respeten los límites operativos de cada plataforma.

7.2. Trabajo futuro

Como trabajo futuro, una posible extensión de este proyecto consiste en integrar herramientas de automatización de configuración como *Ansible*, con el fin de complementar el aprovisionamiento realizado mediante *Terraform*. Esto permitiría automatizar tanto la creación de la infraestructura como la configuración de los sistemas desplegados.

Asimismo, podría ampliarse el alcance de las pruebas mediante entornos con múltiples nodos *compute*, balanceadores de carga, arquitecturas de alta disponibilidad y mecanismos de monitoreo. Esto permitiría evaluar el comportamiento de la solución en escenarios más cercanos a ambientes empresariales de producción y analizar aspectos relacionados con rendimiento, disponibilidad y escalabilidad.

También resultaría de interés incorporar módulos reutilizables en *Terraform* y almacenamiento remoto para el archivo `terraform.tfstate`, facilitando la organización de la infraestructura y la administración colaborativa de los recursos.

Finalmente, sería conveniente ampliar las pruebas comparativas entre nubes privadas y públicas e incorporar otros proveedores de nube. Además, una línea de investigación futura consiste en desarrollar mecanismos que permitan aumentar la portabilidad de configuraciones entre plataformas. Una solución capaz de traducir configuraciones definidas para una nube en recursos equivalentes para otra contribuiría al desarrollo de entornos *multicloud* y reduciría el esfuerzo necesario para migrar infraestructura entre distintos proveedores.

Apéndice A

Código *Terraform* para *OpenStack*

Este apéndice presenta el archivo de configuración utilizado para el despliegue automatizado de una máquina virtual en el entorno *OpenStack* mediante *Terraform*. El código incluye la definición del proveedor, las variables del despliegue, la consulta de la red existente y el recurso de cómputo empleado durante la implementación descrita en el Capítulo 4. Por motivos de seguridad, los datos sensibles asociados a la autenticación, como la contraseña de acceso, se muestran de forma oculta dentro del código presentado.

A.1. Archivo principal `main.tf`

A continuación, se muestra el contenido del archivo principal `main.tf` utilizado para el despliegue automatizado en *OpenStack*.

```
terraform {
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = ">= 1.54.0"
    }
  }
}

provider "openstack" {
  auth_url    = "http://ip/v3"
  user_name   = "admin"
  password    = "contrasena"
  tenant_name = "admin"
  domain_name = "Default"
  region      = "RegionOne"
  insecure    = true
}

variable "nombre_vm" {
  default = "vm-terraform-1"
}

variable "flavor" {
  default = "m1.tiny"
```

```

}
variable "imagen" {
    default = "cirros"
}
variable "red" {
    default = "private"
}
data "openstack_networking_network_v2" "red" {
    name = var.red
}
resource "openstack_compute_instance_v2" "instancia" {
    count = 5
    name      = "vm-terraform-${count.index + 1}"
    flavor_name = var.flavor
    image_id   = "9681ce2d-b638-43b5-ae1a-4df0d2203df1"
    availability_zone = "nova"
    network {
        uuid = data.openstack_networking_network_v2.red.id
    }
}

```

Apéndice B

Código *Terraform* para *Microsoft Azure*

Este apéndice presenta el archivo de configuración utilizado para el despliegue automatizado de una máquina virtual en *Microsoft Azure* mediante *Terraform*. El código incluye la definición del proveedor, las variables del despliegue, los recursos de red y la máquina virtual *Linux* utilizada durante la implementación descrita en el Capítulo 5. Por motivos de seguridad, los datos sensibles asociados a la autenticación y al acceso administrativo, como el identificador de suscripción y la contraseña, se muestran de forma oculta dentro del código presentado.

