Terraform

Jose Torres Lima[‡]
Departamento de Ciencia de la Computación
Universidad Nacional de San Agustín
Email: [‡]jtorresli@unsa.edu.pe

***Resumen*—El siguiente documento recopiló información sobre la herramienta Terraform que es usada para automatización y aprovisionamiento de recursos. La estructura del documento inicia describiendo que es Terraform, en la siguiente sección se describen sus casos de uso, luego se describe el concepto de que es Infraestructura como Código, además de que relación tiene Terraform con este concepto, seguidamente se describe el modo de funcionamiento de Terraform y finalmente se concluye este documento con un ejemplo del uso de Terraform.**

***Keywords*—Terraform, IaC, DevOps, Automatización.**

1. Terraform

Es una herramienta de código abierto, creada por HashiCorp que le permite codificar su infraestructura como archivos de configuración declarativos que se versionan, comparten y se pueden revisar.

Terraform permite construir, cambiar y crear versiones de infraestructura de manera segura y eficiente. Terraform puede administrar proveedores de servicios existentes y populares, así como soluciones internas personalizadas [1].

Los archivos de configuración describen a Terraform los componentes necesarios para ejecutar una sola aplicación o todo su centro de datos. Terraform genera un plan de ejecución que describe lo que hará para alcanzar el estado deseado y luego lo ejecuta para construir la infraestructura descrita. A medida que cambia la configuración, Terraform puede determinar qué cambió y crear planes de ejecución incrementales que se pueden aplicar [1].

La infraestructura que Terraform puede administrar incluye componentes de bajo nivel como instancias de maquinas, almacenamiento y redes, así como componentes de alto nivel como entradas de DNS, características de SaaS, etc [1].

Las características clave de Terraform son [1]:

- **Infraestructura como Código**: La infraestructura se describe mediante una sintaxis de configuración de alto nivel llamada HCL (lenguaje de configuración de HashiCorp) a través de código.
- **Planes de Ejecución**: El plan de ejecución muestra lo que Terraform hará cuando llame a apply. Esto le permite evitar sorpresas cuando Terraform manipula la infraestructura.
- **Gráfico de Recursos**: Terraform crea un gráfico de todos sus recursos y paraleliza la creación y modificación de cualquier recurso no dependiente. Mostrando información sobre las dependencias en su infraestructura.
- **Automatización de cambios**: Se pueden aplicar conjuntos de cambios complejos a su infraestructura con una mínima interacción humana, evitando muchos posibles errores humanos.

Terraform soporta muchos proveedores como son [6]:

- AWS
- Google Cloud
- Azure
- Digital Ocean
- Heroku
- OpenStack
- VMware vSphere/vCloud Director.
- entre otros...

El 85 % de la infraestructura de producción de Adobe se gestiona con Terraform [6].

2. Casos de uso

2.1. *Heroku*

Heroku es una PaaS (Plataforma como Servicio) popular para alojar aplicaciones web. Los desarrolladores crean una aplicación y luego adjuntan complementos, como una base de datos o un proveedor de correo electrónico. Una de las mejores características es la capacidad de escalar elásticamente el número de trabajadores. Sin embargo, la mayoría de las aplicaciones no triviales necesitan rápidamente muchos complementos y servicios externos.

Terraform se puede utilizar para codificar la configuración requerida para una aplicación Heroku, asegurando que todos los complementos necesarios estén disponibles, pero puede ir aún más lejos: configurar DNSimple para establecer un CNAME o configurar Cloudflare como una CDN para la aplicación. Lo mejor de todo es que Terraform puede hacer todo esto en menos de 30 segundos sin usar una interfaz web [2].

2.2. *Aplicaciones Multicapa*

Un patrón muy común es la arquitectura de N capas. La arquitectura de 2 capas más común es un grupo de

servidores web que utilizan una capa de base de datos. Se agregan capas adicionales para servidores API, servidores de almacenamiento en caché, mallas de enrutamiento, etc. Este patrón se usa porque las capas se pueden escalar de forma independiente y proporcionan una separación de preocupaciones.

