PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

SEDE ESMERALDAS

CARRERA

INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

TESIS DE GRADO

TERRAFORM COMO HERRAMIENTA PARA AUTOMATIZAR LA CREACIÓN DE INFRAESTRUCTURAS SIGUIENDO EL CONCEPTO “INFRAESTRUCTURA COMO CÓDIGO”

PREVIO A LA OBTENCIÓN DE TÍTULO DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

LÍNEA DE INVESTIGACIÓN

PROGRAMACIÓN Y DESARROLLO DE SOFTWARE

AUTOR

JURGEN RONALDO HUERLO QUINTERO

ASESOR

MGT. WILSON CHANGO

ESMERALDAS, 2020

TRIBUNAL DE GRADUACIÓN

Trabajo de tesis aprobado luego de haber dado cumplimiento a los requisitos exigidos por el Reglamento de Grado de la PUCESE previo a la obtención del título de Ingeniería en Sistemas y Computación.

Presidente del Tribunal de Graduación

Lector 1
Mgt. Marc Grob

Lector 2
Mgt. José Carvajal

Director(a) de Escuela
Mgt. Susana Patino

Director de Tesis
Mgt. Wilson Chango

AUTORÍA DE TESIS

Yo, JURGEN RONALDO HUERLO QUINTERO, portador de la cédula de ciudadanía 0804593104 declaro que el desarrollo de la presente investigación es totalmente de carácter original, auténtico y personal.

En tal virtud, manifiesto que este estudio efectuado bajo las directrices establecidas por las Normas IEEE es de exclusiva responsabilidad legal y académica del autor

---

JURGEN RONALDO HUERLO QUINTERO

DEDICATORIA

Esta investigación se la dedico principalmente a Dios, por ser mi fuente de inspiración, por darme fuerza y salud para llevar a cabo mis metas y objetivos. Quiero darle las gracias por su amor infinito.

A mi mamá Manlia Quintero por siempre apoyarme en mis estudios e inculcarme buenos valores que me servirán en el resto de mi vida profesional, también a mi papá Fernando Huerlo por darme consejos en los momentos en los que más los necesitaba.

A mis Abuelos Segundo Quintero y Gladys Moreira por cuidar de mi desde pequeño, quererme y cuidarme tanto como a su propio hijo, por enseñarme que si quiero puedo superarme y lograr mis objetivos, enseñarme a apreciar la música y tocar la guitarra la cual es una de mis grandes pasiones.

Jurgen Ronaldo Huerlo Quintero.

AGRADECIMIENTO

Agradezco a Dios por darme el don de la perseverancia, la fuerza y dedicación que me permitió lograr cada reto presentado en esta etapa de mi vida.

A mi familia, por ser parte de este importante proceso quienes con paciencia y cariño lograron ser mi apoyo y refugio.

A mi asesor Mgt. Wilson Chango quien me ha orientado y aportado sabias contribuciones en la preparación de mi trabajo de investigación, también agradezco a la plana de docentes de la Escuela de Sistemas y Computación quienes con paciencia y profesionalismo me guiaron en el camino a lo largo de la carrera.

Finalmente, agradezco a quienes hicieron de mi vida universitaria un camino lleno de alegrías y bellas experiencias.

Jurgen Ronaldo Huerlo Quintero

ÍNDICE DE CONTENIDO

LISTA DE FIGURAS

LISTA DE ANEXOS

LISTA DE TABLAS

LISTA DE GRAFICOS

LISTA DE ECUACIONES

RESUMEN

Evaluar el nivel de factibilidad técnica de la herramienta Terraform para ser adoptada en el ciclo de vida del DevOps mediante la creación y aprovisionamiento de una instancia en la nube. Se estudia esta herramienta ya que permite mejorar la administración de la infraestructura y de esta mejora se beneficiarían los departamentos de TI (Tecnologías de la información), programadores independientes, pequeñas y grandes empresas.

Para realizar el estudio se emplearon 3 proveedores de servicios de la nube con el fin de observar si existen limitaciones al usar diferentes proveedores. Para poder evaluar de forma correcta Terraform se utilizó una metodología experimental ya que es necesario probar la herramienta y ejecutar scripts que creen una infraestructura con recursos, también se utilizó una ficha de evaluación al software para medir 3 características las cuales son usabilidad, eficiencia de desempeño y compatibilidad (Integración).

Los resultados obtenidos demuestran que, si existe una diferencia cuando se emplea Terraform en uno u otro proveedor, estas diferencias son en la sintaxis y configuración mayormente, también se reveló que los tiempos de respuesta de las ejecuciones también cambian dependiendo del proveedor lo cual es un factor importante para tomar en cuenta.

Estos resultados en comparación con otros estudios anteriormente realizados por otros autores, son realmente buenos ya que en algunos casos se coincidió con algunos estudios que estaban en la misma línea de implementación de Terraform, en otros casos es totalmente distinto ya que se usaron herramientas distintas o las métricas propuestas no servían de mucho para Terraform, aunque sin lugar a dudas uno de los mayores logros descubrir que Terraform puede ser incluido en entornos novedosos como CI/CD (Integración Continua y Despliegue continuo por sus siglas en ingles). Estos resultados se interpretaron como un gran acierto ya que se pudo evaluar de forma correcta la herramienta.

ABSTRACT

Evaluate the level of technical feasibility of the Terraform tool to be adopted in the DevOps life cycle by creating and provisioning an instance in the cloud. This tool is studied as it allows improving the management of the infrastructure and this improvement would benefit IT departments, independent programmers, small and large companies.

To carry out the study, 3 cloud service providers were used to observe if there are limitations when using different providers. To correctly evaluate Terraform, an experimental methodology was used since it is necessary to test the tool and execute scripts that create an infrastructure with resources, a software evaluation sheet was also used to measure 3 characteristics which are Usability, Performance Efficiency and Compatibility (Integration).

The results obtained show that, if there is a difference when Terraform is used in one or another provider, these differences are mostly in the syntax and configuration, it was also revealed that the response times of the executions also change depending on the provider, which is an important factor to consider.

These results in comparison with other studies previously carried out by other authors, are good since in some cases some studies that were in the same line of implementation of Terraform were coincided, in other cases it is totally different since different tools were used or the proposed metrics were not of much use for Terraform, although without a doubt one of the greatest achievements to discover that Terraform can be included in novel environments such as CI / CD. These results were interpreted as a great success since the tool could be correctly evaluated.

INTRODUCCIÓN

Presentación de la investigación

En la actualidad la mayoría de grandes empresas en el mundo están adoptando nuevas tecnologías para automatizar procesos. De esta forma se consigue agilizar el despliegue de aplicaciones en entornos de producción. En su investigación Lavriv *et al.* [1] sostiene que una de las formas más conocidas son los entornos de integración y despliegue Continuo (CI/CD por sus siglas en inglés). Una de las metas de estos entornos es darle la responsabilidad al desarrollador de ejecutar su código en entornos similares a los de producción, evitar errores y obtener un software de calidad [1]. El siguiente paso es agilizar y automatizar la creación de infraestructuras y aprovisionarlas de modo que el despliegue de cualquier sistema sea ágil y pueda ser replicado en diferentes situaciones que lo requieran.

El concepto de IAC (Infraestructura como código por sus siglas en inglés) supone que la infraestructura puede representarse como un código de programa que puede ser interpretado por alguna herramienta de automatización a las instrucciones de la API (Interfaz de Programación de Aplicaciones) en la nube según Lavriv *et al.* [1]. Terraform, de la empresa HashiCorp, es una de las herramientas que permite llevar este concepto a la práctica. Como se explica en la investigación de Barath [2] “la infraestructura se describe utilizando una sintaxis de configuración de alto nivel. Esto permite que un plano de su centro de datos sea versionado y tratado como lo haría con cualquier otro código”. Además, la infraestructura se puede compartir y reutilizar.

Esta investigación se encuentra organizado en tres apartados donde el primer apartado se encuentra todo el marco teórico que respalda esta investigación, en el segundo apartado se encuentran las metodologías que se emplean y por último está el apartado de los aspectos administrativos en él se muestran los recursos, cronogramas y financiamientos que respaldan esta investigación.

Planteamiento del problema

Actualmente, aunque existan herramientas de automatización no siempre son usadas ya que algunas empresas y departamentos de TI siguen optando por contratar o designar a una persona específica para la creación y aprovisionamiento de instancias que serán utilizadas para desplegar aplicaciones. Al momento de migrar las infraestructuras que

fueron creadas de forma tradicional a un formato de código es inevitable pensar en preguntas como las siguientes: ¿Es conveniente administrar las infraestructuras usando un formato de código?, ¿Es seguro migrar las infraestructuras a un concepto como lo es IAC? Todo esto debido a que son tecnologías nuevas a las que no todos están acostumbrados y si en algún caso llegan a fallar esto supone grandes pérdidas debido a que las aplicaciones y servicios que están alojados en estas infraestructuras no estarían funcionando y es por esto por lo que buscar una herramienta que brinde soporte en casos de emergencia, sea estable y brinde muchos beneficios es una tarea un tanto compleja.

Tratando de dar solución a estas preguntas y otras demandas de los usuarios y empresas, HashiCorp [3] crea Terraform la cual es una herramienta que bajo el concepto (IAC) permite crear y aprovisionar instancias usando una sintaxis que es parecida a los lenguajes de programación modernos, esto también puede suponer desventajas ya que se pueden cometer las mismas malas prácticas de programación.

Con la aparición de esta herramienta también nacen otras incógnitas como: ¿Es Terraform la herramienta adecuada para automatizar la creación de instancias en la nube? Para ello es que esta investigación se realiza para tratar de dar respuesta a las preguntas que usuarios o empresas puedan tener al momento de elegir Terraform y el concepto de IAC para administrar sus infraestructuras.

A. Rahman, J. Stallings y L. Williams [4], manifiestan que los scripts de IAC escritos en Terraform ayudan a crear y administrar la implementación de software de una manera ágil de forma similar al código fuente de un software; los scripts de IAC cambian frecuentemente y pueden contener defectos. Los defectos en estos scripts pueden tener consecuencias nefastas: por ejemplo nuevamente en [4] se explica que GitHub experimentó una interrupción del DNS (Sistema de nombres de dominio por sus siglas en inglés) causada por un defecto en un script de IAC. Para evitar estos errores se deben realizar pruebas en diferentes ecosistemas de la nube con el fin de encontrar errores que puedan afectar directamente al funcionamiento de Terraform y tenerlos en cuenta. De esta forma se evitan problemas a un corto y mediano plazo.
Para evitar la pérdida de los archivos de configuración se utiliza un manejador de versiones y mantener un control sobre los cambios que se le realizan a cada uno de los archivos, en este caso es GitLab [5], el servicio preferido por la mayoría de DevOps debido a que es fácil de usar, y permite integraciones con diferentes herramientas de la misma área.

Justificación

La importancia de esta investigación recae en el hecho de obtener resultados que aseguren la efectividad de Terraform frente a diferentes proveedores de la nube y quede constancia que ya es una herramienta apta para ser utilizada por empresas y personas particulares. Este proyecto entrega una nueva perspectiva del porque es bueno adoptar esta herramienta ya que se realizarán pruebas, se estudiará la curva de aprendizaje y entre otros parámetros para definir y concluir en que Terraform es el siguiente paso en lo que se refiere a la automatización en los procesos de crear y aprovisionar instancias en la nube.

La presente investigación pretende estudiar administración de la infraestructura ya que es una tarea complicada hoy en día debido a que si un componente de esta falla es necesario la presencia física de una persona para poder solucionar el problema. El hecho de depender de una persona que se encuentre disponible para llevar a cabo una tarea que involucre estar ingresando a la consola del proveedor y configurar todo manualmente en estos tiempos en los que todo se está automatizando es inaceptable ya que significa que no hay adaptación a las nuevas tendencias por parte de las empresas o usuarios particulares. Terraform llega a solucionar estos inconvenientes que tienen las grandes empresas e incluso pequeños departamentos de TI (Tecnologías de la información) ya que trabajando de forma remota se puede realizar cambios en el código de la infraestructura y ejecutar el script para solucionar el problema. Es por esto y entre otras razones por la cual esta investigación se centra en evaluar la herramienta antes mencionada y dar certeza de que es apta para ser utilizada.

Objetivos

Objetivo general

Evaluar el nivel de factibilidad técnica de la herramienta Terraform para ser adoptada en el ciclo de vida del DevOps mediante la creación y aprovisionamiento de una instancia en la nube.

Objetivos específicos

1. Definir el alcance y limitación técnica de Terraform en ambientes de producción.
2. Aplicar la herramienta Terraform en ecosistemas heterogéneos de computación en la nube (Google Cloud Platform, Amazon Web Services y Microsoft Azure).
3. Conocer el comportamiento de las instancias creadas a partir de Terraform en los ecosistemas de computación en la nube en los que se hizo la integración.

Capítulo 1

1. MARCO TEÓRICO

1.1. Antecedentes

El desarrollo de esta investigación ha sido posible gracias a la búsqueda de diferentes artículos científicos y libros que se encuentran alojados en diferentes bases de datos los cuales fueron recuperados usando la siguiente cadena de búsqueda: TERRAFORM OR INFRASTRUCTURE AS A CODE OR IAC. Con esta cadena de búsqueda se recuperaron 25 artículos y libros entre las diferentes bases de datos como: IEEE Explorer, ACM (Association for Computing Machinery) , Scopus, Springer y Willey. Los criterios que se utilizaron para seleccionar los artículos en los que se ha basado esta investigación son: por tipo de herramienta lo cual está explicado en el segundo, tercer y cuarto párrafo de esta sección y por proveedor que se explica en los últimos dos párrafos de esta sección.

La herramienta Argon es utilizada en el estudio presentado por Sandobalin *et al* [6], el cual tiene como objetivo "Comparar una herramienta impulsada por modelos (Argon) con una conocida herramienta centrada en código (Ansible)", emplearon una metodología experimental ya que llevaron a cabo una familia de 3 experimentos, como resultado se obtuvo que Argon es más eficaz en lo que respecta a apoyar el enfoque de IAC en términos de definir la infraestructura en la nube; por lo tanto llegaron a la conclusión de que Argon es más eficaz al modelar la infraestructura de la nube y automatizar la generación de scripts para diferentes herramientas de DevOps.

En la investigación de Dalla *et al* [7], se propone un catálogo que consta de 46 métricas para identificar propiedades de IAC, para lo cual se utilizó la metodología de revisión bibliográfica ya que los autores hicieron uso de varios artículos científicos y se obtuvieron 46 métricas de las cuales 8 son independientes del lenguaje, 14 han sido adaptadas y 24 se refieren a algunas características de Ansible que también son observables en otros lenguajes de configuración y orquestación; la conclusión a la que llegaron los autores fue que estas métricas pueden ser aplicadas a otros lenguajes basados en métricas por ejemplo (Chef y Pupet).

Dentro de la línea que sigue el concepto de IAC el cual es automatizar procesos, también se encuentran otras herramientas como Docker, Chef, Puppet, Jenkins, entre otras y las cuales son explicadas en la investigación de A. Agarwal, S. Gupta y T. Choudhury [8], que se titula "Desarrollo de software continuo e integrado usando DevOps" en donde se

proporcionan varias metodologías y herramientas que pueden constituir una canalización de CI/CD eficaz, para ello, se realizó una revisión de la literatura con diferentes preguntas de investigación, lo que se consiguió fue plantear una propuesta para el flujo de trabajo y los test; tomando en cuenta la revisión de la literatura que los autores realizaron, se llegó a la conclusión de que son muy pocos los estudios publicados con respecto al área de DevOps lo cual puede influir en porqué es tan difícil adoptar estas tecnologías.

Terraform ha sido empleado en diferentes investigaciones, pero particularmente los autores D. Ivanova, P. Borovska y S. Zahov [9], es utilizada en el proceso de desarrollo de una Plataforma como servicio (PaaS por sus siglas en ingles) el objetivo de esta investigación es lograr la creación de este entorno en la nube mediante Terraform, la metodología empleada no se especifica pero debido a que se desarrolla en un entorno real, se intuye que es una metodología experimental, los resultados obtenidos son el entorno que se ha creado usando una moderna herramienta de infraestructura como código; la conclusión a la que llegaron es que pueden tener un mayor control de su arquitectura usando Terraform ya que todo está declarado en forma de código.

Es importante recalcar que también se han publicado estudios como el del autor A. Rahman [10], el cual ahonda en las "Características defectuosas de los Scripts de infraestructura como código", el objetivo de esta investigación es identificar las características de los scripts de IAC y el proceso de desarrollo de IAC que se correlacionan con defectos y violan los objetivos de seguridad y privacidad, la metodología empleada es una revisión de la literatura y los resultados que se consiguen son varias propuestas para evitar los defectos en los scripts de IAC; el autor concluye su investigación proponiendo el estudio de las características estructurales que se relacionan con los defectos.