B.1. Archivo principal `main.tf`

A continuación, se muestra el contenido del archivo principal `main.tf` utilizado para el despliegue automatizado en *Microsoft Azure*.

```
terraform {
  required_version = ">= 1.5.0"
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = ">= 3.100.0"
    }
  }
}

provider "azurerm" {
  features {}
  subscription_id      = var.subscription_id
  resource_provider_registrations = "none"
}

# ===== VARIABLES =====
variable "num_vms" {
  type    = number
  default = 5
}

variable "nombre_vm" {
  type    = string
  default = "vm-terraform-azure-test"
```



```

}
variable "subscription_id" {
    type = string
}
variable "region" {
    type      = string
    default = "southcentralus"
}
variable "rg_name" {
    type      = string
    default = "rg-terraform-azure-test"
}
variable "vm_size" {
    type      = string
    default = "Standard_D2s_v3"
}
variable "admin_user" {
    type      = string
    default = "azureuser"
}
variable "admin_pass" {
    type      = string
    sensitive = true
}
variable "vnet_name" {
    type      = string
    default = "vnet-terraform-test"
}
variable "subnet_name" {
    type      = string
    default = "subnet-terraform-test"
}
variable "vnet_cidr" {
    type      = string
    default = "10.10.0.0/16"
}
variable "subnet_cidr" {
    type      = string
    default = "10.10.1.0/24"
}
# ===== RESOURCE GROUP =====
resource "azurerm_resource_group" "rg" {
    name      = var.rg_name
    location = var.region
}
# ===== NETWORK =====
resource "azurerm_virtual_network" "vnet" {

```

```

    name                = var.vnet_name
    location             = azurerm_resource_group.rg.location
    resource_group_name = azurerm_resource_group.rg.name
    address_space       = [var.vnet_cidr]
}
resource "azurerm_subnet" "subnet" {
    name                = var.subnet_name
    resource_group_name = azurerm_resource_group.rg.name
    virtual_network_name = azurerm_virtual_network.vnet.name
    address_prefixes    = [var.subnet_cidr]
}
# ===== NIC =====
resource "azurerm_network_interface" "nic" {
    count = var.num_vms
    name   = "${var.nombre_vm}-${count.index + 1}-nic"
    location = azurerm_resource_group.rg.location
    resource_group_name = azurerm_resource_group.rg.name
    ip_configuration {
        name                = "ipconfig1"
        subnet_id           = azurerm_subnet.subnet.id
        private_ip_address_allocation = "Dynamic"
    }
}
# ===== VM =====
resource "azurerm_linux_virtual_machine" "vm" {
    count = var.num_vms
    name   = "${var.nombre_vm}-${count.index + 1}"
    location = azurerm_resource_group.rg.location
    resource_group_name = azurerm_resource_group.rg.name
    size     = var.vm_size
    admin_username = var.admin_user
    admin_password = var.admin_pass
    disable_password_authentication = false
    network_interface_ids = [
        azurerm_network_interface.nic[count.index].id
    ]
    os_disk {
        caching              = "ReadWrite"
        storage_account_type = "Standard_LRS"
    }
    source_image_reference {
        publisher = "Canonical"
        offer     = "0001-com-ubuntu-server-jammy"
        sku       = "22_04-lts"
        version   = "latest"
    }
}
}

```

```
# ===== OUTPUT =====  
output "vm_ids" {  
  value = azurerm_linux_virtual_machine.vm[*].id  
}
```

Apéndice C

Configuración adicional del nodo *Compute*

Este apéndice presenta la modificación realizada en el archivo `/etc/libvirt/qemu.conf` del nodo *Compute*, la cual fue necesaria para la correcta gestión de las máquinas virtuales dentro del entorno *OpenStack*.

C.1. Archivo `/etc/libvirt/qemu.conf`

A continuación, se muestra la configuración aplicada en el archivo `/etc/libvirt/qemu.conf`.

```
user = "nova"
group = "nova"
dynamic_ownership = 0
```

C.2. Archivo `/etc/neutron/neutron.conf`

Asimismo, en el nodo *Compute* se realizó una configuración adicional dentro del archivo `/etc/neutron/neutron.conf`, específicamente en la sección `[agent]`, con el propósito de definir el mecanismo utilizado por *Neutron* para ejecutar operaciones privilegiadas relacionadas con red.

```
[agent]
root_helper = sudo neutron-rootwrap /etc/neutron/rootwrap.conf
```

Apéndice D

Configuración adicional del nodo *Controller*

D.1. Archivo `/etc/hosts`

A continuación, se muestra la configuración aplicada en el archivo `/etc/hosts` del nodo *Controller*, utilizada para la resolución local de nombres dentro del entorno *OpenStack*. En esta configuración se definieron entradas correspondientes al nodo controlador, al nodo de cómputo y a la interfaz externa utilizada para el acceso a la plataforma web. Estas entradas permiten facilitar la comunicación interna entre los servicios de *OpenStack* y asegurar la correcta resolución de nombres dentro de la infraestructura.

```
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4

x.x.x.x controller
x.x.x.x compute
x.x.x.x controller-ext

::1 localhost localhost.localdomain localhost6 localhost6.localdomain6
```