Terraform es una herramienta ideal para construir y gestionar estas infraestructuras. Cada capa puede describirse como una colección de recursos y las dependencias entre cada capa se manejan automáticamente; Terraform se asegurará de que la capa de la base de datos esté disponible antes de que se inicien los servidores web y de que los balanceadores de carga conozcan los nodos web. Luego, cada capa se puede escalar fácilmente usando Terraform modificando un solo valor de configuración de conteo. Debido a que la creación y el aprovisionamiento de los recursos están codificados y automatizados, escalar elásticamente con carga se vuelve trivial [2].

2.3. *Clústeres de autoservicio*

En un cierto tamaño de organización, resulta muy difícil para un equipo de operaciones centralizado administrar una infraestructura grande y en crecimiento. En cambio, se vuelve más atractivo crear una infraestructura de .ªutoservicio", lo que permite a los equipos de productos administrar su propia infraestructura mediante las herramientas proporcionadas por el equipo de operaciones central.

Con Terraform, el conocimiento de cómo construir y escalar un servicio se puede codificar en una configuración. Las configuraciones de Terraform se pueden compartir dentro de una organización, lo que permite a los equipos de clientes usar la configuración como una caja negra y usar Terraform como una herramienta para administrar sus servicios [2].

2.4. *Demostraciones de software*

El software moderno está cada vez más interconectado y distribuido. Aunque existen herramientas como Vagrant para crear entornos virtualizados para demostraciones, sigue siendo un gran desafío realizar demostraciones de software en una infraestructura real que se asemeje más a los entornos de producción.

Los escritores de software pueden proporcionar una configuración de Terraform para crear, aprovisionar y ejecutar una demostración en proveedores de la nube como AWS. Esto permite que los usuarios finales muestren fácilmente el software en su propia infraestructura e incluso permite ajustar parámetros como el tamaño del clúster para probar herramientas de manera más rigurosa a cualquier escala [2].

2.5. *Ambientes desechables*

Es una práctica común tener tanto un entorno de producción como de preparación o de control de calidad. Estos entornos son clones más pequeños de su contraparte de producción, pero se utilizan para probar nuevas aplicaciones antes de lanzarlas en producción. A medida que el entorno de producción crece y se vuelve más complejo, resulta cada vez más molesto mantener un entorno de ensayo actualizado.

Con Terraform, el entorno de producción puede codificarse y luego compartirse con staging, QA o dev. Estas configuraciones se pueden utilizar para poner en marcha rápidamente nuevos entornos para realizar pruebas y luego eliminarlos fácilmente. Terraform puede ayudar a dominar la dificultad de mantener entornos paralelos y hace que sea práctico crearlos y destruirlos elásticamente [2].

2.6. *Redes definidas por software*

Las redes definidas por software (SDN) se están volviendo cada vez más frecuentes en el centro de datos, ya que proporciona más control a los operadores y desarrolladores y permite que la red soporte mejor las aplicaciones que se ejecutan en la parte superior. La mayoría de las implementaciones de SDN tienen una capa de control y una capa de infraestructura.

Terraform se puede utilizar para codificar la configuración de redes definidas por software. Terraform puede utilizar esta configuración para configurar y modificar automáticamente la configuración mediante la interfaz con la capa de control. Esto permite versionar la configuración y automatizar los cambios. A modo de ejemplo, AWS VPC es una de las implementaciones de SDN más utilizadas y puede ser configurada por Terraform [2].

2.7. *Planificadores de recursos*

En infraestructuras a gran escala, la asignación estática de aplicaciones a máquinas se vuelve cada vez más desafiante. Para resolver ese problema, existen varios planificadores como Borg, Mesos, YARN y Kubernetes. Estos se pueden utilizar para programar de forma dinámica contenedores Docker, Hadoop, Spark y muchas otras herramientas de software.