1.2. Bases teóricas-conceptuales

1.2.1. Infraestructura como código

1.2.1.1. Definición

Una definición para la IAC según Kief Morris [11], en su estudio manifiesta que: "Infraestructura como código es un enfoque para administrar la infraestructura de TI (Tecnología de la Información) para la era de la nube, los microservicios y la entrega continua que se basa en prácticas del desarrollo de software", es importante porque otra definición como la que del estudio [1], explica con claridad el concepto de IAC dicta

que “la infraestructura puede representarse como un código de programa que puede ser interpretado por alguna herramienta de automatización a las instrucciones de la API en la nube” [1]. Esto quiere decir que la IAC es una forma de describir la estructura de una infraestructura, pero en forma de código obteniendo así beneficios como el versionado.

1.2.1.2. Arquitectura

Para realizar una correcta implementación del concepto IAC y Terraform se pueden seguir arquitecturas, que ya indican como debe ser la implementación de las herramientas y cuando deben ser implementadas. ARGON de la investigación de J. Sandobalin, E. Insfran y S. Abrahao [12], se indica como deben estar estructuradas las herramientas para un correcto modelado de la infraestructura, en la Figura 1 Arquitectura ARGON, recuperada de se puede apreciar como Terraform es utilizado en conjunto con Ansible y en Amazon y Azure.

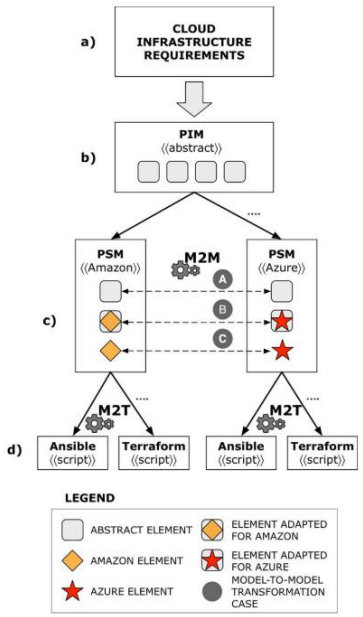


Figura 1 Arquitectura ARGON, recuperada de [12]

En el mismo estudio [12], se explican algunas de las características de esta arquitectura como por ejemplo PIM (Plataforma independiente del modelo), en esta etapa es cuando el metamodelo de una infraestructura abstracta es definido, lo que hace es abstraer las capacidades de la nube. En la práctica no se suele crear un PIM. PSM(Plataforma específica del modelo), es aquí donde se especifican los recursos para un proveedor particular, se utiliza la plataforma independiente del modelo definida en la anterior capa para los proveedores específicos proveedores de la nube en la figura 1 se muestra un PSM para Amazon y Azure [12]. De este modo es importante darse una idea de cómo poder

ejecutar una arquitectura en la cual se están implementando herramientas que se basan únicamente en realizar las acciones mediante código que es declarado en un archivo.

1.2.1.3. Infraestructura Mutable e Inmutable

Muchas veces cuando se desarrollan aplicaciones, están se van actualizando con el tiempo y salen nuevas versiones que pueden consumir más o menos recursos es entonces que surge la necesidad de acomodar la infraestructura que soporta a esa aplicación para ello hay dos opciones que se pueden seguir. Una de ellas es actualizar la aplicación sin cambiar la instancia, de esta forma se acomodan los recursos para la nueva versión de la aplicación a esto se le conoce como infraestructura mutable ya que va cambiando a través de las versiones, por otro lado T. Grønli y R. Kazman [13], explican que se puede crear una instancia dedicada a la nueva versión y borrar la instancia de la versión anterior esto implica que se está creando una nueva infraestructura por lo cual no hay cambios, solo se reemplaza una instancia por otra. A esta forma de gestionar la infraestructura se le conoce como infraestructura inmutable ya que no hay cambios. Aunque Terraform permite realizar cambios a las infraestructuras ya existentes, en muchos casos no es lo recomendable debido a que el objetivo de esta herramienta es proveer de infraestructuras que puedan ser reemplazadas por completo sin ningún problema, y de hecho tienen limitaciones en algunas características como por ejemplo actualizar el tipo de procesador en una instancia que se está ejecutando, en este caso la herramienta recomienda eliminar la instancia y volver a levantar otra con el procesador deseado.

1.2.1.4. Principios de la Infraestructura como código

La IAC tiene bases en las cuales se apoya, algunas de estas pueden llegar a ser parte de los beneficios que este concepto ofrece los cuales son: Los sistemas se pueden reproducir repetidamente, los sistemas pueden ser desechables, consistencia de configuración y todo debe ser versionado. A continuación, se habla a detalle de cada uno de estos principios.

- Los sistemas se pueden reproducir repetidamente, en [14], se habla que los sistemas pueden ser fácilmente reproducidos debido a que cada elemento de la infraestructura se puede reproducir repetidamente de manera confiable y sin esfuerzo, esta comprensión es crucial porque los scripts IAC describen todos los pasos necesarios para crear y aprovisionar el recurso solicitado, por ejemplo, qué software debe instalarse, su versión correcta, nombre de host, entre otros.

- Sistemas Desechables, en [14] se encuentra la definición de este principio la cual es “IAC ha mejorado los viejos sistemas estáticos en sistemas dinámicos y desechables que pueden cambiar de una manera rápida y fácil por lo cual los recursos de las nuevas infraestructuras dinámicas se crean, destruyen, actualizan, redimensionan y reubican fácilmente en el sistema”.

Esto es fundamental porque antes los sistemas eran estáticos y de un solo propósito, ya que eran creados de forma específica para cumplir una necesidad, pero con la IAC esto se ha dinamizado ya que esta permite que la infraestructura que apoya estos sistemas sea escalable debido a que se pueden actualizar y redimensionar acorde a lo que se necesita. Nuevamente en [14], el autor indica que las mejoras y los parches en la infraestructura se hicieron más fáciles Esto implica que los cambios se pueden manejar sin problemas su investigación porque deja claro que esto es crucial en las infraestructuras de nube a gran escala donde el sistema no puede confiar en el hardware subyacente.

- Los humanos siempre han tenido gran participación en los errores que se producen en la consistencia de configuración, aunque existan estándares y metodologías siempre existe ese margen de error creado por los humanos y es por lo que Michael Hüttermann [14] sugiere que:

  La implementación de IAC estandariza completamente la configuración de la infraestructura, dejando poco espacio para errores humanos. Como resultado, las posibilidades de encontrar problemas de incompatibilidad se reducen considerablemente, y la ejecución de las aplicaciones en ejecución se vuelve más consistente y fluida [14].
- Todo debe estar versionado, según [14] tener la infraestructura como código permite usar Sistemas de Control de Versiones (VCS por sus siglas en ingles), para mantener un control de la calidad del código o restaurar a una versión estable cuando algún bloque de código nuevo está causando un error o no está funcionando. Cada característica del sistema se identifica mediante etiquetas y números de versión, lo que mejora el rastreo y la fijación de anomalías más complicadas [14]. Además, el VCS permite dar paso a el código de la infraestructura pueda ser procesado en entornos CI/CD porque estos trabajan con código.

1.2.2. Herramientas para el desarrollo de Infraestructura como código

IAC como concepto ha dado lugar a que muchas empresas desarrollen sus propias herramientas para satisfacer las necesidades de sus trabajadores o clientes en el caso de los proveedores de servicios en la nube. Sea este el caso de AWS (Servicios web de Amazon por sus siglas en ingles) . Por otro lado, HashiCorp ofrece Terraform que es la herramienta que se estudia en esta investigación y del cual se habla en el siguiente apartado.

La herramienta AWS "*CloudFormation"* según la web oficial de la herramienta [15] provee de un método fácil para la creación y administración de los recursos AWS que están relacionados de esta manera se facilita el aprovisionamiento y la actualización. Provee de plantillas con configuraciones que son las más usadas y recomendadas por los usuarios, es importante recalcar que se pueden crear plantillas personalizadas desde cero y adaptara los recursos según se requiera [15], esto es importante ya que en los libros que explican su funcionamiento y en la web oficial informan que:

> Proporciona un lenguaje común para que modele y aprovisione recursos de aplicación de AWS y de terceros en su entorno de nube. AWS *CloudFormation* permite utilizar lenguajes de programación o un archivo de texto simple para modelar y aprovisionar, de una manera segura y automatizada, todos los recursos necesarios para las aplicaciones en todas las regiones y cuentas. Esto proporciona una única fuente de confianza para los recursos de AWS y de terceros [5], [16].

1.2.2.1. Terraform

Dentro de las opciones que Terraform ofrece para los distintos tipos de usuarios con diferentes necesidades se encuentra Terraform Cloud. Esta es una aplicación que ayuda a los equipos a usar Terraform de manera conjunta. Gestiona las ejecuciones de Terraform según su web oficial [17] es un entorno consistente y confiable, este proporciona un acceso a datos compartidos de estado y secretos, controles de acceso para aprobar cambios en la infraestructura y entre otras opciones que puedan ser de interés para equipos de trabajo.

Terraform Enterprise es otra de las opciones que esta herramienta ofrece, pero en este caso va directamente enfocado en las empresas. Según su página oficial [18] esta es una distribución auto hospedada de Terraform Cloud. Ofrece a las empresas una instancia privada de la aplicación Terraform Cloud, sin límites de recursos y mayores prestaciones arquitectónicas adicionales para trabajos de un nivel empresarial entre

estas se encuentran algunas como el registro de auditoría y el inicio de sesión único SAML (*Secure Assertion Markup Language,* por sus siglas en inglés).

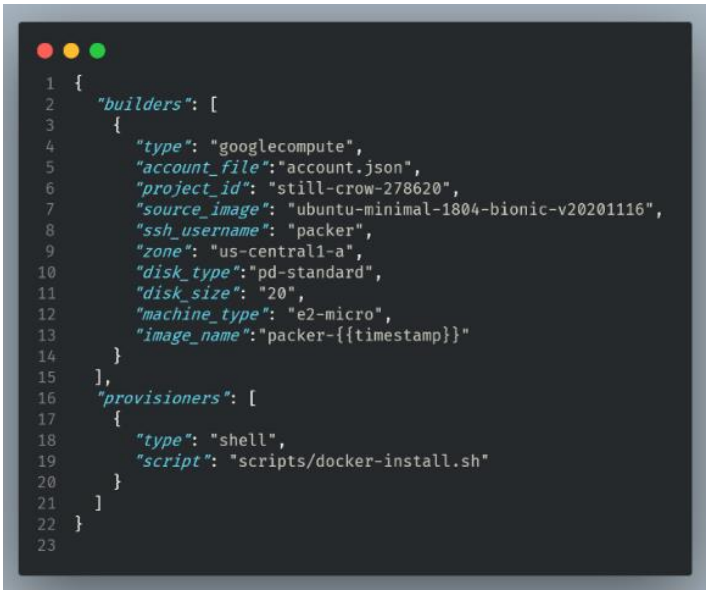
"Terraform es una herramienta para construir, cambiar y versionar infraestructura de forma segura y eficiente. Terraform puede administrar proveedores de servicios existentes y populares, así como soluciones internas personalizadas" [3], lo que significa que es independiente del proveedor. Terraform CLI (Interfaz de Línea de Comandos) es en sí la herramienta donde se ingresan los comandos como puede ser un `terraform init`. Esta herramienta de línea de comandos forma parte de las dos opciones mencionadas anteriormente ya que es imprescindible para usar Terraform.

Una de las mayores ventajas que Terraform posee frente a otras herramientas como *CloudFormation* es que no depende de un proveedor único como AWS si no que puede ser empleada en cualquier proveedor entre los cuales están *Google Cloud Platform* (GCP por sus siglas en ingles) y además de proveedores de servicios *Cloud*, también proporciona soporte para otros tipos de proveedores de soluciones de tipo PAAS (Plataforma Como Servicio) por sus siglas en inglés) o SAAS (Software Como Servicio por sus siglas en inglés) [15]. Las limitaciones llegan con los errores que se comenten al momento de programar ya que al usar HCL (HashiCorp Lenguaje por sus siglas en inglés) que es el medio para declarar la infraestructura y es en este punto cuando se cometen malas prácticas de programación y errores comunes por ejemplo *"Code Smells"* [19].

1.2.2.2. Packer

Medina y Schumann [20] explican que Packer es una herramienta creada por HashiCorp, para facilitar la creación de imágenes de máquinas. Anima a utilizar los scripts automatizados, para instalar y configurar todo el software necesario dentro de las imágenes de máquinas creadas por Packer.

Es una herramienta de código abierto que se puede utilizar en conjunto con Terraform, es liviano y se puede ejecutar en cualquiera de los sistemas operativos principales, los archivos pueden ser declarados en formato JSON (JavaScript object notation por sus siglas en inglés) con archivos con la sintaxis HCL, en donde se declara el sistema operativo, las credenciales para conectarse al proveedor, el tipo de máquina, el tamaño del almacenamiento y principalmente cuentan con un apartado de *provisioners* en donde se pueden especificar acciones de aprovisionamiento, por ejemplo en la Figura 2 Script de Packer se puede apreciar que se ejecuta un script que va a instalar Docker.



```
{
  "builders": [
    {
      "type": "googlecompute",
      "account_file":"account.json",
      "project_id": "still-crow-278620",
      "source_image": "ubuntu-minimal-1804-bionic-v20201116",
      "ssh_username": "packer",
      "zone": "us-central1-a",
      "disk_type":"pd-standard",
      "disk_size": "20",
      "machine_type": "e2-micro",
      "image_name":"packer-{{timestamp}}"
    }
  ],
  "provisioners": [
    {
      "type": "shell",
      "script": "scripts/docker-install.sh"
    }
  ]
}
```

Figura 2 Script de Packer. Fuente: Elaboración Propia

La forma en la que funciona esta herramienta es que toma toda la configuración declarada en el archivo y crea una instancia temporalmente a la cual Packer se conecta por medio de SSH para realizar las acciones declaradas en los *provisioners* luego de realizar todas estas acciones se elimina la instancia, pero se guarda una imagen de esta la cual ya estaría lista para ser usada manualmente en la consola del servidor o automáticamente especificándola en un script de Terraform.

1.2.2.3. Ansible

Es una poderosa herramienta de gestión de la configuración, en [2] se indica que “la singularidad de Ansible en comparación con otras herramientas de gestión es que también se utiliza para la implementación y la orquestación”. Mejora la seguridad al funcionar en un modelo basado en push donde las maquinas remotas reciben las partes que son necesarias del código y las maquinas remotas no interfieren con la configuración de las otras máquinas [2]. Las configuraciones se realizan en archivos YML en los cuales se describen las acciones que realizara ansible el cual es mayormente utilizado para aprovisionar entornos ya sea de software o complementos para que los despliegues sean correctos.

1.2.3. Herramientas para el versionado de código

1.2.3.1. GitHub

Es un sistema en el cual se alojan repositorios git el cual actualmente es propiedad de Microsoft, en este sistema se puede ver de forma gráfica las ramas que existen en el

repositorio también se pueden ver los cambios que se han hecho por cada uno de los que intervienen en el proyecto [21].

Actualmente GitHub cuenta con herramientas enfocadas en ayudar a los desarrolladores y a los DevOps como por ejemplo GitHub actions [22] el cual automatiza, personaliza y ejecuta tus flujos de trabajo de desarrollo de software directamente en tu repositorio con GitHub Actions. Puedes descubrir, crear y compartir acciones para realizar cualquier trabajo que quieras, incluido CI/CD, y combinar acciones en un flujo de trabajo completamente personalizado.

Es una buena opción para poder versionar los scripts de IAC y mantener un control de calidad en las características que se van agregando de forma en que si llega a fallar siempre existe una versión comprobada que funciona.

1.2.3.2. GitLab

Es otro sistema para manejar repositorios Git, pero de código libre el cual también facilita el desarrollo colaborativo entre varios desarrolladores se pueden configurar los proyectos para que sean públicos de libre acceso para todos o privado con diferentes permisos para cada uno de los miembros, permite la integración con herramientas para editar el código de forma online como GitPod.

GitLab CI / CD según su web oficial [23] está integrado en GitLab, es una aplicación web con una API que almacena su estado en una base de datos también gestiona proyectos y provee de una interfaz de usuario agradable, además de otras funciones de GitLab. GitLab Runner es una aplicación la cual procesa los proyectos, esto se puede implementar por separado y funciona con GitLab CI / CD a través de una API.