Bibliografía

- [1] HashiCorp, “Terraform documentation,” 2026.
- [2] S. Main, “Kvm and qemu as linux hypervisor,” Aug. 2018. Imagen extraída del artículo en Medium.
- [3] I. A. Cedillo Martínez, “Análisis de seguridad en una nube privada orquestada por openstack,” tesis de licenciatura, Universidad Nacional Autónoma de México, Facultad de Ingeniería, 2025. Tesis para obtener el título de Ingeniero en Telecomunicaciones.
- [4] OpenStack Documentation, “Install openstack services — installation guide: Minimal deployment for 2024.2 (dalmatian).” <https://docs.openstack.org/install-guide/openstack-services.html#minimal-deployment-for-2024-2-dalmatian>, 2026. Consultado en línea; página actualizada el 2026-02-18.
- [5] M. Leonard, “Red hat openstack and citrix netscaler.” <https://mcleonard.blogspot.com/2016/07/red-hat-openstack-citrix-netscaler.html>, November 2015. Consultado el 12 de marzo de 2026.
- [6] Microsoft Azure, “¿Qué son las nubes pública, privada e híbrida?.” <https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-are-private-public-hybrid-clouds>. Consultado en línea el 2026-03-31.
- [7] Red Hat, “Diferencias entre iaas, paas y saas.” <https://www.redhat.com/es/topics/cloud-computing/iaas-vs-paas-vs-saas>, 2023. Publicado el 4 de agosto de 2023; consultado en línea el 2026-03-31.
- [8] Red Hat, “¿Cuál es la diferencia entre la nube y la virtualización?.” <https://www.redhat.com/es/topics/cloud-computing/cloud-vs-virtualization>, 2023. Publicado el 5 de septiembre de 2023; consultado en línea el 2026-03-31.
- [9] Red Hat, “¿Qué es un hipervisor?.” <https://www.redhat.com/es/topics/virtualization/what-is-a-hypervisor>, 2023. Publicado el 23 de marzo de 2023; consultado en línea el 2026-03-31.
- [10] Arsys, “Escalado horizontal vs vertical: diferencias y ejemplos.” <https://www.arsys.es/blog/escalado-horizontal-vs-vertical>. Consultado en línea el 2026-03-31.
- [11] Microsoft Azure, “¿Qué es el escalado horizontal y vertical?.” <https://azure.microsoft.com/es-mx/resources/cloud-computing-dictionary/scaling-out-vs-scaling-up>. Consultado en línea el 2026-03-31.

- [12] Red Hat, “Aplicaciones con estado vs sin estado.” <https://www.redhat.com/es/topics/cloud-native-apps/stateful-vs-stateless>, 2023. Consultado en línea el 2026-03-31.
- [13] Red Hat, “Red hat enterprise linux server.” <https://www.redhat.com/es/technologies/linux-platforms/enterprise-linux/server>. Consultado en línea el 2026-03-31.
- [14] Red Hat, “¿Qué es una máquina virtual?.” <https://www.redhat.com/es/topics/virtualization/what-is-a-virtual-machine>, 2023. Consultado en línea el 2026-03-31.
- [15] Red Hat, “¿Qué es la infraestructura como código (iac)?.” <https://www.redhat.com/es/topics/automation/what-is-infrastructure-as-code-iac>, 2023. Consultado en línea el 2026-03-31.
- [16] Arsys, “¿Qué es terraform y cuáles son sus ventajas?.” <https://www.arsys.es/blog/que-es-terraform-y-cuales-son-sus-ventajas>. Consultado en línea el 2026-03-31.
- [17] IBM, “Terraform.” <https://www.ibm.com/mx-es/think/topics/terraform>. Consultado en línea el 2026-03-31.
- [18] YouTube, “Introducción a terraform.” <https://www.youtube.com/watch?v=SeR1a-79dII>, 2023. Video en línea; consultado el 2026-03-31.