Universidad de Zaragoza

Trabajo de fin de grado

Automatización del despliegue de simulaciones distribuidas en cloud híbrido

Deployment automation of distributed simulations in a hybrid cloud

Hayk Kocharyan

Director: Unai Arronategui Arribalzaga
Codirector: José Ángel Bañares Bañares

Grado en Ingeniería Informática
Tecnologías de la Información

Departamento de Informática e Ingeniería de Sistemas
2021

Resumen

Computer Science for Complex System Modeling (COSMOS) es un grupo de investigación dentro del Departamento de Informática e Ingeniería de Sistemas (DIIS) que trabaja en el desarrollo de sistemas distribuidos. Uno de los proyectos que se encuentra bajo desarrollo es la simulación de Sistema de Eventos Discretos (SED) con una gran amplitud. Su foco principal en estos últimos años es la puesta en marcha de un simulador de SED en un entorno distribuido. En este TFG se propone una prueba para el desarrollo y automatización del despliegue de una infraestructura para la ejecución de **simulaciones distribuidas en un entorno de Cloud Híbrido.**

Uno de los problemas que se afronta cuando este equipo se encuentra ante simulaciones con tamaño amplio es la incapacidad de realizar estas mismas en un único nodo, este problema se solventa con el uso de un simulador distribuido. La solución anterior genera de nuevo un obstáculo que consiste en que la operativa para la puesta en marcha de este simulador y la posterior recogida de datos es complicada y poco cómoda, por lo que es aquí donde entra en juego el Trabajo de Fin de Grado (TFG) que se documenta en esta memoria.

El proyecto que se desarrolla proporciona una solución para facilitar la operativa de las simulaciones. Esto se consigue con el uso de Slurm, un planificador de tareas desplegado en Cloud Híbrido. Elegir un planificador de software libre no ha sido una decisión fácil, puesto que, hoy en día, existen múltiples herramientas en el mercado, todas ellas similares y muy completas. Tras seleccionar el planificador más adecuado, se ha podido desplegar el sistema y elaborar una solución. La **solución propone una infraestructura heterogénea** que hace uso de Google Cloud Platform (GCP) como proveedor del cluster en Cloud Público, y, una cluster on-premise como proveedor del Cloud Privado.

El despliegue del sistema en este entorno híbrido se encuentra automatizado para facilitar su futura puesta en marcha en otras infraestructuras. Esta automatización ha sido realizada haciendo uso de Terraform y Ansible, dos programas de modelización de recursos. El primero ha sido usado para los recursos Cloud Público (máquinas virtuales, zonas DNS, firewall, etc), mientras que el segundo ha sido usado para los recursos de administración de sistemas del Cloud Privado.

Para terminar se han realizado dos tipos de evaluaciones. En primer lugar, se han validado los módulos de automatización de creación y modelización de recursos, y de esta manera se ha demostrado que los módulos desarrollados llevan a cabo su trabajo de manera correcta. Y, en segundo lugar, se han llevado a cabo pruebas con distintos casos de uso de despliegues de simulaciones en Cloud Privado, Cloud Público y Cloud Híbrido para comprobar la robustez y las capacidades del sistema desplegado. Una vez validado el sistema se han hecho pruebas en un entorno real de simulación distribuida. Tras comprobar que las simulaciones se ejecutaban de manera correcta con una asignación de recursos adecuada, y se podía obtener la salida de estas simulaciones de manera centralizada y sencilla, se dio por evaluado de manera correcta el sistema.

Como resultado de ese proyecto, no solo se ha elaborado un TFG, sino que también se ha colaborado en la publicación de un articulo de investigación que ha sido aprobado, publicado y seleccionado como mejor artículo de la conferencia en la GECON - International Conference on the Economics of Grids, Clouds, Systems, and Services. Este proyecto servirá como base para futuros usuarios, profesores y grupos de investigación que deseen seguir la linea de investigación del despliegue de una infraestructura para simulaciones distribuidas en un entorno de Cloud Híbrido.

Dedicatoria

A mi familia, que me han apoyado siempre y han confiado en mi en todo momento.

A mi padre y mi madre, por haber tomado decisiones duras en su vida para poder darme la oportunidad de poder estudiar y conseguir lo que quiero.

A mis amigos, que siempre han estado ahí cuando les necesitaba.

A Adela, que ha sido un punto de apoyo importante, sobre todo en los momentos de mayor desmotivación.

A José Ángel y Unai, por aportar sus amplios conocimientos durante estos años de carrera, y en especial en este trabajo ya que sin ellos no habría sido lo mismo.

Glosario

Cloud Privado define aquellos recursos que se poseen, son gestionados, y administrados de manera privada dentro de una organización. I, V, 2, 9, 12, 17, 19, 42

Cloud Público hace referencia al modelo de computación en la nube en que las aplicaciones y recursos se ofrecen vía Internet como un servicio. I, VI, 2, 9, 15, 17, 19, 20, 42, 55

Cloud Híbrido este termino hace referencia al uso simultáneo del Cloud Privado y el Cloud Público de manera mezclada. I, 2, 9, 17, 19

firewall sistema cuya función es prevenir y proteger a nuestra red privada, de intrusiones o ataques de otras redes, bloqueándole el acceso. Permite el tráfico entrante y saliente que hay entre redes u ordenadores de una misma red. . VI, VIII, X, 7, 11, 13, 15, 29, 30, 38, 39, 45, 47

Infraestructure as a Service es un termino que se usa cuando un proveedor alquila infraestructura (e.g. servidores, máquinas virtuales, BBDD) como un servicio. IV, 10

on-premise en español "en local". Se refiere a la instalación del programa de manera local en las instalaciones de la empresa. Esto lleva a crear una infraestructura informática compleja con servidores que requieren mantenimiento. I, VI, X, 2, 10, 11, 12, 13, 15, 17, 18, 19, 23, 28, 29, 42, 52, 53, 54, 55