1.2.4. Normativas para evaluar calidad

1.2.4.1. Calidad de software

Para evaluar la calidad de un producto software se han creado modelos para llevar un control de calidad y la definición más fiable que se puede encontrar es en la página de la ISO 2500 la cual es:

> La calidad del producto software se puede interpretar como el grado en que dicho producto satisface los requisitos de sus usuarios aportando de esta manera un valor. Son precisamente estos requisitos (funcionalidad, rendimiento, seguridad, mantenibilidad, etc.) los que se encuentran representados en el modelo de calidad, el

cual categoriza la calidad del producto en características y subcaracterísticas [24], esto permite obtener un producto de calidad.

1.2.4.2. ISO 25010 para la evaluación de software

Dentro de esta norma existen diferentes apartados para evaluar la calidad de un software entre los cuales están: Usabilidad, Fiabilidad y Eficiencia de Desempeño los cuales serán explicados a continuación:

- "Usabilidad es la capacidad del software para entendido, aprendido y usado por cualquier usuario en cualquier situación" [24], esto demuestra la experiencia del usuario al interactuar con el producto o sistema.
- "Compatibilidad es la Capacidad de dos o más sistemas o componentes para intercambiar información y/o llevar a cabo sus funciones requeridas cuando comparten el mismo entorno hardware o software" [24].
- "Eficiencia es lo que representa el desempeño del producto de software relativo a la cantidad de recursos que utiliza el software" [24], esto implica que se pueden realizar pruebas y sacar conclusiones acerca del producto si beneficia o por el contrario perjudica.

1.3. Marco legal

Esta investigación se sustenta legalmente en el artículo 322 de la Constitución de la republica del Ecuador, capitulo sexto: trabajo y producción, sección segunda: tipos de propiedad, en donde se especifica que "se reconoce la propiedad intelectual de acuerdo con las condiciones que señale la ley. Se prohíbe toda forma de apropiación de conocimientos colectivos, en el ámbito de las ciencias, tecnologías y saberes ancestrales" [25]. Esta investigación se hace alrededor de tecnologías de libre acceso a las cuales los lectores podrán acceder sin inconvenientes es necesario apoyarse en el artículo 142 tecnologías libres. Este se puede encontrar en el código orgánico de economía social de los conocimientos o COESC por sus siglas. El artículo se ubica en la sección 5: Disposiciones especiales sobre ciertas obras y en el apartado Segundo: De las tecnologías libres y formatos abiertos en el cual "se entiende por software de código abierto al software en cuya licencia el titular garantiza al usuario el acceso al código fuente y lo faculta a usar dicho software con cualquier propósito" [26]. Dentro del plan nacional del buen vivir en el apartado de "Propuesta de metas para homologación de indicadores y construcción de información" [27], se hace mención a la importancia de tener una buena

infraestructura para mejorar el sector tecnológico lo cual tiene relación con esta investigación ya que se busca de mejorar la administración de las infraestructuras.

Dentro de la Gobernanza de Ecuador en Internet [28], se encuentran el apartado de infraestructura en donde se encuentra el apartado de “Nube Informática” en el cual se explican conceptos claves que abordan esta investigación, fuera de este apartado también se tienen en cuenta términos comunes en una infraestructura, como protocolos, DNS e IP.

Estas normativas legales son importantes en la investigación dado que por medio de estas se está demostrando que se respeta el trabajo de todos los autores que se mencionan en cada uno de los apartados de esta investigación. Todas las herramientas usadas y presentadas son de libre acceso (Software libre) por lo cual es importante apoyarse en normativas legales que sustenten estas herramientas de libre acceso.

Capítulo 2

2. METODOLOGÍA

2.1. Delimitación espaciotemporal

Esta investigación no cuenta con una delimitación espacial debido a que es una investigación netamente experimental y puede ser replicado en cualquier institución. La delimitación temporal está entre los últimos 5 años (2015-2020), en los que se ha recopilado información que se ha publicado durante estos años y es la que se utilizó para desarrollar la investigación.

Por otro lado, esta investigación se realizó en el segundo semestre del año 2020 debido a que se necesitaba tiempo para preparar las herramientas, contratar los proveedores en los cuales se realizó la práctica y todos los requerimientos necesarios para obtener los resultados que se presentaron en la investigación.

2.2. Tipo de Investigación

Esta investigación es de tipo cualitativa y cuantitativa; es decir híbrida, ya que en la primera parte se cualifica y se analiza la herramienta Terraform basándose en datos y las experiencias de autores que ya han puesto en práctica la herramienta en diferentes situaciones. En la segunda parte de la investigación, cuando se experimentó con Terraform poniendo aprueba la efectividad que tiene en diferentes ecosistemas de la nube, es cuando entra el lado cuantitativo ya que se evaluaron tiempos de ejecución, efectividad al momento de desplegar, entre otros aspectos; es decir información que se puede medir.

2.3. Métodos

La investigación se llevó a cabo con el método experimental dado que se empleó Terraform en los diferentes proveedores, con el fin de realizar pruebas, medir tiempos de respuesta, calificar aspectos de usabilidad entre otros más, todo esto para realizar una correcta evaluación de la herramienta. El método experimental tiene como objetivo identificar causas y evaluar sus efectos por lo cual es importante para esta investigación debido a que llega a una situación práctica donde se necesita de un experimento.

2.4. Población y muestra

Para esta investigación se consideró como muestra los proveedores de servicios de la nube se escogió a los 3 más usados actualmente ya que estos son mencionados en muchos libros, artículos y conferencias por lo cual es un indicador de que son los más populares,

estos son: Amazon Web Services, Google Cloud Platform y Microsoft Azure, de esta forma la población y la muestra son lo mismo

2.5. Variables e indicadores

Para llevar a cabo la investigación se identificaron las variables las cuales son un total del 4, con sus respectivos indicadores los cuales conforman un total de 3 en la Tabla 1 Variables e indicadores sujetos a estudio, se visualizan tanto el tipo de variable, las entidades las cuales van desde el servidor en primer lugar seguido de las personas así mismo están las variables e indicadores.

Tabla 1 Variables e indicadores sujetos a estudio

VARIABLES	INDICADORES	TIPO DE VARIABLE	ENTIDADES
Rendimiento	Tiempo de ejecución Tiempo de despliegue Tiempo de desarrollo	Cuantitativa	Servidor
Interoperabilidad	Proveedores Alcance Estándares	Cualitativa	Servidor
Usabilidad	Satisfacción Curva de aprendizaje Interfaz Eficacia Eficiencia	Cualitativa	Personas
Integración	Compatibilidad Complementos	Cualitativa	Personas

Las variables listadas en la **¡Error! No se encuentra el origen de la referencia.**, son d escritas a continuación al igual que sus respectivos indicadores:

Variable Rendimiento. Se refiere al desempeño de la herramienta Terraform cuando se la está empleando si esta es lenta, rápida o está en un punto intermedio.

Indicadores

- Tiempo de ejecución. El tiempo en que la herramienta toma las ordenes que se le han dado y las ejecuta.
- Tiempo de despliegue. Tiempo en que la herramienta se conecta con el proveedor y crea las instancias y las aprovisiona.

Variable Interoperabilidad. Hace referencia a la cuan operable es la herramienta entorno a diferentes situaciones.

Indicadores

- Proveedores. Cuan compatible es la herramienta y cuanto soporte tiene con respecto a los diferentes proveedores.
- Alcance. La capacidad para automatizar procesos que tiene la herramienta por ejemplo si es capaz de configurar los DNS.
- Estándares. Los estándares que ya están definidos para realizar una correcta implementación de la herramienta.

Variable Usabilidad. Si la herramienta tiene una sintaxis fácil de entender y de recordar para los usuarios.

Indicadores

- Satisfacción. Si el usuario siente que la usar la herramienta es fácil, y satisface sus necesidades.
- Curva de aprendizaje. El manejo de la herramienta es fácil de aprender por los usuarios.
- Interfaz. Si es intuitiva para los usuarios y fácil de usar.
- Eficacia. Si es rentable usarla, o si consume muchos recursos.

Variable Integración. Facilidad para ser usada dentro de entornos de CI/CD o en conjunto con otras herramientas.

Indicadores

- Compatibilidad. Es fácil de usar en conjunto con otras herramientas que complementen su uso.
- Complementos. Cuenta con plugins que permitan utilizarla con herramientas como Jenkins.

2.6. Técnicas e instrumentos de recolección de datos

Existen varias técnicas para recolectar datos, en esta investigación se adoptó el experimento y esta técnica se utilizó para obtener datos por lo cual el proceso consistió en crear instancias en los diferentes proveedores de la nube con Terraform para medir la eficiencia de desempeño, calificar la usabilidad, la compatibilidad y entre otros aspectos. Para ello se planteó un esquema donde se detalla cómo se realizó la ejecución del experimento como se muestra en la Figura 3 Diagrama de ejecución, el diseño se plantea con el objetivo de obtener como respuesta las variables e indicadores que se usaron para evaluar la herramienta Terraform en los diferentes ecosistemas de la nube e identificar si su comportamiento cambia dependiendo de donde se lo está ejecutando. También se

puede visualizar que para cada uno de estos proveedores se utilizó la misma plantilla de esta forma no se afectaron los resultados ya que estuvieron en igualdad de condiciones.

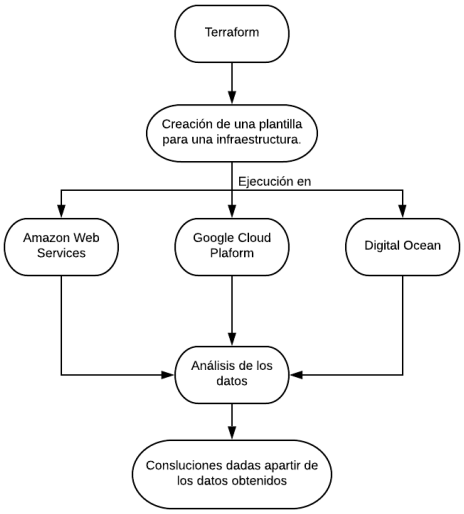


Figura 3 Diagrama de ejecución. Fuente: Elaboración propia

El diagrama empieza con Terraform que es la herramienta que se evaluó por lo tanto el siguiente paso fue definir y crear plantillas con la sintaxis HCL, esta plantilla se aplicó a los 3 proveedores con el fin de obtener resultados y evaluarlos usando las variables.

Las instancias que se crearon son simples debido a que constan de una maquina estándar con disco duro, memoria RAM (Random access memories) y procesador pero que a las cuales tendrán instalado un servidor http, para simular que esta se ha creado para desplegar una aplicación web.

Dependiendo de estos resultados se evaluó la herramienta para saber si se comporta de forma distinta por ejemplo si los tiempos de despliegue son más rápidos en AWS que en GCP y sacar las conclusiones pertinentes. La evaluación se realizó en base a la ficha de evaluación que se detalla en el Anexo 1 Ficha de evaluación, la cual está basada en algunas características que se encuentran en la ISO 25010, entre los cuales están Usabilidad, Eficiencia de desempeño y Compatibilidad, dicho instrumento ha sido validado por expertos y sus firmas dando el aprobado a el instrumento se encuentra en el Anexo 2 Aprobación Instrumento por Mgt. Marc Grob, Anexo 3 Aprobación Instrumento por Susana Patiño y Anexo 4 Aprobación Instrumento por Evelin Flores.

2.6.1. Explicación de los ítems del instrumento de recolección de datos

Con el fin de determinar las limitaciones y el alcance de Terraform en ambientes de producción se realizó una ficha de evaluación la cual se puede apreciar en el Anexo 1 Ficha de evaluación con el fin de obtener datos que faciliten esta tarea. Las puntuaciones que se utilizaron para realizar una correcta ponderación de los resultados fueron las siguientes, la casilla alto equivale a un valor de 3, medio equivale a un valor de 2,bajo equivale a un valor de 1. La Tabla 2 Escala de ponderación de la Usabilidad se creó multiplicando el puntaje más alto en este caso 3 con el número de ítems que tiene el primer apartado los cuales son 5; de ahí se crea la escala para medir el grado de importancia.

Tabla 2 Escala de ponderación de la Usabilidad.

Escala	Significado	Grado de importancia
12-15	Alto grado de usabilidad	Alto
8-11	Grado de usabilidad medio	Medio
1-7	Grado de usabilidad bajo	Bajo

Los ítems usados para evaluar la usabilidad de Terraform en cada uno de los proveedores son los que se encuentran en la Tabla 3 Ítems para evaluar la Usabilidaddonde se puede apreciar que se evalúa la facilidad con la cual se aprende a usar la herramienta dependiendo del proveedor, ya que existe configuración que cambia, al igual que la sintaxis HCL, el nivel de documentación por parte de HashiCorp es muy importante debido a que una herramienta sin una buena documentación no sirve de mucho ya que se dificulta mucho el aprendizaje de esta así mismo una herramienta CLI simple y fácil de usar es muy importante ya que no debe haber retrasos al ejecutar los comandos, el nivel de adecuación funcional según la ISO 25010 [24] es la capacidad que tiene la herramienta para darle a entender al usuario y es verdaderamente lo que este necesita y por último la complejidad o simplicidad que adopta la sintaxis HCL dependiendo del proveedor.

Tabla 3 Ítems para evaluar la Usabilidad.

Marque con una X	**Bajo**	**Medio**	**Alto**
Facilidad con la que se aprende a usar la herramienta			
Nivel de Documentación (Por parte de HashiCorp)			
Nivel de simplicidad de la herramienta CLI			
Nivel de adecuación funcional			
Nivel de simplicidad de la sintaxis HCL			

La eficiencia de desempeño también es importante, ya que al fin y al cabo lo que se desea de una herramienta es que sea veloz realizando sus tareas y es por ello por lo que en el segundo apartado de la ficha de evaluación se encuentra esta característica en donde se miden los tiempos de ejecución en segundos tal y como se muestra en la Tabla 4 Ítems para evaluar la Eficiencia de desempeño, cabe recalcar que en algunos casos el tiempo de creación de los recursos es diferente al tiempo completo de la creación y hay diferencias de entre 1 y 2 segundos o más.

Tabla 4 Ítems para evaluar la Eficiencia de desempeño.

Tiempos de Ejecución (en Segundos)	**Tiempo**
Tiempo de creación del o los recursos	
Tiempo completo de la creación	
Tiempo de destrucción del recurso	
Tiempo de despliegue del servidor	

La compatibilidad entre herramientas de la misma índole es muy importante ya que estas pueden complementarse entre sí ya sea teniendo una buena interoperabilidad lo cual les permite usar los datos generados entre herramientas, para ello en el tercer apartado de la ficha de evaluación se evalúa la compatibilidad e integración de Terraform con otras herramientas dentro de los diferentes proveedores de la nube que se seleccionaron para esta investigación, de esta forma la Tabla 5 Ítems para evaluar la Compatibilidad (Integración). con los ítems refleja los aspectos que se evaluaron, cabe recalcar que los puntajes son los mismos que anteriormente se usaron.

Tabla 5 Ítems para evaluar la Compatibilidad (Integración).

Marque con una X	**Bajo**	**Medio**	**Alto**
Nivel de integración con otras herramientas IAC (Ansible, Chef, entre otros)			
Nivel de integración con entornos CI/CD			
Capacidad de usar las Imágenes de Maquina generadas con Packer			

Con el fin de obtener una escala con la cual se puedan ubicar los resultados para dar un veredicto de la Compatibilidad (Integración) de Terraform se creó una escala de ponderación similar a la anteriormente mostrada para el primer apartado de la ficha de evaluación dicha escala es la presentada en la Tabla 6 Escala de ponderación de la compatibilidad (Integración). y los valores de la escala se calcularon multiplicando 3 que es el puntaje más alto por el número de ítems lo cual da como resultado 9 y de ahí se crea la escala.

Tabla 6 Escala de ponderación de la compatibilidad (Integración).

Escala	Significado	Grado de importancia
8-9	Alto grado de compatibilidad	Alto
5-7	Grado de compatibilidad medio	Medio
1-4	Grado de compatibilidad bajo	Bajo

2.7. Técnicas de procesamiento y análisis de datos

Para esta investigación se tomaron en cuenta algunas técnicas para procesar y analizar los datos las cuales se han marcado con un "si" en la Tabla 7 Técnicas de análisis y procesamiento de datos que se muestra a continuación.

Tabla 7 Técnicas de análisis y procesamiento de datos

Simulación de Monte Carlos	**No**
Análisis de patentes y literatura científica	No
Experimentos A/B	No
Análisis de escenario	Si
Análisis de correlación	No
Predicción matemática	No
Análisis basado en estadística descriptiva	No
Visualización de datos	Si

Se tomó en cuenta el Análisis de escenario porque permite analizar varios eventos cuando las ejecuciones de Terraform fueron llevadas a cabo, así mismo la visualización de datos permite obtener gráficas con las cuales se pueden interpretar los datos de manera concisa, en este caso permitió interpretar los resultados de las ejecuciones de Terraform en los diferentes ecosistemas de la nube y de este modo visualizar pequeñas variaciones que puedan influyeron en las conclusiones finales de esta investigación.