Terraform no se limita a proveedores físicos como AWS. Los planificadores de recursos pueden tratarse como un proveedor, lo que permite que Terraform les solicite recursos. Esto permite que Terraform se utilice en capas: para configurar la infraestructura física que ejecutan los planificadores y para el aprovisionamiento en la red programada [2].

2.8. *Implementación de múltiples nubes*

A menudo resulta atractivo distribuir la infraestructura en varias nubes para aumentar la tolerancia a fallos. Al utilizar solo una región o un proveedor en la nube, la tolerancia a fallas está limitada por la disponibilidad de ese proveedor. Tener una implementación de múltiples nubes permite una recuperación más elegante de la pérdida de una región o de todo el proveedor.

Realizar implementaciones de múltiples nubes puede ser un gran desafío, ya que muchas de las herramientas existentes

para la administración de infraestructura son específicas de la nube. Terraform es independiente de la nube y permite usar una única configuración para administrar múltiples proveedores e incluso para manejar dependencias entre nubes. Esto simplifica la gestión y la orquestación, lo que ayuda a los operadores a construir infraestructuras multinube a gran escala [2].

3. Infraestructura como código

La infraestructura como código (IaC) es la práctica para configurar automáticamente las dependencias del sistema y para aprovisionar instancias locales y remotas [3]. Los profesionales consideran IaC como un pilar fundamental para implementar prácticas de DevOps, lo que les ayuda a entregar rápidamente software y servicios a los usuarios finales. Las organizaciones de tecnología de la información (TI), como GitHub, Mozilla, Facebook, Google y Netflix han adoptado IaC [4].

El uso de scripts de IaC es esencial para implementar la práctica del despliegue continuo. Las tecnologías IaC populares, como Chef y Puppet, proporcionan mecanismos para configurar y aprovisionar automáticamente la infraestructura de implementación de software mediante instancias en la nube. Las organizaciones de tecnología de la información como Ambit Energy, GitHub, Mozilla y Netfix utilizan estos mecanismos para aprovisionar instancias basadas en la nube, como Amazon Web Services (AWS), administrando bases de datos y administrando cuentas de usuario tanto en instancias informáticas locales como remotas. El uso de scripts de IaC ha ayudado a las organizaciones de TI a aumentar su frecuencia de implementación [4].

DevOps promueve el uso de una noción típica de desarrollo de software, es decir, código fuente, también para los diseños de infraestructura, de manera que todo el conjunto de scripts, código de automatización y configuración, modelos, dependencias requeridas y operacionales. Los parámetros de configuración se pueden expresar utilizando el mismo lenguaje estándar, de forma muy similar a como utilizaría Java y las bibliotecas auxiliares de Java para armar su propia aplicación de software; esta práctica es la infraestructura como código. El propósito detrás de esta propuesta es reutilizar las prácticas de desarrollo de software comunes y exitosas para acelerar las operaciones de software: piense en usar el control de versiones del código de infraestructura con el propósito de eliminar errores y retroceder o usar patrones de diseño de infraestructura para juntar rápidamente soluciones comunes a problemas conocidos o incluso explotar la ingeniería basada en modelos para extraer especificaciones de IasC inmediatamente del diseño arquitectónico y modelos de desarrollo [5].

La Fig. 1 muestra cómo se comparan las herramientas IAC más populares.

	Source	Cloud	Type	Infrastructure	Language	Agent	Master	Community	Maturity
Chef	Open	All	Config Mgmt	Mutable	Procedural	Yes	Yes	Large	High
Puppet	Open	All	Config Mgmt	Mutable	Declarative	Yes	Yes	Large	High
Ansible	Open	All	Config Mgmt	Mutable	Procedural	No	No	Huge	Medium
SaltStack	Open	All	Config Mgmt	Mutable	Declarative	Yes	Yes	Large	Medium
CloudFormation	Closed	AWS	Provisioning	Immutable	Declarative	No	No	Small	Medium
Heat	Open	All	Provisioning	Immutable	Declarative	No	No	Small	Low
Terraform	Open	All	Provisioning	Immutable	Declarative	No	No	Huge	Low