2.8.Normas éticas

Esta investigación se sostiene en normativas éticas las cuales van de acuerdo con los lineamientos y reglamento de grados de la PUCESE, como se ha indicado en el apartado de normativas legales se respeta el trabajo de cada uno de los autores que se han nombrado en esta investigación, dándole el reconocimiento que se merecen por las ideas y conceptos que se utilizan en esta investigación. Las herramientas que se utilizaran en esta investigación son de acceso libre y se respeta los términos y las condiciones que traen consigo las licencias del software que se usa.

Capítulo 3

3. Resultados

3.1. Análisis e interpretación de los resultados

Los resultados obtenidos al realizar las pruebas en los diferentes ecosistemas de la nube se presentan de forma cronológica en la cual se explican los procedimientos y configuraciones más relevantes al igual que el código que se utilizó en cada uno de los ecosistemas. Las salidas obtenidas al final de cada ejecución son presentadas en forma de imagen de modo que se puede observar claramente los resultados, aunque posteriormente son explicados, cabe recalcar que se utilizó la herramienta Packer en conjunto con la herramienta Terraform para realizar las pruebas ya que esta herramienta permite agilizar la creación de imágenes de máquina.

3.1.1. Alcance y limitación técnica de Terraform en ambientes de producción.

Usabilidad, cada uno de los puntajes obtenidos por cada uno de los proveedores se ven reflejados en el Gráfico 1 Representación de los puntajes obtenidos al evaluar la Usabilidad. en donde AWS tuvo un puntaje de 13 lo cual lo sitúa en un alto grado de usabilidad, GCP obtuvo un puntaje 15 lo cual también lo convierte en un proveedor en donde Terraform tiene un alto grado de usabilidad y por último Azure obtuvo una puntuación de 10 lo cual le da un grado medio de usabilidad.

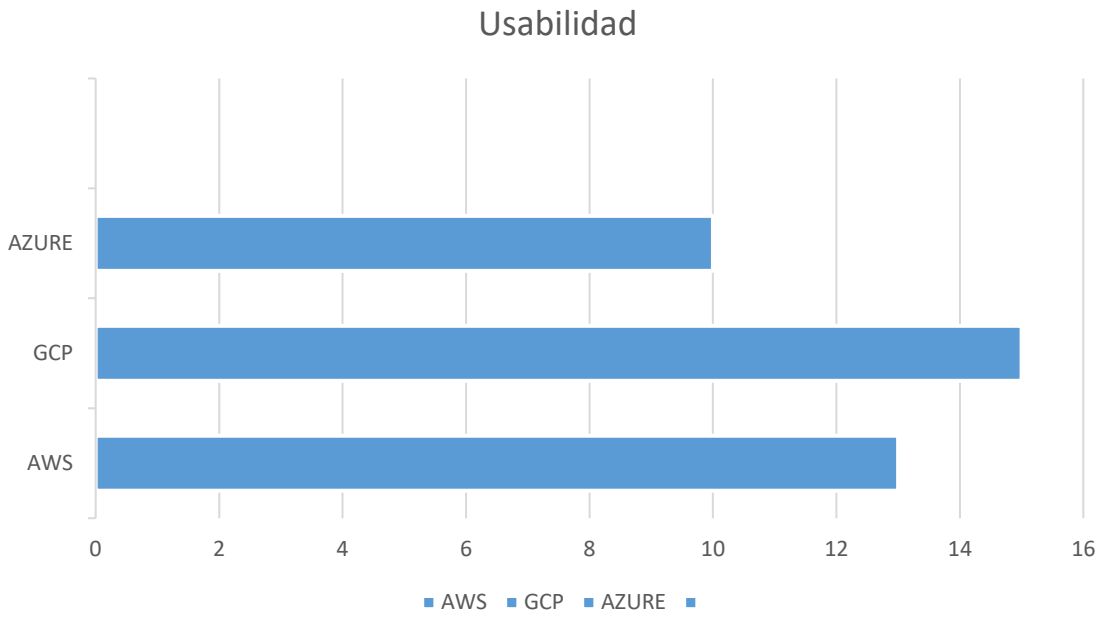


Gráfico 1 Representación de los puntajes obtenidos al evaluar la Usabilidad.

Es notable que la herramienta es mucho más fácil de usar en GCP y esto es porque la sintaxis es mucho más simple, las configuraciones mucho más fáciles, posee una documentación bastante clara, no hay errores con respecto al cliente en donde se ejecutan los comandos y la propia herramienta tiene la capacidad para indicarle al usuario si es lo que está buscando o no.

A partir de estos datos se puede deducir que el soporte de cada proveedor hacia Terraform y de la empresa HashiCorp hacia cada proveedor es un factor importante ya que mientras esta relación sea buena y se vaya mejorando con el tiempo, el alcance de Terraform será mayor ya que el proveedor le dejará acceder a muchas más APIs para así tener un mayor alcance y personalización en los recursos que se pueden automatizar.

En este caso la limitación de Terraform va directamente ligada al proveedor, ya que si este ya tiene su propia herramienta para automatizar la creación de recursos mediante la infraestructura como código y quiere que sus usuarios usen más la herramienta propia del proveedor este empezará a limitar el acceso a algunas APIs y reducirá la capacidad para personalizar un recurso, en este caso el único que tiene una herramienta propia es AWS, la cual es Herramientas para el desarrollo de Infraestructura como código, que ya ha sido mencionada en el primer Capítulo 1 de esta investigación.

Eficiencia de desempeño, en este apartado se midieron los tiempos de cada uno de los proveedores los cuales son plasmados en la Tabla 8 Resultados de la evaluación de la

Eficiencia de desempeño en donde el número de ítem hace alusión al orden en el cual están las mediciones de la Tabla 4 Ítems para evaluar la Eficiencia de desempeño, cabe recalcar que la unidad de medida empleada son los segundos.

Tabla 8 Resultados de la evaluación de la Eficiencia de desempeño

Ítem	AWS	GCP	AZURE
1	25s	18s	79s
2	25s	18s	82s
3	34s	16s	47s
4	50s	25s	120s

Tal y como muestran los resultados GCP es el proveedor en el cual Terraform ha conseguido un mejor desempeño durante la creación de los recursos, la eliminación de los mismos y el tiempo de despliegue del servidor que estaba alojado en la instancia recién creada, claramente es un indicador de lo pulido que esta la integración entre proveedor y herramienta, a diferencia de Azure el cual obtuvo el peor puntaje de todos demorando incluso 2 minutos en que se pueda observar el despliegue del servidor que estaba en la instancia, esto se debe mayormente a que para poder crear una instancia que pueda tener un servidor Http se necesitan de varios recursos lo cual demuestra lo mal que esta la relación herramienta – proveedor, esta puede ser una limitación ya que no sirve de nada utilizar Terraform en un proveedor donde es tan laborioso crear algo tan básico como una instancia de máquina virtual, por otro lado AWS se mantiene en medio de estos dos con resultados buenos que no están al nivel de GCP pero tampoco tan malos como Azure, aunque si le tomo el doble de tiempo que a Google en poder desplegar el servidor Http.

Compatibilidad, los puntajes obtenidos por cada uno de los proveedores en los que se implementó Terraform y en los que se juzgó la compatibilidad entre Terraform y otras herramientas, son los siguientes: AWS con un puntaje de 9 lo cual le da un alto grado de compatibilidad, GCP con 9 al igual que el anterior tiene un alto grado de compatibilidad y Azure con un puntaje de 8 lo cual aunque lo deja con un alto grado es con menos puntaje que los anteriores proveedores; estos resultados se puede apreciar de una mejor forma en el Gráfico 2 Representación de los puntajes obtenidos al evaluar la Compatibilidad.

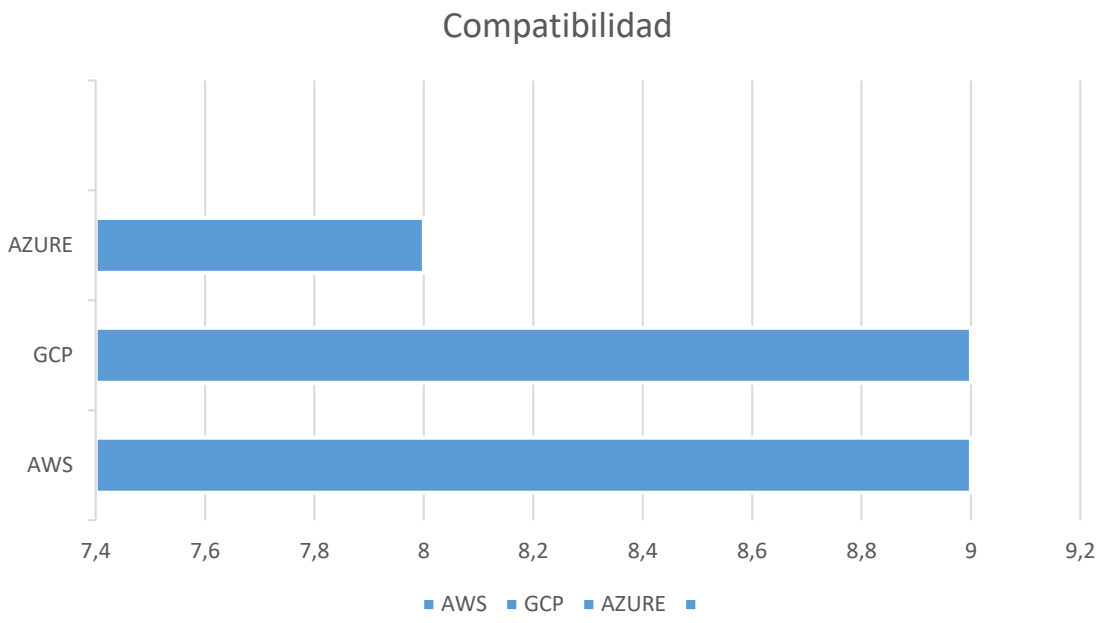


Gráfico 2 Representación de los puntajes obtenidos al evaluar la Compatibilidad.

Un punto importante a favor de AWS es que, a pesar de tener una herramienta propia, sigue manteniendo una buena compatibilidad con herramientas de terceros y en el caso de tener instancias creadas con su herramienta propia, estas no interfieren con instancias creadas con Terraform, en el caso de GCP y Azure no cuentan con herramientas propias de IAC.

Luego de haber obtenido todos estos resultados, fue más claro definir el alcance y las limitaciones técnicas de Terraform en ambientes de producción como los proveedores usados para esta investigación; el alcance de la herramienta está fuertemente ligado al proveedor ya que Terraform puede gestionar los recursos declarados en los scripts pero únicamente si el proveedor le concede permiso a sus diferentes APIs, afortunadamente la mayoría de proveedores si llegan a conceder estos permisos y es por eso que Terraform es ampliamente usado, especialmente en los 3 proveedores en los que se lo ha probado a lo largo de esta investigación, el alcance de Terraform llega hasta el tipo de recurso que el proveedor le permita automatizar.

Como se mencionó anteriormente estas limitaciones están profundamente ligadas al proveedor y más aún a aquellos que tienen sus propias herramientas basadas en el concepto de IAC, ya que si estos obligan a sus usuarios a usar sus propias herramientas ya están limitando a Terraform, debido a que es probable que dejen de brindar soporte o empiecen a limitar la capacidad de automatización, la gestión de sus servicios también es otro punto que limita debido a que si estos no tienen una gestión correcta de sus servicios

también limita la funcionalidad y su capacidad de uso, es aquí donde entra la optimización que le da cada proveedor a sus servicios para que sean mucho más fáciles de automatizar, esto se puede ver claramente entre GCP y Azure el primero tiene muy pulidos sus servicios y con esto es mucho más fácil utilizar Terraform ya que no se encuentra con grandes limitaciones, el segundo no tiene tan pulidos sus servicios y más aún la documentación propia que ofrece es pobre y confusa por lo cual limita a los usuarios y a la herramienta ya que no se la puede usar a su máxima capacidad.

3.1.2. Aplicación de la herramienta Terraform en ecosistemas heterogéneos de computación en la nube y su comportamiento.

AWS

Amazon Web Services es el primer proveedor o ecosistema en el cual se realizaron las pruebas y para esta práctica en específico hay dos opciones, crear un AMI (Amazon Machine Image) el cual es un tipo de imagen de maquina la cual ya puede estar configurada con software preinstalado, llaves SSH entre otras configuraciones para que al momento de crear la instancia esta ya se encuentre aprovisionada con lo necesario, la otra opción es utilizar una imagen básica que ofrece AWS, cabe recalcar que aparte escoger esa imagen es obligatorio pasar como parámetro "id" el dueño de la imagen que se va a usar todo esto se puede conseguir en la documentación oficial de AWS donde se explican que son las AMI como usarlas y entre más información que de momento no es relevante para esta práctica.

Packer es una herramienta creada por HashiCorp de la cual ya se habló en el marco teórico de esta investigación la cual permite crear un AMI de forma sencilla usando archivos JSON como se presenta en la Figura 4 Script de Packer para crear un AMI en AWS. donde se puede observar un poco de la sintaxis que se utiliza al crear estos scripts, se declara que el tipo es amazon-ebs para indicarle que es en Amazon donde se realizarán las acciones del script y muy importante se especifica la imagen base con la cual se crea un AMI ya aprovisionada, en este caso fue un Ubuntu-xenial-16.04 la cual se escogió debido a que era la única que no presentaba errores y se encontraba estable, algo importante a recalcar cuando ya hay nuevas versiones con soporte, toda esta configuración se encuentra dentro de los *Builders* en donde también se deberían especificar las credenciales de acceso que Amazon recomienda no incluirlas en el código si no que configurarlas en la herramienta CLI que ellos proporcionan para una mayor seguridad que fue lo que se hizo.

```
{
  "builders": [
    {
      "type": "amazon-ebs",
      "region": "us-east-2",
      "source_ami_filter": {
        "filters": {
          "virtualization-type": "hvm",
          "name": "ubuntu/images/*ubuntu-xenial-16.04-amd64-server-*",
          "root-device-type": "ebs"
        },
        "owners": ["099720109477"],
        "most_recent": true
      },
      "instance_type": "t2.micro",
      "ssh_username": "ubuntu",
      "ami_name": "packer-example {{timestamp}}",
      "security_group_id":"sg-09b2677c100d36a35"
    }
  ],
  "provisioners": [
    {
      "type": "shell",
      "script": "scripts/docker-install.sh"
    }
  ]
}
```

Figura 4 Script de Packer para crear un AMI en AWS. Fuente: Elaboración propia

Los *provisioners* es el apartado en donde como su nombre lo indica se procede a aprovisionar la imagen, en este caso se especifica que será una instrucción de tipo Shell y que el script a utilizar se encuentra en la carpeta scripts con el nombre de docker-install.sh el cual servirá de ejemplo para demostrar que al final de la ejecución se creará una imagen que tendrá Docker instalado el cual se escogió ya que es una herramienta que hoy en día se está usando mucho para realizar despliegue de aplicaciones con su tecnología de contenedores. Los comandos básicos para usar Packer son:

`Packer validate nombre-script.json` y `packer build nombre-script.json` el primero valida que no existan errores de sintaxis y el segundo ejecuta el script lo cual da una salida como la que se observa en la Figura 5 Salida por consola de la ejecución de packer en AWS. y Figura 6 Salida por consola de la ejecución de packer en AWS. donde se muestra que se usa una conexión SSH para aprovisionar la imagen, y ejecutar el script de instalación y sobre todo el tiempo que se demoró en crear la imagen que fueron 9 minutos con 47 segundos.

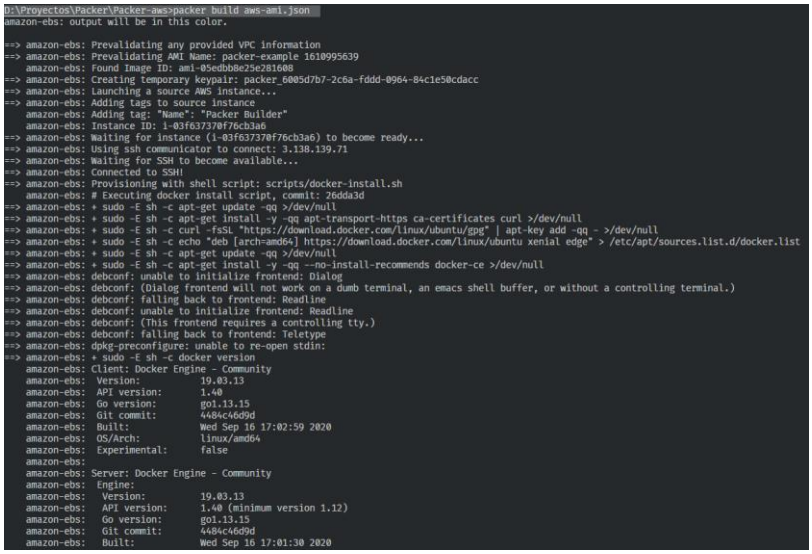


Figura 5 Salida por consola de la ejecución de packer en AWS. Fuente: Elaboración propia

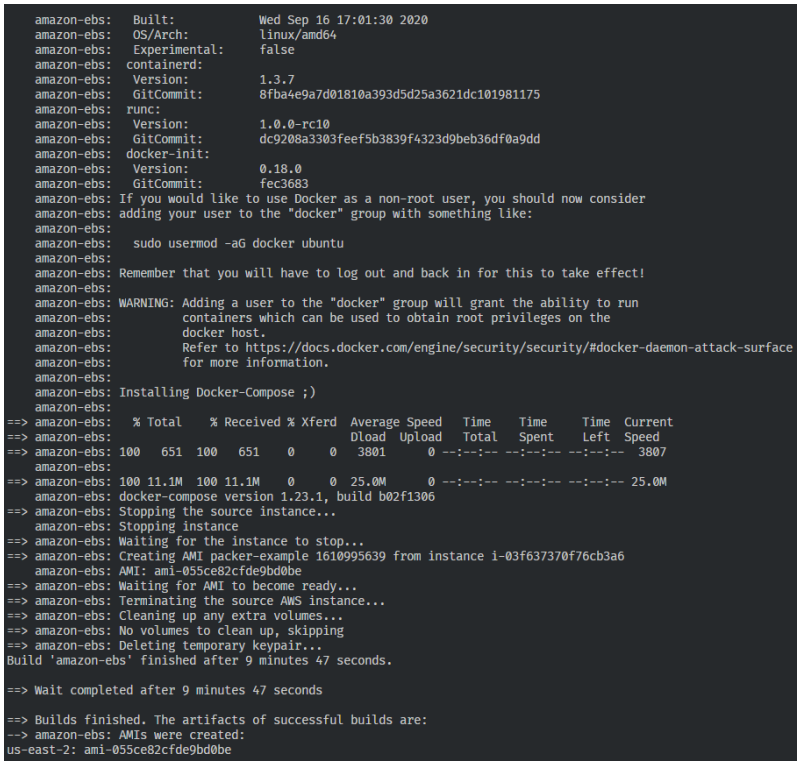


Figura 6 Salida por consola de la ejecución de packer en AWS. Fuente: Elaboración propia

Una vez creada la imagen se procede a crear el script de terraform los cuales usan una extensión TF y en la Figura 7 Script de Terraform para crear una instancia en AWS. se puede observar el código que se ha creado para levantar una instancia en AWS, lo primero que se realiza es declarar el *provider* y la región en la que se va a levantar la instancia, posterior a ello se declaran los recursos en este caso serán una instancia y un *security group*.

```
provider "aws" {
  region = "us-east-2"
}

resource "aws_instance" "practica-instance" {
  ami             = var.ami_id
  instance_type   = var.instance_type
  tags            = var.tags
  security_groups = [aws_security_group.ssh_conection.name]
  user_data = <<-EOF
        #!/bin/bash
        docker run -d -p 80:80 --name test nginx
      EOF
}

resource "aws_security_group" "ssh_conection" {
  name = var.sg_name

  dynamic "ingress" {
    for_each = var.ingress_rules
    content {
      from_port   = ingress.value.from_port
      to_port     = ingress.value.to_port
      protocol    = ingress.value.protocol
      cidr_blocks = ingress.value.cidr_blocks
    }
  }
  dynamic "egress" {
    for_each = var.egress_rules
    content {
      from_port   = egress.value.from_port
      to_port     = egress.value.to_port
      protocol    = egress.value.protocol
      cidr_blocks = egress.value.cidr_blocks
    }
  }
}

```

Figura 7 Script de Terraform para crear una instancia en AWS. Fuente: Elaboración propia

Como se puede ver se utilizan variables debido a que la configuración para realizar esta práctica era muy extensa por lo cual se decidió separar en varios archivos los cuales tienen como nombre variables.tf que es en donde se declaran las variables y se les pone un valor por defecto, prod.tfvars que es donde se ingresan los valores de las variables y output.tf que es donde se declara que es lo que se desea retornar luego de la ejecución en este caso se levanta una instancia así que lo ideal es retornar la IP de la instancia creada, este archivo se pueden observar en el Anexo 5 Script de declaración de variables. y las variables en él Anexo 6 Script de ingreso de valor para las variables.. El uso de variables trae ventajas tales como poder usar un *for each* para iterar elementos de una lista y no tener que escribir todo unido y tener como resultado un código poco entendible, de esta forma se pretende tener un código de calidad.

Para ejecutar este script se necesitan ejecutar 3 comandos básicos los cuales son: `terraform init`, `terraform validate`, `terraform plan` y `terraform apply -var-file="prod.tfvars"`

El primer comando descarga todo lo necesario para poder realizar la ejecución en el proveedor especificado, el segundo valida la sintaxis HCL, el tercero realiza una

planeación de lo que se va a crear es decir que se planifica la ejecución en donde obviamente existirán campos vacíos ya que estos solo se generarán al momento de ejecutar y es aquí donde se puede revisar si alguna configuración se ha pasado por alto y el ultimo comando es el cual se encarga de ya levantar todos los recursos que se han especificado en el script en este caso se le pasa como argumento el nombre del archivo donde se encuentran especificados los valores de las variables caso contrario si no se le pasa un archivo de variables la herramienta CLI pedirá que se ingresen los valores de las variables por consola.

De esta forma se obtiene como resultado la creación de la instancia y el *security group* que permitirá el tráfico http por el puerto 80 y en la consola se podrá leer la salida que envía terraform la cual se encuentra en la Figura 8 Salida por consola de la ejecución de Terraform en AWS. donde se ve que se crearon 2 recursos los cuales son el *security group* y la instancia, no se modificó ningún recurso, no se destruyó ningún recurso y se obtiene como salida la IP de la instancia recién creada junto con el tiempo total que se demoró en realizar todas las acciones el cual fue de 25 segundos.

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

Changes to Outputs:
  + instance_ip = [
      + (known after 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

aws_security_group.ssh_conection: Creating...
aws_security_group.ssh_conection: Creation complete after 7s [id=sg-0300c5664ff4c069f]
aws_instance.practica-instance: Creating...
aws_instance.practica-instance: Still creating... [10s elapsed]
aws_instance.practica-instance: Still creating... [20s elapsed]
aws_instance.practica-instance: Creation complete after 25s [id=i-0eb57a1825a438f4c]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

instance_ip = [
  "18.189.11.104",
]
```

Figura 8 Salida por consola de la ejecución de Terraform en AWS. Fuente: Elaboración propia

Una vez hecho esto se puede acceder a la IP y deber aparecer el mensaje de bienvenida de NGINX ya que se especificó que al crear la instancia ejecute un comando que levante un contenedor de Docker con el servidor antes mencionado en el puerto 80 y en la consola de AWS en el apartado de instancias debe aparecer la instancia creada.

Si no se desea seguir con la instancia que está consumiendo recursos y por lo tanto seguir generando gastos se puede ejecutar el comando `terraform destroy` el cual destruirá todos los recursos creados con terraform ya que al contar con un archivo llamado

terraform.tfstate donde se están guardando las acciones realizadas Terraform sabe exactamente que recursos ha creado y que es lo que debe eliminar o en caso de actualizar sabe que recursos debe actualizar, al realizar esta acción se obtiene una salida como la de la Figura 9 Salida por consola de la eliminación de los recursos creados con terraform. en donde dice que no se han creado ni actualizado recursos pero si se han eliminado 2 recursos y por tanto la IP de la instancia pasa a ser *null* ya que dicha instancia ya no existe.

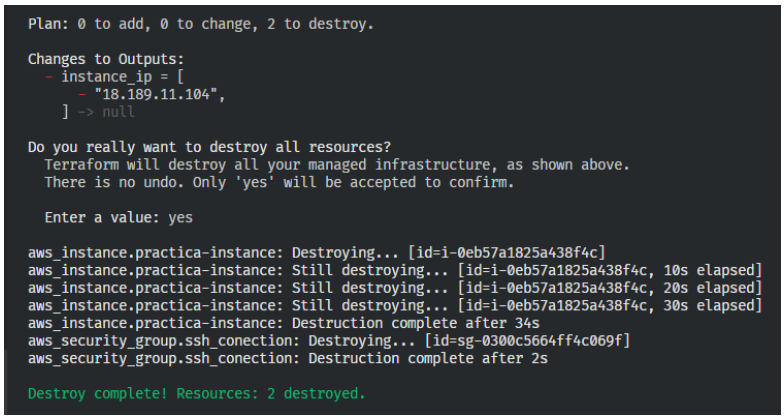
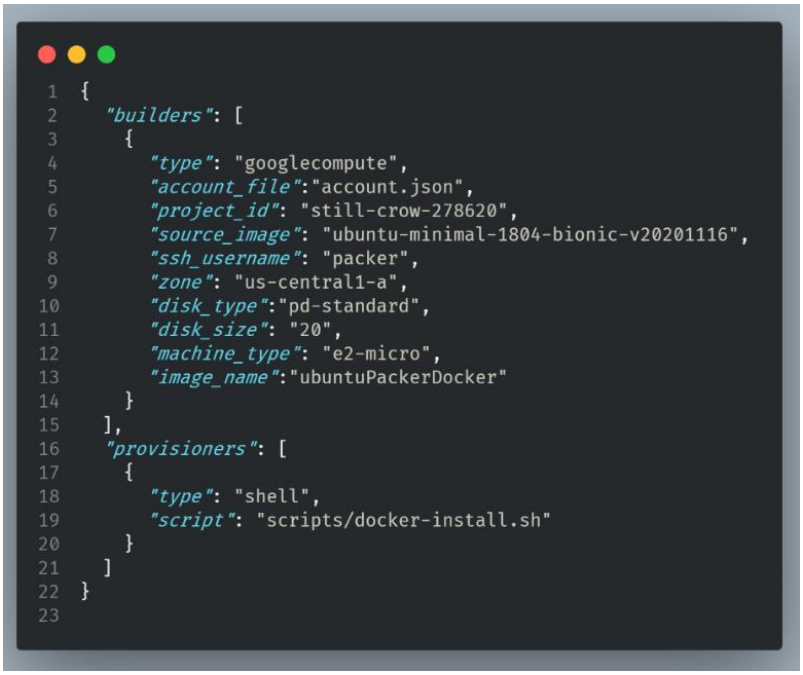


Figura 9 Salida por consola de la eliminación de los recursos creados con terraform. Fuente: Elaboración propia

GCP

El segundo proveedor de la nube es Google Cloud Platform, en el cual nuevamente se realizaron pruebas con el mismo fin que en el anterior proveedor, crear una imagen de maquina con Ubuntu y aprovisionarle Docker para ello es utilizada la herramienta Packer y el script para poder realizar estas acciones es el que se presenta en la Figura 10 Script de Packer para GCP en donde lo más resaltante es que existe un argumento *account_file* que es donde se especifica el archivo en formato JSON que se obtiene haciendo configuraciones en la consola del proveedor que es de los únicos procedimientos manuales que se deben realizar previo a la automatización con Packer y Terraform, este archivo es lo que permite autenticarse con el API de Google para poder crear la instancia temporal que tendrá instalado un sistema operativo Ubuntu, el tipo de maquina es una e2-micro la cual es lo suficiente en potencia para que se le pueda instalar Docker mediante un script.



```
{
  "builders": [
    {
      "type": "googlecompute",
      "account_file":"account.json",
      "project_id": "still-crow-278620",
      "source_image": "ubuntu-minimal-1804-bionic-v20201116",
      "ssh_username": "packer",
      "zone": "us-central1-a",
      "disk_type":"pd-standard",
      "disk_size": "20",
      "machine_type": "e2-micro",
      "image_name":"ubuntuPackerDocker"
    }
  ],
  "provisioners": [
    {
      "type": "shell",
      "script": "scripts/docker-install.sh"
    }
  ]
}
```

Figura 10 Script de Packer para GCP. Fuente: Elaboración propia

Cabe recalcar que debido a la forma en cómo se maneja GCP hay que indicar el id del proyecto ya que pueden crearse instancias en un proyecto que no tengan nada que ver con las instancias en otro proyecto y por lo cual es necesario indicar el id del proyecto, al igual que la zona en la cual se va a crear.

Al ejecutar el comando `Packer build ami-gcp.json` se empezará a crear la máquina virtual en la cual se realizarán las acciones declaradas en el script y tal como se muestra en la Figura 11 Salida por consola de la ejecución del script al terminar de realizar las acciones en este caso la última acción es la instalación de Docker-compose, se procede a eliminar la instancia y a crear la imagen que es lo importante en este caso ya que enseguida estará disponible para poder ser usada, luego de hecho elimina el disco que utilizó ya que si no lo elimina este podría estar generando gastos por mantenimiento y esto se refleja en la facturación final de GCP.

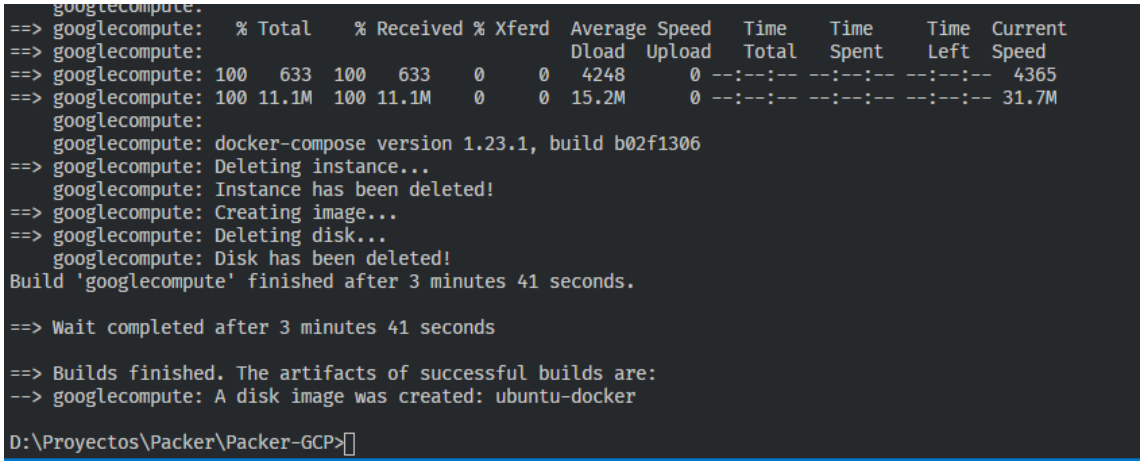
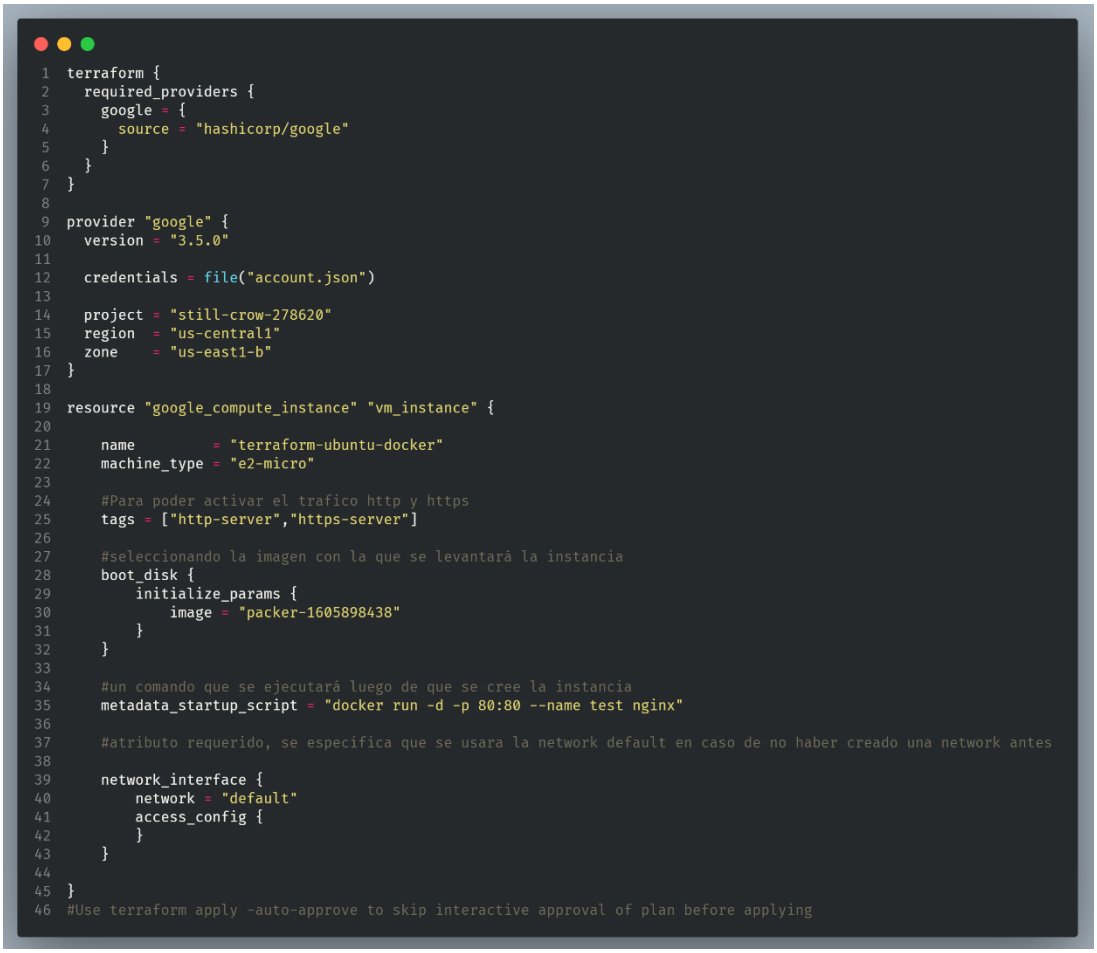


Figura 11 Salida por consola de la ejecución del script. Fuente: Elaboración propia

Todo este proceso tomó 3 minutos y 41 segundos lo cual es dos veces menos tiempo que el que le tomo crear una imagen en AWS, de este modo ya existe una idea de cuál de los proveedores funcionan de mejor forma con las herramientas de HashiCorp, esto puede deberse a muchas razones como el mantenimiento de la API del proveedor o más actualizaciones por parte de la empresa dueña de la herramienta específicamente para GCP, lo que es innegable es que el tiempo en el cual se realizó este proceso es sustancialmente mejor que en el anterior proveedor.

Con la imagen lista lo siguiente es crear un script en Terraform que la utilice, para ello lo primero que se debe realizar es declarar que el *providers* que se utilizara es el de Google y que el origen es *hashicorp/google* lo cual es importante ya que existen varias opciones disponibles en la documentación oficial de la herramienta, pero en este caso se utilizará la oficial que es mantenida por la propia organización dueña de la herramienta, luego de ello hay que pasar varios parámetros de forma obligatoria como la versión que se va a utilizar, las credenciales que están en el archivo JSON anteriormente usado, el proyecto, la región y la zona. Una vez configurado esto se procede a crear los recursos en este caso una *google_compute_instance* como se puede apreciar en la Figura 12 Script de Terraform para GCP, se le asigna un nombre a la instancia y el tipo de maquina en este caso nuevamente es una e2-micro, muy importante especificar los *tags* como están en el script para que esta instancia permita el tráfico http y https, caso contrario no se podrá acceder al servidor que se levantara en ella.