Figura 1: Una comparación de las herramientas IAC más populares [7]

4. Relación con el concepto: Infraestructure as Code

En Terraform la infraestructura se describe mediante una sintaxis de configuración de alto nivel llamada HCL (lenguaje de configuración de HashiCorp) a traves de codigo. Esto permite un *blueprint* de su centro de datos para que sea versionado y tratado como lo haría con cualquier otro código. Además, la infraestructura se puede compartir y reutilizar [1].

Con Terraform se puede aprovisionar usando HCL los siguientes recursos [6]:

- S3- Simple Storage Service
- VPC- Virtual Private Cloud
- SQS- Simple Queue Service
- Route53- Hosted DNS
- RDS- Relational Database Service
- IAM- Identity and Access Management
- ECS- EC2 Container Service
- entre otros...

5. Modo de Funcionamiento

Escribir la configuración de nuestra infraestructura en HCL. Una vez escrita, ejecute el siguiente comando:

```
terraform init
```

Listing 1: Inicializando

El comando anterior inicializará Terraform y creará dos carpetas, así como un archivo de estado. El archivo de estado se utiliza para realizar un seguimiento de los recursos que ya se han creado. A continuación, debe ejecutar:

```
terraform plan
```

Listing 2: Planeación de los recursos

Terraform realizará un ensayo y le solicitará un resumen detallado de los recursos que está a punto de crear. Si está seguro de que todo se ve bien, puede crear los recursos con:

```
terraform apply
```

Listing 3: Aplicar la configuración

Se le pedirá que confirme sus opciones; simplemente escriba sí. El proceso tarda unos minutos en aprovisionar los recursos. Luego de la construcción Terraform crea

terraform.tfstate y terraform.tfstate.backup que son los dos archivos utilizados por Terraform para realizar un seguimiento de los recursos que se crearon.

El siguiente comando terraform graph se utiliza para generar una representación visual de una configuración o un plan de ejecución. La salida está en formato DOT, que GraphViz puede utilizar para generar gráficos que muestran las dependencias de los recursos.

```
terraform graph | dot -Tpng > graph.png
```