```
terraform {
  required_providers {
    google = {
      source = "hashicorp/google"
    }
  }
}

provider "google" {
  version = "3.5.0"

  credentials = file("account.json")

  project = "still-crow-278620"
  region  = "us-central1"
  zone    = "us-east1-b"
}

resource "google_compute_instance" "vm_instance" {

    name         = "terraform-ubuntu-docker"
    machine_type = "e2-micro"

    #Para poder activar el trafico http y https
    tags = ["http-server","https-server"]

    #seleccionando la imagen con la que se levantará la instancia
    boot_disk {
        initialize_params {
            image = "packer-1605898438"
        }
    }

    #un comando que se ejecutará luego de que se cree la instancia
    metadata_startup_script = "docker run -d -p 80:80 --name test nginx"

    #atributo requerido, se especifica que se usara la network default en caso de no haber creado una network antes

    network_interface {
        network = "default"
        access_config {
        }
    }

}
#Use terraform apply -auto-approve to skip interactive approval of plan before applying
```

Figura 12 Script de Terraform para GCP. Fuente: Elaboración propia

Un aspecto importante es el de especificarle a esta instancia la imagen que usará como sistema operativo, en este caso es la imagen que se creó anteriormente con Packer, muy importante y donde se puede apreciar de mejor forma la diferencia entre algunas palabras reservadas de HCL para cada proveedor es en el *metadata_startup_script* que es el comando específico para Google para poder lanzar un comando luego de que la instancia ha sido creada en este caso se levanta un contenedor de Docker con NGINX en el puerto 80 y muy importante declarar la network_interface que es obligatoria y en caso de no tener ninguna creada se debe especificar que se utilizará la que está por defecto.

Una vez configurado todo esto, se debe correr el comando `terraform init` el cual va a descargar todo lo necesario para que Terraform se conecte con Google y cree el recurso que se le ha indicado en el script, luego de ello se ejecuta el comando `terraform plan` para tener una planificación de todo lo que se va a crear y verificar posibles errores de lógica que pueda tener el script, con un `terrafom apply -auto-approve` se creará el recurso que en este caso únicamente se trata de una instancia y no pedirá una confirmación ya que se le especificó que se auto apruebe al lanzar el

comando, esto da una salida como la que se puede observar en la Figura 13 Salida por consola de Terraform en GCP..

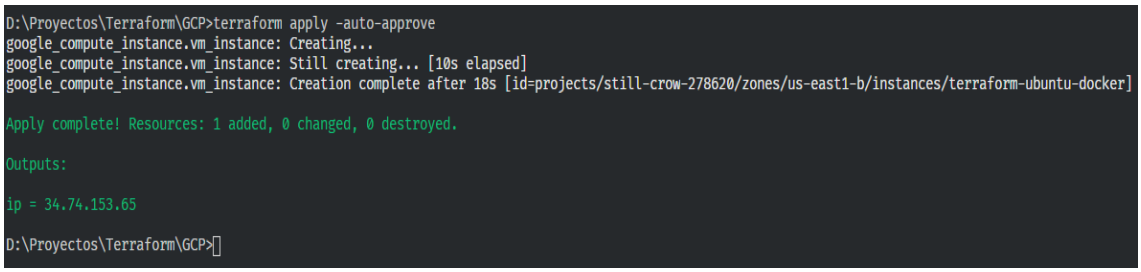


Figura 13 Salida por consola de Terraform en GCP. Fuente: Elaboración propia

Dado a que solo se necesita crear un único recurso para crear una instancia que pueda correr un servidor NGINX la ejecución es muy rápida ya que solo le toma 18 segundos lo cual es bastante veloz considerando el tiempo que toma crear manualmente esta misma instancia, pero interactuando con la consola del proveedor. Gracias al archivo *outputs.tf* que se puede observar en el Anexo 7 Archivo Outputs.tf para obtener la ip de la instancia recién creada. se obtiene la IP de la instancia recién creada para poder acceder a ella y verificar que el servidor está funcionando de forma correcta.

Una vez realizada esta práctica ya no es necesario tener esta instancia corriendo y consumiendo recursos que pueden generar costos, para lo cual existe el comando `terraform destroy` el cual únicamente destruye las instancias creadas con terraform dejando todas las instancias que no se han creado con la herramienta intactas, de esta forma no se interfiere con otros recursos e instancias, en la Figura 14 Salida por consola de Terraform destroy en GCP se puede ver como no toma mucho tiempo en eliminar todo lo que ha creado.

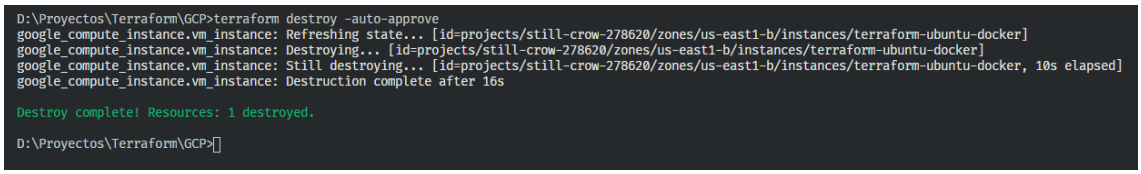


Figura 14 Salida por consola de Terraform destroy en GCP. Fuente: Elaboración propia

Azure

El ultimo proveedor en donde se realizaron las pruebas fue Azure el cual a pesar de ser propiedad de Microsoft ya permite crear instancias que utilicen distribuciones basadas en Linux como por ejemplo Ubuntu, en este caso lo primero que se debe de hacer es bajar el

cliente de Azure en la computadora donde se ejecutará Terraform y Packer ya que este ayuda a obtener información para poder conectarse a la API del proveedor, con las credenciales listas, lo siguiente es crear el script tal y como se ve en la Figura 15 Script de Packer para Azure en donde se declaran las credenciales en las variables, estas credenciales están censuradas por medidas de seguridad.

Figura 15 Script de Packer para Azure. Fuente: Elaboración propia

Como se puede observar claramente este script es mucho más extenso que los anteriores creados para los 2 proveedores con los que anteriormente se realizaron las mismas pruebas, en el apartado de *Builders* se declara el tipo el cual tiene que ser un *azure-arm* y luego pasar el valor de las variables que anteriormente se han declarado. El sistema operativo que se escogió es el mismo que se ha usado en las anteriores pruebas, un Ubuntu server, todo esto se especifica en el script inclusive quien es el que publica el sistema operativo y la versión que se desea usar; un aspecto muy importante indicar el nombre del grupo de recursos en donde se creara la instancia y donde se almacenara la imagen creada a partir de esta instancia, el nombre que tendrá la imagen el cual debe ser único, luego de ello se especifican algunos tags que no tienen mucha relevancia y la locación en donde se desplegara y el tipo de máquina que se usará. Como es costumbre en todas estas pruebas se aprovisionará esta imagen con Docker usando un script; ejecutando el

comando `packer build ami-azure.json` se obtiene una salida por consola como la que se aprecia en la Figura 16 Salida por consola de Packer en Azure

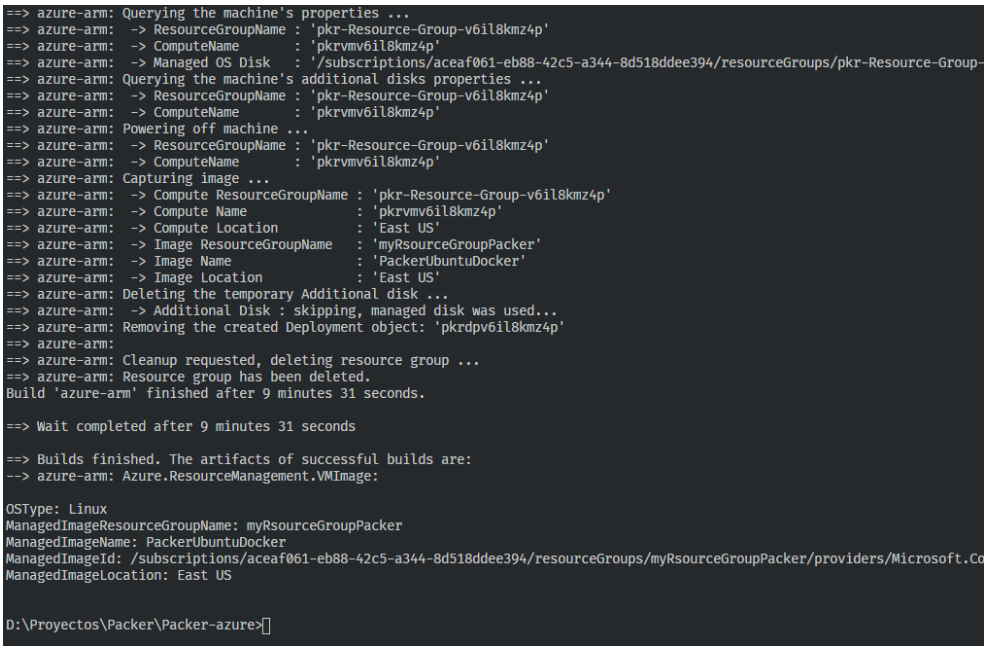


Figura 16 Salida por consola de Packer en Azure. Fuente: Elaboración propia

El tiempo que le tomó realizar todas estas acciones fue muy parecido al que tuvo AWS ya que fueron 9 minutos con 31 segundos y en comparación con GCP es el doble de tiempo, en la salida también se especifica que el resultado es un Sistema Operativo Linux, el Id de la imagen, la locación y el nombre de la imagen estos datos son muy importantes para poder usar esta imagen con Terraform.

El script de Terraform para poder crear una instancia que pueda realizar las mismas acciones que los anteriores scripts para los diferentes proveedores, difiere bastante ya que se necesitan crear mucho más recursos lo cual se transforma en una desventaja ya que el tiempo de creación de estos recursos será mayor, aunque esto ya depende completamente del proveedor que en este caso es Azure, el script se ha divido en dos secciones ya que es bastante extenso y la primera es la Figura 17 Primer sección del script de Terraform para Azure donde se especifica el proveedor y la versión que se usará. Luego de especificar el proveedor ya se declaran todos los recursos que se emplearán en este caso lo primero es crear un grupo de recursos en donde se alojaran las instancias creadas con Terraform, esto incluye el lugar el cual en este caso fue el Este de Estados Unidos.

```
terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = ">= 2.26"
    }
  }
}

provider "azurerm" {
  features {}
}

resource "azurerm_resource_group" "example" {
  name     = "TerraformResourceGroup"
  location = "eastus"
    tags = {
        environment = "Terraform test"
    }
}

resource "azurerm_virtual_network" "example" {
  name                = "example-network"
  address_space       = ["10.0.0.0/16"]
  location            = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
}

resource "azurerm_public_ip" "publicip" {
  name                = "myTFPublicIP"
  location            = "eastus"
  resource_group_name = azurerm_resource_group.example.name
  allocation_method   = "Static"
}

# Create Network Security Group and rule
resource "azurerm_network_security_group" "nsg" {
  name                = "myTFNSG"
  location            = "eastus"
  resource_group_name = azurerm_resource_group.example.name

  security_rule {
    name                       = "HTTP"
    priority                   = 100
    direction                  = "Outbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "*"
    source_address_prefix      = "*"
    destination_address_prefix = "*"
  }
}

resource "azurerm_subnet" "example" {
  name                 = "internal"
  resource_group_name  = azurerm_resource_group.example.name
  virtual_network_name = azurerm_virtual_network.example.name
  address_prefixes     = ["10.0.2.0/24"]
}

resource "azurerm_network_interface" "example" {
  name                = "example-nic"
  location            = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name


  ip_configuration {
    name                          = "internal"
    subnet_id                     = azurerm_subnet.example.id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id          = azurerm_public_ip.publicip.id
  }
}
```

Figura 17 Primer sección del script de Terraform para Azure. Fuente: Elaboración propia

Muy importante crear una red virtual en donde se le especifica el espacio de direcciones que se usará cuando se cree la instancia, un nombre para esta red y luego parámetros más simples como la locación y el grupo de recursos, el tercer recurso es una ip publica ya que Azure no genera una ip publica de forma automática para la instancia, esta no es una configuración muy complicada ya que solo hay que especificar el nombre y el método de asignación que en este caso fue estático, los demás parámetros no son nada nuevo ya que se usaron anteriormente.

Un grupo de seguridad es muy importante para permitir peticiones que procedan de diferentes ordenadores, y es por eso por lo que con el atributo *security_role* se permite todo tipo de peticiones con el protocolo TCP. Una subred también es requerida aunque

no hay mayor complicación en su configuración ya que se usan parámetros de los recursos anteriormente creados; muy importante una interfaz de red ya que en esta se especifica la IP publica e IP privada que se van a usar para asignarle a la instancia que se va a crear luego de ejecutar el script, luego de crear estos recursos indispensables ya se puede continuar con la creación del último recurso el cual es la instancia en sí, dicho recurso de puede apreciar en la Figura 18 Segunda sección del script de Terraform para Azure