Listing 4: Representación gráfica

6. Ejemplos

6.1. *Google Cloud - Creando una instancia*

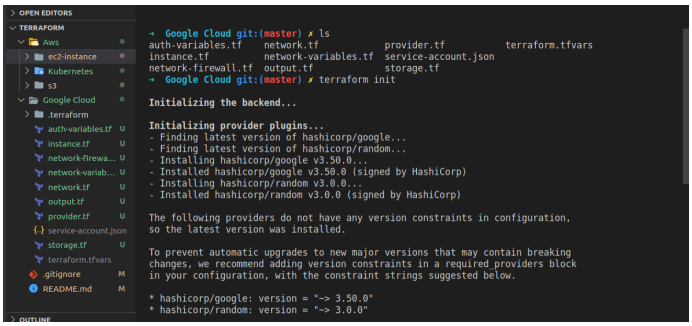

Figura 2: Terraform init

Figura 3: Validando que los archivos estén correctamente escritos

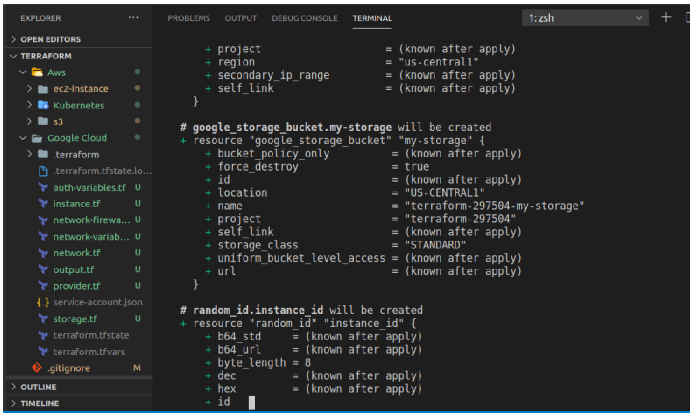

Figura 4: Terraform plan

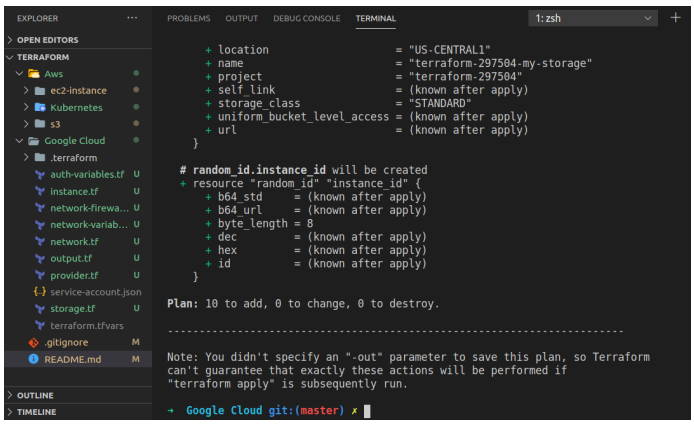

Figura 5: Terraform plan

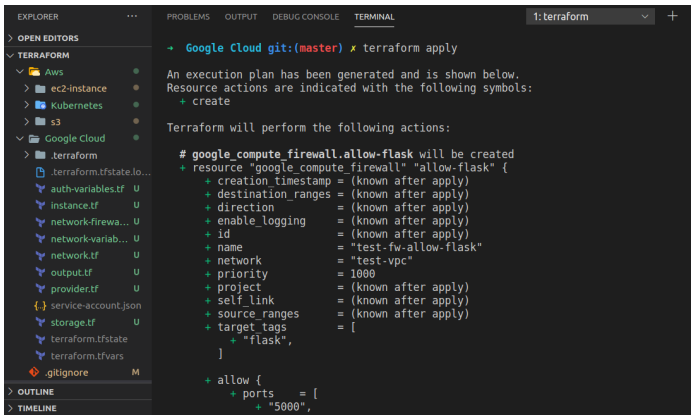

Figura 6: Terraform apply

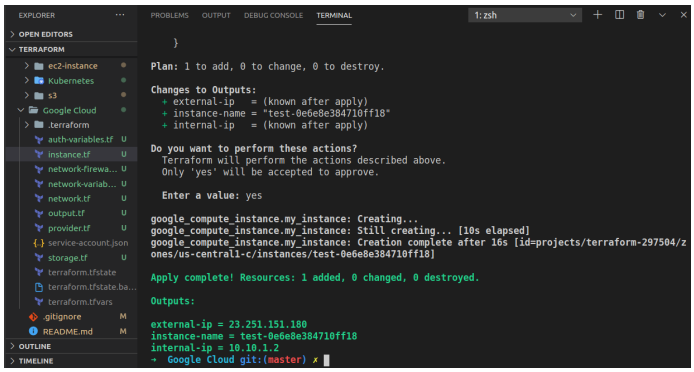

Figura 7: Terraform apply

Figura 8: Google cloud Storage creado

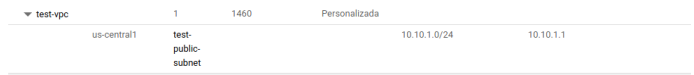

Figura 9: Vpc creado

Figura 10: Reglas de Firewall creados

Figura 11: Instancia creada

Figura 12: Subred creada

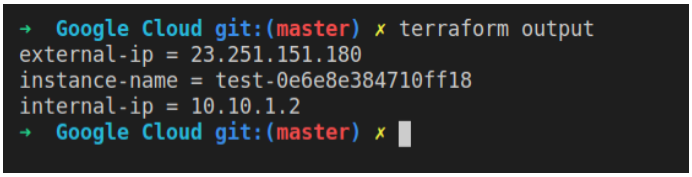

Figura 13: Imprimiendo datos de salida

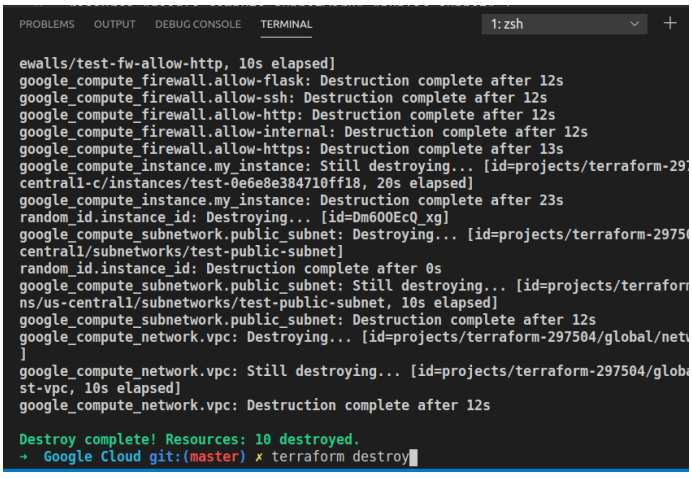

Figura 14: Eliminando todos los recursos creados

6.2. *Aws - Creando tres instancias*

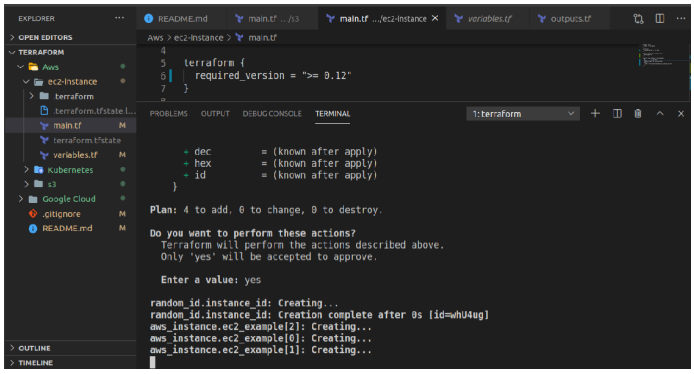

Figura 15: Creando 3 instancias en Aws

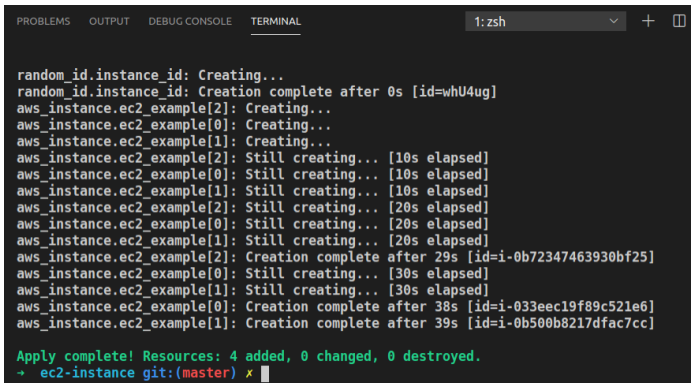