```
#Datos del resoruce_group donde esta alojada la imagen de packer
data "azurerm_resource_group" "image" {
  name = "myRsourceGroupPacker"
}

data "azurerm_image" "image" {
  name                = "PackerUbuntuDocker"
  resource_group_name = data.azurerm_resource_group.image.name
}

resource "azurerm_virtual_machine" "example" {
  name                = "example-machine"
  resource_group_name = azurerm_resource_group.example.name
  location            = azurerm_resource_group.example.location
  vm_size                = "Standard_DS1_v2"
  network_interface_ids = [
    azurerm_network_interface.example.id,
  ]

  delete_os_disk_on_termination = true

  delete_data_disks_on_termination = true

  storage_image_reference {
      id = data.azurerm_image.image.id
  }

  storage_os_disk {
    name              = "myosdisk1"
    caching           = "ReadWrite"
    create_option     = "FromImage"
    managed_disk_type = "Standard_LRS"
  }

  storage_data_disk {
    name = "mydisk"
    lun          = 0
    caching        = "ReadWrite"
    create_option  = "Empty"
    disk_size_gb   = 20
  }

  os_profile {
    computer_name  = "hostname"
    admin_username = "testadmin"
    admin_password = "Password1234!"
    custom_data = "docker run -d -p 80:80 --name test nginx"
  }

  os_profile_linux_config {
    disable_password_authentication = false
  }

  tags = {
    environment = "staging"
  }
}

data "azurerm_public_ip" "ip" {
  name                = azurerm_public_ip.publicip.name
  resource_group_name = azurerm_virtual_machine.example.resource_group_name
  depends_on          = [azurerm_virtual_machine.example]
}
```

Figura 18 Segunda sección del script de Terraform para Azure. Fuente: Elaboración propia

Para poder utilizar la imagen creada con Packer es importante señalar en unas variables el nombre del recurso donde se encuentra dicha imagen y el nombre de la misma, dentro del recurso de la instancia se debe especificar el tipo de instancia que se creará en este caso fue una Estándar_DS1_V2 la cual es lo suficiente para poder correr un servidor http

como NGINX, se especifican la interfaz de red que se utilizará para la instancia lo cual es fácil ya que anteriormente ya se ha creado ese recurso, es muy importante las líneas de código donde se especifica que sucederá con el disco y con la información luego de que se elimine la instancia ya que muchas veces se necesita que esa información perdure, aunque en este caso están configurados para que se elimine el disco y la información que hay en él ya que no es de relevancia mantener un disco luego de que esta instancia sea borrada, para poder utilizar la imagen creada con Packer se debe especificar el id de la misma en el atributo *storage_image_reference* lo cual es fácil ya que anteriormente ya se crearon variables con esa información para evitar que el id este escrito explícitamente en el script, de este modo al eliminar esta imagen y crear otra ya no es necesario cambiar el código de Terraform. Luego de ello se configuran parámetros del disco como por ejemplo el tamaño de este y en el apartado *os_profile* se configuran parámetros para el sistema operativo de la instancia como el usuario, contraseña y un comando que se lanzara apenas se inicie este sistema operativo, en este caso es el comando que levanta un contenedor de Docker con un servidor http y por último se crea una variable para poder mostrar la IP de la instancia recién creada, dicha variable es usada en el archivo outputs.tf que es el que se encarga de mostrar varios datos de salida que se le especifiquen.

Al instalar todo lo necesario con `terraform init` y planificar la ejecución con `terraform plan` no queda más que aplicar todo lo creado en Azure con un `terraform apply` obteniendo como salida por consola lo que se observa en la Figura 19 Salida por consola de Terraform en Azure

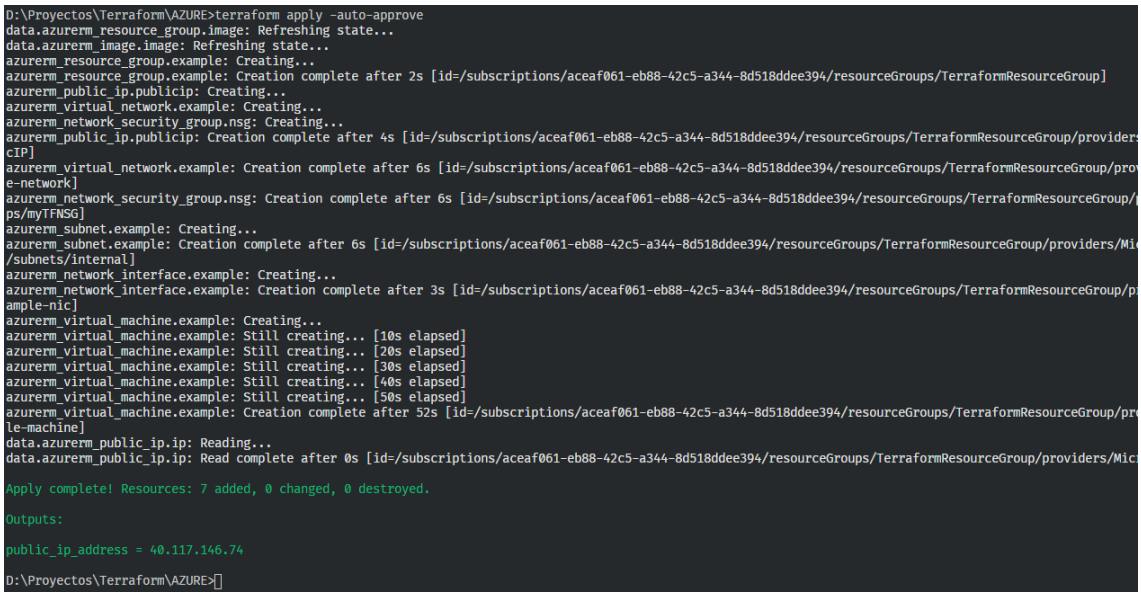


Figura 19 Salida por consola de Terraform en Azure. Fuente: Elaboración propia

Al ser tantos recursos la ejecución tomó mucho tiempo en completarse en comparación con las ejecuciones en otros proveedores, aunque si únicamente se tome en cuenta la creación de la instancia sigue siendo un tiempo alto de 52 segundos, por lo cual es obvio que quizá la integración con los servicios de Azure no esta tan pulido como con los 2 proveedores anteriormente mencionados en esta sección de resultados de igual forma se obtiene la ip publica y al acceder a ella se puede observar la bienvenida que da el servidor http por defecto. Este comportamiento se puede tomar como una desventaja o como una ventaja dependiendo de la perspectiva de quien este usando Terraform en Azure, ya que puede representar una mayor personalización en sus recursos al levantar una instancia con una interfaz de red y una red virtual a su gusto, o también puede representar una perdida tiempo para quien desee únicamente levantar una instancia sin necesidad de configurar recursos que probablemente no usará al máximo.

De cualquier forma, al eliminar esta instancia en la Figura 20 Salida por consola de terraform destroy en Azure, toma casi el mismo tiempo de creación de la instancia en eliminar todos los recursos unos 47 segundos ya que debe eliminar en total 7 de estos.

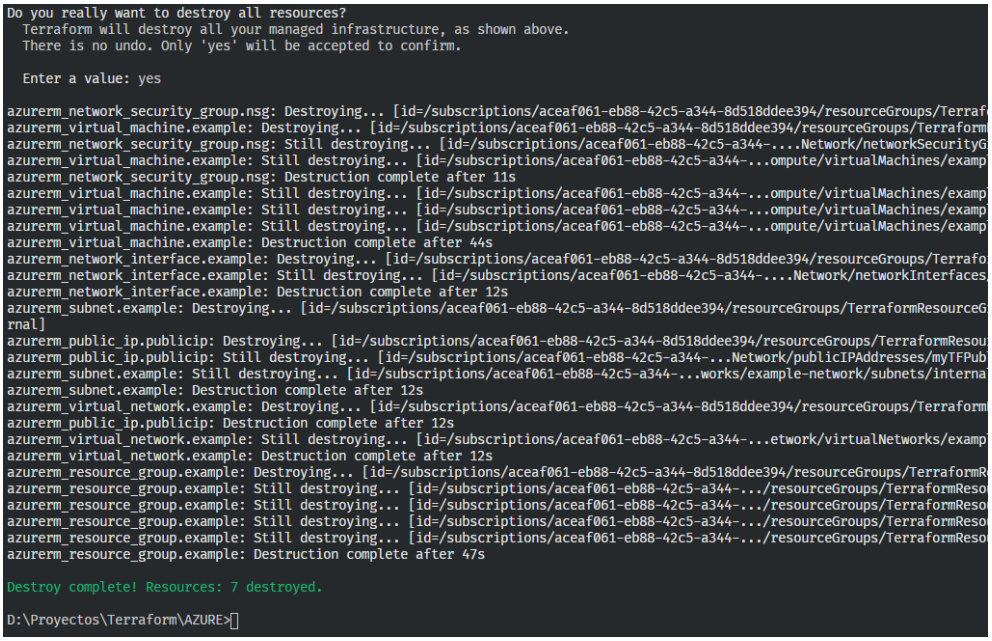


Figura 20 Salida por consola de terraform destroy en Azure. Fuente: Elaboración propia

Comportamiento

El comportamiento de estas instancias no varía mucho entre los proveedores en términos de estabilidad, ya que ninguna de estas generó un problema por ejemplo en el caso de GCP ya había una instancia creada antes de que se creara una con terraform como se visualiza en la Figura 21 Consola de GCP Instancia creada con Terraform y no existió

ningún tipo de problema generado con la nueva instancia, ya que al eliminar la instancia con Terraform la otra ni siquiera sufrió cambios.

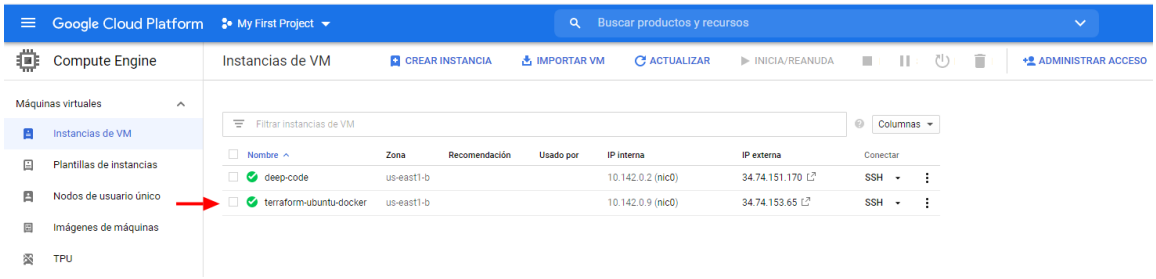


Figura 21 Consola de GCP Instancia creada con Terraform. Fuente: Elaboración propia

Si bien es cierto que en términos de estabilidad ninguna se diferencia de otra en velocidad de operabilidad tanto GCP como AWS son muy superiores que Azure ya que los servidores http estuvieron disponibles casi al instante en que terminó la ejecución de terraform, sin embargo, en Azure tomo algunos minutos el poder visualizar la pantalla de bienvenida de NGINX y esto más que pueda deberse a Terraform es más una cuestión de optimización del proveedor, lo cual probablemente no lo convierta en la primera opción para crear varias instancias y obtener resultados de inmediato, al menos con instancias creadas con Terraform.

En cada uno de los proveedores se pudo utilizar las imágenes creadas con Packer, en el cual resalta más Google ya que no es necesario especificarle ningún tipo de Id o proyecto en el que se encuentre la imagen, únicamente con escribir el nombre de la imagen Terraform es capaz de encontrar la imagen y montarla en una instancia.

Cabe recalcar que para cada una de las instancias creadas en los diferentes proveedores se declaró un tipo de máquina básico así que las 3 instancias estaban en igualdad de condiciones por lo que el margen de diferencia entre cada una de estas es mínimo y ninguna de estas se tardó más que otra por la cantidad de hardware que utilizaba, sino más bien por una cuestión de optimización tanto por HashiCorp y el proveedor de servicios de la nube.

Capítulo 4

Discusión

En la investigación de Sandobalin *et al.* [6] se habla de que Argon es más eficaz en lo que respecta a apoyar el enfoque IAC en comparación con Ansible y es aquí en la cual surgen diferencias con Terraform que es la herramienta que se evalúa en esta investigación, ya que esta herramienta posee una gran facilidad para modelar la infraestructura en la nube tomando en cuenta que hay configuraciones que cambian con respecto al proveedor, por lo tanto es bastante adaptable, se sugiere que los autores de la investigación en la que Argon demuestra su eficacia al modelar infraestructura en la nube no tomaron en cuenta una herramienta idónea para realizar su comparación, estos debían buscar una más potente y dirigida a crear recursos como máquinas virtuales, redes, subredes y entre otros. No obstante, escogieron a Ansible que se usa mayormente para aprovisionar recursos ya creados con otras herramientas o creados de forma manual y es diferente al objetivo de esta investigación, ya que se busca evaluar a Terraform la cual sirve justamente para crear estos recursos antes mencionados y aprovisionarlos con la misma herramienta o con herramientas de terceros enfocadas al aprovisionamiento.

Dalla *et al.* [7] propone 46 métricas para identificar las propiedades de la IAC de las cuales hay 24 que son específicas para Ansible que aunque sean observables también en otros lenguajes de configuración y orquestación no son de mucha ayuda para Terraform, 14 se adaptaron para Ansible y 8 son independientes del lenguaje las cuales en apreciación de este trabajo son las únicas que pueden ser de ayuda para las personas que usen Terrafom y cualquier otra herramienta ya que son generales y esto se ve reflejado en [7] ya que los autores indican que algunas de estas métricas pueden ser usadas en otras herramientas como Chef y Pupet.

Automatizar procesos es la tendencia actualmente y en [8] queda claro que ya existen muchas herramientas que ayudan con esta tarea y que al usarlas juntas en entornos CI/CD tienen un gran potencial, Terraform no se queda atrás ya que esta herramienta puede ser incluida en estos entornos antes mencionados ya sea al ser manejados por herramientas como Jenkins o GitLab CI/CD para ejecutar scripts de IAC de forma totalmente automatizada y sin que un usuario tenga que ejecutar los comandos de Terraform, de esta forma existirían proyectos en donde este el código fuente de una app y el script con la

infraestructura necesaria para desplegar esa app y solo con realizar un *push* al repositorio todo este proyecto se levante directamente.

Los conceptos como PaaS cada vez son más usados ya que van ligados a ofrecer un servicio y es en [9], en donde se puede apreciar que Terraform es utilizado perfectamente para el desarrollo de una plataforma como servicio, es aquí en donde se puede apreciar la gran ventaja que genera esta herramienta ya que como conclusión se indica que obtuvieron un mayor control de su arquitectura usando Terraform, ya que todo estaba declarado como código obteniendo así ventajas como el versionado de código, con esto se puede apreciar que al igual que en esta investigación se realizaron pruebas creando instancias y levantando un servidor Http por lo cual las investigaciones comparten ideas y resultados.

Es importante recordar que al usar scripts es muy probable que se comentan errores o fallos de seguridad y es por ello por lo que A. Rahman [10], profundiza en las características defectuosas de estos scripts, se concuerda en que el autor está en toda su razón, ya que al declarar todo en forma de código es muy fácil pasar por alto configuraciones de seguridad o privacidad que podrían comprometer la integridad de un proyecto y las personas involucradas en él, por eso se está de acuerdo con adoptar las propuestas hechas en [10], para evitar los defectos en los scripts de IAC.

Capítulo 5

Conclusiones

En esta investigación se pudo apreciar cual es el alcance y limitación que posee Terraform, lo cual es muy importante para quienes desean implementar esta herramienta en su flujo de trabajo, ya que es bueno conocer hasta dónde puede llegar aquello que se desea empezar a usar; básicamente tanto la limitación y el alcance van ligados al proveedor en donde se esté empleando la herramienta ya que si existe diferencia entre uno y otro al crear los scripts y en los tiempos de ejecución. En este último se pudo notar que, si existía una diferencia en los tiempos, aunque al ponerlo en práctica con proyectos grandes o pequeños estos segundos de diferencia no representan un problema.

Actualmente se demostró que Terraform es la mejor herramienta para poder crear recursos en los proveedores de nube seleccionados para esta investigación, ya que no hubo mayor complicación en realizar la parte práctica, debido a que en la mayoría de los casos la documentación fue lo suficientemente buena para poder avanzar aunque cabe recalcar que por parte de los proveedores si hubo una excepción en la documentación y este es el caso de Azure en donde la documentación presentada no tenía nada que ver con la documentación que presentaba HashiCorp, lo cual dificultó un poco el aprendizaje de Terraform en ese proveedor y mencionar que GCP posee una documentación muy clara y concisa de todo lo que se desea saber acerca de usar Terraform para crear instancias en su nube.