Figura 16: Creando 3 instancias en Aws

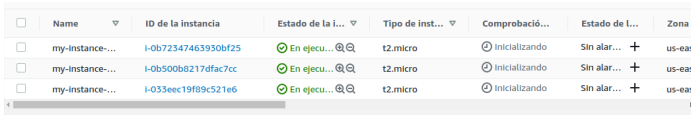

Figura 17: Las 3 instancias en Aws

6.3. *Aws - Creando un S3*

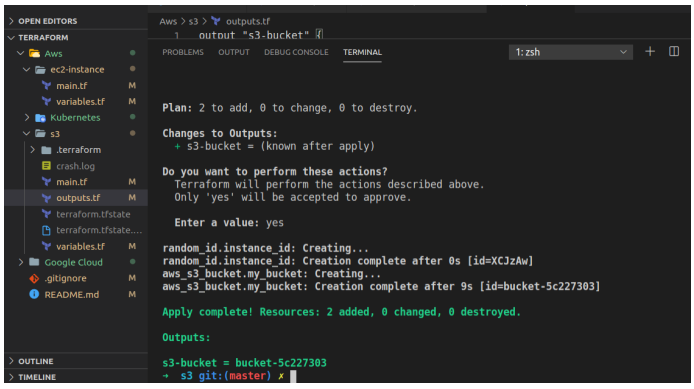

Figura 18: Creando S3

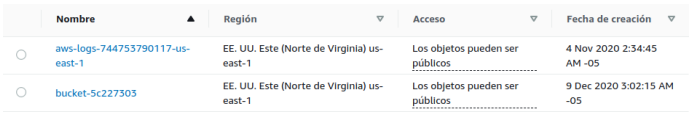

Figura 19: Bucket S3 creado

```
→  s3 git:(master) ✗ terraform output s3-bucket
bucket-5c227303
→  s3 git:(master) ✗
```

Figura 20: Mostrando el nombre del Bucket creado

```
→  s3 git:(master) ✗ terraform graph | dot -Tpng > graph.png
→  s3 git:(master) ✗
```

Figura 21: Graficando las dependencias

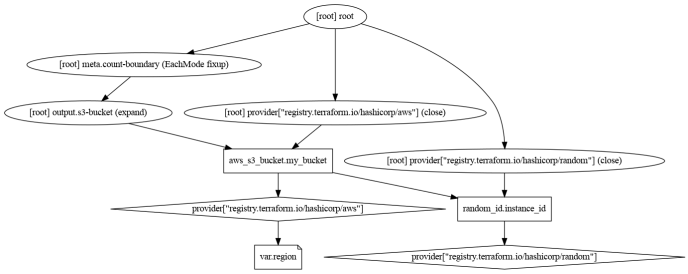

Figura 22: Grafo creado

6.4. Aws - Creando un Cluster de Kubernetes (EKS)

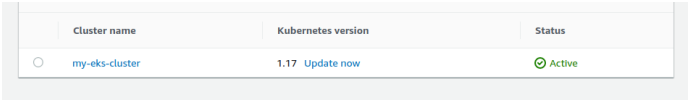

Figura 23: Cluster Elastic Kubernetes Service

El código esta disponible en: https://github.com/Jomat18/Terraform-Kubernetes

REFERENCIAS

[1] HashiCorp. Introduction to Terraform. [Online]. Available: https://www.terraform.io/intro/index.html. [Accessed: 07-dic-2020]

[2] HashiCorp. Use Cases. [Online]. Available: https://www.terraform.io/intro/use-cases.html. [Accessed: 07-dic-2020]

[3] Morris, K. (2016). Infrastructure as code: managing servers in the cloud. .ºReilly Media, Inc.".

[4] Rahman, A., Mahdavi-Hezaveh, R., & Williams, L. (2019). A systematic mapping study of infrastructure as code research. Information and Software Technology, 108, 65-77.

[5] Artac, M., Borovssak, T., Di Nitto, E., Guerriero, M., & Tamburri, D. A. (2017, May). DevOps: introducing infrastructure-as-code. In 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C) (pp. 497-498). IEEE.

[6] Kelvin Jasperson. Terraform at Adobe. [Online]. Available: https://www.usenix.org/sites/default/files/conference/protected-files/srecon16_slides_jasperson.pdf. [Accessed: 07-dic-2020]

[7] Yevgeniy Brikman. Why we use Terraform and not Chef, Puppet, Ansible, SaltStack, or CloudFormation. [Online]. Available: https://blog.gruntwork.io/why-we-use-terraform-and-not-chef-puppet-ansible-saltstack-or-cloudformation-7989dad2865c. [Accessed: 07-dic-2020]