Las nuevas tendencias apuntan a automatizar todo lo que se pueda y no cabe duda de que con Terraform se puede automatizar esos procesos manuales de configuración en la consola del proveedor, aunque es importante integrarlo con otras herramientas como Packer que crea imágenes de máquina ya aprovisionadas y listas para ser usadas por Terraform de esta forma queda un par de herramientas de la misma empresa que cubren esa necesidad de aprovisionar y crear infraestructura. IAC como concepto supone una revolución en la computación en la nube y es algo que la mayoría de los programadores y DevOps deberían aprender y comprender para no quedarse atrás, siempre y cuando se respeten las buenas prácticas y se mantenga un nivel de seguridad y privacidad bueno evitando defectos en estos scripts.

Recomendaciones

Se recomienda ampliamente el uso de Terraform para crear recursos en la nube y sobre todo que se use en conjunto con Packer ya que entre estas herramientas se complementan para agilizar la creación de toda un infraestructura que es lo que se busca, el proveedor en el cual se obtuvieron los mejores resultados fue GCP; por lo tanto, si se desea replicar el experimento de esta investigación se recomienda emplear el proveedor antes mencionado ya que es el que más facilidades le brinda al usuario.

Para continuar con el estudio de la IAC también se recomienda estudiar para luego utilizar herramientas como Vault por HashiCorp la cual sirve para administrar y proteger datos confidenciales como lo pueden ser las llaves de acceso o credenciales necesarias para poder usar Terraform y Packer con cualquier proveedor, de esta forma se conseguirá una mayor seguridad evitando alguien externo use de manera inadecuada estas llaves de acceso o credenciales. En general el ecosistema de herramientas que provee HashiCorp es muy bueno y funcionan muy bien entre sí por lo cual no hay excusas para no manejar de manera correcta estos datos confidenciales.

Por último, también se recomienda el no quedarse únicamente con Terraform y seguirse adaptando a las nuevas herramientas que vayan apareciendo con el tiempo ya que tal y como avanza la industria tecnológica es cuestión de tiempo para que aparezca una herramienta que llegue a sustituir Terraform y es importante aprenderla ya adaptarse a ella para seguir creciendo como programador o como DevOps cualquiera que sea el caso.

REFERENCIAS

[1] O. Lavriv, M. Klymash, G. Grynkevych, O. Tkachenko, and V. Vasylenko, "Method of cloud system disaster recovery based on 'Infrastructure as a code' concept," *14th Int. Conf. Adv. Trends Radioelectron. Telecommun. Comput. Eng. TCSET 2018 - Proc.*, vol. 2018-April, pp. 1139–1142, 2018, doi: 10.1109/TCSET.2018.8336395.

[2] C. Computing, "Single and Multi-Cloud Disaster Recovery Management using Terraform and Ansible Ram Barath Thiyagarajan Supervisor :," p. 19, 2019.

[3] HashiCorp, "Terraform - Terraform by HashiCorp." https://www.terraform.io/intro/index.html (accessed Jun. 28, 2020).

[4] A. Rahman, J. Stallings, and L. Williams, "Poster: Defect prediction metrics for infrastructure as code scripts in DevOps," *Proc. - Int. Conf. Softw. Eng.*, pp. 414–415, 2018, doi: 10.1145/3183440.3195034.

[5] B. Campbell, *The Definitive Guide to AWS Infrastructure Automation*. 2020.

[6] J. Sandobalin, E. Insfran, and S. Abrahao, "On the effectiveness of tools to support infrastructure as code: Model-driven versus code-centric," *IEEE Access*, vol. 8, pp. 17734–17761, 2020, doi: 10.1109/ACCESS.2020.2966597.

[7] S. Dalla, D. Di, F. Palomba, and D. A. Tamburri, "Towards a Catalogue of Software Quality Metrics for Infrastructure Code," pp. 1–7, 2020.

[8] A. Agarwal, S. Gupta, and T. Choudhury, "Continuous and Integrated Software Development using DevOps," *2018 Int. Conf. Adv. Comput. Commun. Eng.*, no. June, pp. 290–293, 2018, doi: 10.1109/icacce.2018.8458052.

[9] D. Ivanova, P. Borovska, and S. Zahov, "Development of PaaS using AWS and Terraform for medical imaging analytics," *AIP Conf. Proc.*, vol. 2048, no. December, 2018, doi: 10.1063/1.5082133.

[10] A. Rahman, "Characteristics of defective infrastructure as code scripts in DevOps," *Proc. - Int. Conf. Softw. Eng.*, no. i, pp. 476–479, 2018, doi: 10.1145/3183440.3183452.

[11] Kief Morris, *"Infrastructure as Code: Managing Servers in the Cloud,"* 1st Edition. 2016.

[12] J. Sandobalin, E. Insfran, and S. Abrahao, "ARGON: A model-driven infrastructure provisioning tool," *Proc. - 2019 ACM/IEEE 22nd Int. Conf. Model Driven Eng. Lang. Syst. Companion, Model. 2019*, vol. 8, pp. 738–742, 2019, doi: 10.1109/MODELS-C.2019.00114.

[13] T. Grønli and R. Kazman, “Immutable Infrastructure Calls for Immutable Architecture : Deploying a Changeless Architecture in the Cloud,” vol. 6, pp. 7058–7066, 2019.

[14] M. Hüttermann and M. Hüttermann, “Infrastructure as Code,” *DevOps Dev.*, pp. 135–156, 2012, doi: 10.1007/978-1-4302-4570-4_9.

[15] P. M. Díaz, “Diseño de una Infraestructura Cloud en AWS Mediante una Solución de Tipo Infrastructure as Code para una Aplicación Web de Reproducción Multimedia,” 2019.

[16] Amazon Web Services, “AWS CloudFormation - Infraestructura como código y aprovisionamiento de recursos de AWS.” https://aws.amazon.com/es/cloudformation/ (accessed Jun. 29, 2020).

[17] HashiCorp, “Terraform Cloud - Terraform by HashiCorp.” https://www.terraform.io/docs/cloud/index.html (accessed Jun. 28, 2020).

[18] HashiCorp, “Terraform Enterprise - Terraform by HashiCorp.” https://www.terraform.io/docs/enterprise/index.html (accessed Jun. 28, 2020).

[19] J. Schwarz, A. Steffens, and H. Lichter, “Code smells in infrastructure as code,” *Proc. - 2018 Int. Conf. Qual. Inf. Commun. Technol. QUATIC 2018*, pp. 220–228, 2018, doi: 10.1109/QUATIC.2018.00040.

[20] O. Medina and E. Schumann, “DevOps for SharePoint,” *DevOps for SharePoint*, pp. 197–223, 2018, doi: 10.1007/978-1-4842-3688-8.

[21] R. Glassey, “Adopting Git/Github within Teaching: A Survey of Tool Support,” *CompEd 2019 - Proc. ACM Conf. Glob. Comput. Educ.*, pp. 143–149, 2019, doi: 10.1145/3300115.3309518.

[22] “Documentación de GitHub Actions - GitHub Docs.” https://docs.github.com/es/free-pro-team@latest/actions (accessed Dec. 10, 2020).

[23] “GitLab CI | GitLab.” https://about.gitlab.com/stages-devops-lifecycle/continuous-integration/ (accessed Dec. 10, 2020).

[24] ISO25000, “ISO 25010.” https://iso25000.com/index.php/normas-iso-25000/iso-25010?limit=3&limitstart=0 (accessed Jul. 01, 2020).

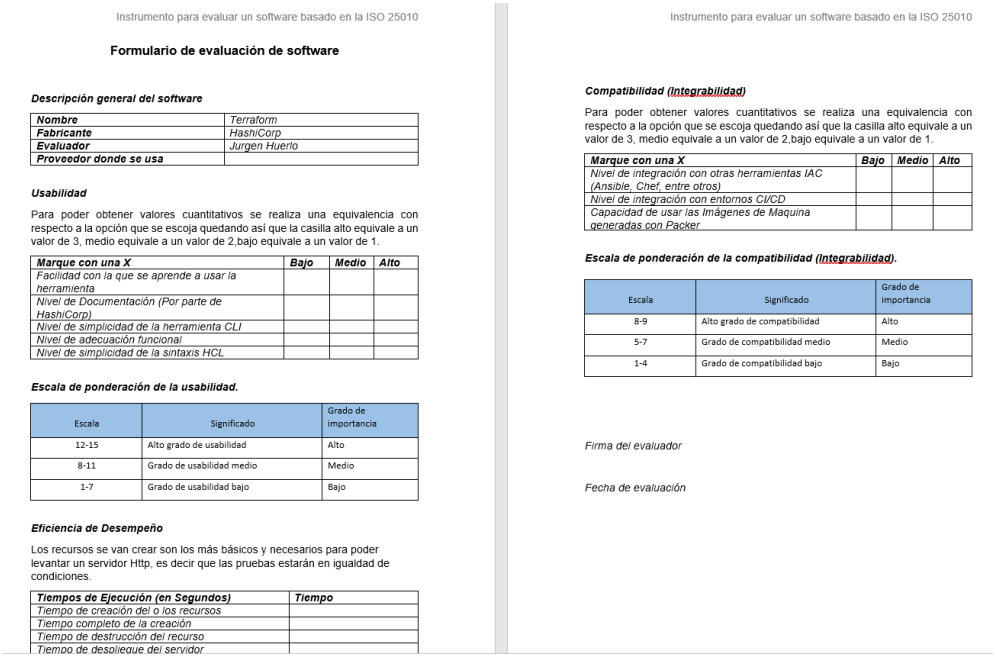
[25] República de Ecuador, “Constitución del Ecuador,” *Regist. Of.*, vol. 449, p. 151, 2015.

[26] Asamblea Nacional del Ecuador, “Coesc,” *Regist. Of.*, vol. IV, pp. 30–31, 2016, [Online]. Available: http://www.wipo.int/edocs/lexdocs/laws/es/ec/ec075es.pdf.

[27] S. Plan Nacional De Desarrollo/, “Plan Nacional para el Buen Vivir 2013-2017.pdf,” p. 600, 2013, [Online]. Available: www.buenvivir.gob.ec.

[28] J. A. Delgado, “Gobernanza de Internet en Ecuador: Infraestructura y acceso / Internet Governance in Ecuador: Infrastructure and Access,” *Encuentro Nac. Gob. Internet*, p. 31, 2014.

ANEXOS



Instrumento para evaluar un software basado en la ISO 25010

Formulario de evaluación de software

Descripción general del software

Nombre	*Terraform*
Fabricante	*HashiCorp*
Evaluador	*Jurgen Huerlo*
Proveedor donde se usa	

Usabilidad

Para poder obtener valores cuantitativos se realiza una equivalencia con respecto a la opción que se escoja quedando así que la casilla alto equivale a un valor de 3, medio equivale a un valor de 2,bajo equivale a un valor de 1.

Marque con una X	***Bajo***	***Medio***	***Alto***
Facilidad con la que se aprende a usar la herramienta			
Nivel de Documentación (Por parte de HashiCorp)			
Nivel de simplicidad de la herramienta CLI			
Nivel de adecuación funcional			
Nivel de simplicidad de la sintaxis HCL			

Escala de ponderación de la usabilidad.

Escala	Significado	Grado de importancia
12-15	Alto grado de usabilidad	Alto
8-11	Grado de usabilidad medio	Medio
1-7	Grado de usabilidad bajo	Bajo

Eficiencia de Desempeño

Los recursos se van crear son los más básicos y necesarios para poder levantar un servidor Http, es decir que las pruebas estarán en igualdad de condiciones.

Tiempos de Ejecución (en Segundos)	***Tiempo***
Tiempo de creación del o los recursos	
Tiempo completo de la creación	
Tiempo de destrucción del recurso	
Tiempo de despliegue del servidor	

Instrumento para evaluar un software basado en la ISO 25010

Compatibilidad (Integrabilidad)

Para poder obtener valores cuantitativos se realiza una equivalencia con respecto a la opción que se escoja quedando así que la casilla alto equivale a un valor de 3, medio equivale a un valor de 2,bajo equivale a un valor de 1.

Marque con una X	***Bajo***	***Medio***	***Alto***
Nivel de integración con otras herramientas IAC (Ansible, Chef, entre otros)			
Nivel de integración con entornos CI/CD			
Capacidad de usar las Imágenes de Maquina generadas con Packer			

Escala de ponderación de la compatibilidad (Integrabilidad).

Escala	Significado	Grado de importancia
8-9	Alto grado de compatibilidad	Alto
5-7	Grado de compatibilidad medio	Medio
1-4	Grado de compatibilidad bajo	Bajo

Firma del evaluador

Fecha de evaluación

Anexo 1 Ficha de evaluación

Pontificia Universidad Católica del Ecuador
Sede Esmeraldas
Escuela de Sistemas y Computación

PUCE

El instrumento contiene instrucciones claras y precisas para responder el cuestionario.	x		
Los ítems permiten el logro del objetivo de la investigación.	x		
Los ítems están distribuidos en forma lógica y secuencial.	x		
El número de ítems es suficiente para recoger la información. En caso de ser negativa su respuesta, sugiera los ítems a añadir.	x		
VALIDEZ			
APLICABLE:	**x**	**NO APLICABLE:**	
APLICABLE ATENDIENDO LAS OBSERVACIONES:			
Validado por: Marc Grob			
C.I.: 1720151503			
Firma:			
Fecha: 22 feb 2021			

Anexo 2 Aprobación Instrumento por Mgt. Marc Grob

Pontificia Universidad Católica del Ecuador
Sede Esmeraldas
Escuela de Sistemas y Computación

											capaz o no?, considere replantear el elemento
13											

ASPECTOS GENERALES	SI	NO	OBSERVACIONES
El instrumento contiene instrucciones claras y precisas para responder el cuestionario.	x		Podría tener mayor explicación en las escalas.
Los ítems permiten el logro del objetivo de la investigación.	x		
Los ítems están distribuidos en forma lógica y secuencial.	x		
El número de ítems es suficiente para recoger la información. En caso de ser negativa su respuesta, sugiera los ítems a añadir.	x		

VALIDEZ			
APLICABLE:	x	NO APLICABLE:	
APLICABLE ATENDIENDO LAS OBSERVACIONES:	x		
Validado por: SUSANA PATIÑO			
C.I.: 0802119017			
Firma:			
Fecha:			

Anexo 3 Aprobación Instrumento por Susana Patiño

Pontificia Universidad Católica del Ecuador
Sede Esmeraldas
Escuela de Sistemas y Computación

12	X		X			X	X		X		

ASPECTOS GENERALES	SI	NO	OBSERVACIONES
El instrumento contiene instrucciones claras y precisas para responder el cuestionario.	X		
Los ítems permiten el logro del objetivo de la investigación.	X		Considere observación en los ítems donde se comparan tiempos de ejecución
Los ítems están distribuidos en forma lógica y secuencial.	X		
El número de ítems es suficiente para recoger la información. En caso de ser negativa su respuesta, sugiera los ítems a añadir.	X		

VALIDEZ			
APLICABLE:		NO APLICABLE:	
APLICABLE ATENDIENDO LAS OBSERVACIONES:	X		
Validado por: Evelin Lorena Flores García			
C.I.: 0801945411			
Firma: Evelin Flores G.			
Fecha: 21/02/2021			

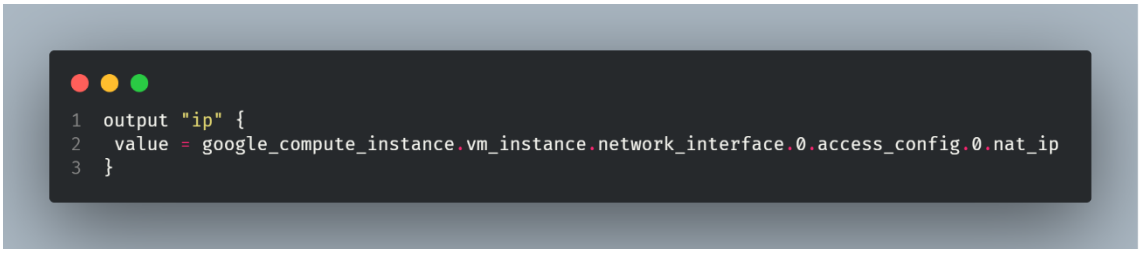
Anexo 4 Aprobación Instrumento por Evelin Flores

```
variable "ami_id" {
    default=""
    description="Base image to create an instance"
    type=string
}

variable "instance_type" {
    default=""
    description="type of instance that we will create"
    type=string
}

variable "tags" {
    type=map
}

variable "sg_name" {

}

variable "ingress_rules" {

}

variable "egress_rules" {

}
```

Anexo 5 Script de declaración de variables.

```
ami_id="ami-055ce82cfde9bd0be"
instance_type="t2.micro"
tags={
    Name="practica1",
    Enviroment="prod"
}
sg_name = "jurgen-rules"
ingress_rules = [
    {
        from_port = "22",
        to_port = "22",
        protocol = "tcp",
        cidr_blocks = ["0.0.0.0/0"]
    },
    {
        from_port = "80",
        to_port = "80",
        protocol = "tcp",
        cidr_blocks = ["0.0.0.0/0"]
    }
]
egress_rules = [
    {
        from_port = 0,
        to_port = 0,
        protocol = -1,
        cidr_blocks = ["0.0.0.0/0"]
    }
]
```

Anexo 6 Script de ingreso de valor para las variables.



Anexo 7 Archivo Outputs.tf para obtener la ip de la instancia recién creada.