Playbook conjunto de tareas (configuración de ficheros, permisos, instalación de paquetes) que se desea que se haga en un nodo a través de Ansible. VIII, X, 6, 14, 17, 22, 32, 35, 36, 51, 52

rollback operación que devuelve a algún estado previo. VIII, X, 32, 35, 51

Túnel IPSec IPSec es la abreviatura de Internet Protocol Security y hace referencia a un conjunto de protocolos que aseguran el cifrado de paquetes y la autenticación de conexiones a través del protocolo IP. En resumen, un Túnel IPSec hace referencia a aquel túnel de red cuyos paquetes se encuentran cifrados con la tecnología de cifrado y autentificado de IPsec. VIII, X, 13, 15, 28, 42, 43, 47, 59

Virtual Private Cloud el término Virtual Private Cloud (VPC) hace referencia a **aquellos recursos que un usuario despliega en un proveedor de cloud público**. Este término resulta confuso ya que los recursos se encuentran en un proveedor de **cloud público** pero el término lleva incluida la palabra ***"private"***. A pesar de ser confuso, se empleará esta palabra a lo largo de la memoria ya que es una concepto que ha sido acuñado por todos los proveedor de cloud actuales para referencias a sus recursos y sería difícil llevar a cabo explicaciones si no se hace uso de esta palabra.. VI, 10, 15

Acrónimos

API Aplication Programming Interface. 5

AWS Amazon Web Services. 7, 10

BBDD Bases de Datos. III

COSMOS Computer Science for Complex System Modeling. I, 1

DHCP Dynamic Host Configuration Protocol. 13

DIIS Departamento de Informática e Ingeniería de Sistemas. I, 1

DNS Domain name server. VI, VIII, 4, 10, 11, 12, 13, 15, 19, 24, 27, 28, 29, 40, 43, 44, 45, 59

EINA Escuela de Ingeniería y Arquitectura. 2, 12

GCP Google Cloud Platform. I, VI, VIII, X, 2, 7, 10, 13, 17, 18, 19, 24, 27, 28, 29, 30, 36, 37, 40, 42, 43, 44, 52, 53, 54, 55

GECON GECON - International Conference on the Economics of Grids, Clouds, Systems, and Services. I

HPC High Performance Computing. Glosario: High Performance Computing, 9

IaaS Infraestructure as a Service. Glosario: Infraestructure as a Service, 10

IP Internet Protocol. 12, 13, 27, 28, 34, 42, 45, 48, 49, 54, 55

MAC Media Access Control. 13, 42

SED Sistema de Eventos Discretos. I, 1, 9, 53

SO Sistema Operativo. 13, 27

TFG Trabajo de Fin de Grado. I, 2, 19

VPC Virtual Private Cloud. Glosario: Virtual Private Cloud, X, 15, 28, 29, 36, 38, 48, 49, 50

VPN Virtual Private Network. VI, VIII, 4, 10, 11, 12, 13, 19, 28, 39, 42, 49, 59

Índice general

Índice de figuras

Fragmentos de código

Capítulo 1

Introducción

1.1. Contexto

Este proyecto es una tarea de investigación del Departamento de Informática e Ingeniería de Sistemas (DIIS) y del grupo Computer Science for Complex System Modeling (COSMOS). Su labor se centra en el desarrollo de sistemas distribuidos complejos. Estos requieren modelos formales para trazar soluciones, usar infraestructuras de computación distribuida, y emplear técnicas de gestión y explotación de grandes cantidades de datos.
En concreto, este proyecto se lleva bajo la linea de investigación de simulación de escenarios y modelos para evaluar, diseñar, decidir o reconfigurar sistemas complejos. También se estudia las **arquitecturas de ejecución sobre cloud** para dar soporte a simulaciones.

En este contexto, este trabajo facilitará la gestión de infraestructuras heterogéneas distribuidas y el despliegue de servicios para soportar la simulación de **Sistema de Eventos Discretos (SED)** de gran tamaño. Así mismo, será un buen proyecto para seguir investigando y conseguir el hito del escalado bajo demanda.

1.2. Motivación y Alcance

Los SED son unas de las técnicas de modelado mas populares. Los SED modelan las operaciones de un sistema como secuencias de la aparición de hechos discretos relevantes, a los que se llaman *eventos*. Cada evento tiene lugar en un instante y produce un cambio de estado en el sistema. Las acciones que ocurren en estos sistemas llevan consigo información como coste económico, duración, consumo energético, etc. Los modelos de SED pueden tener tamaños muy grandes, llevando a la inviabilidad de las simulaciones de las mismas en una única unidad de ejecución. Por este motivo y junto a la necesidad de querer mejorar los tiempos de simulación de estos modelos, surge la necesidad de procesar los eventos de manera concurrente, dando a lugar a los sistemas distribuidos de eventos discretos.

El uso de modelos distribuidos de SED se extiende a multitud de ámbitos. La industria del automóvil es de las redes industriales más complejas, haciendo uso de unas las redes más largas y complejas de cadenas de suministro. Esta industria ha hecho uso de las simulaciones de manera exhaustiva durante muchos años[13].

También se observa el uso de SED en la industria de la medicina. Actualmente está teniendo lugar una de las pandemias mas importante del siglo *XXI*. La investigación en el ámbito de la ciencia ha sido clave para avanzar con el estudio de esta pandemia. Los SED distribuidos se llevan usando para la simulación de vacunación en masa, y en este caso no ha sido menos[1].

Si es cierto que las simulaciones distribuidas son una buena herramienta para muchos usuarios, sin embargo, no se está aprovechando la capacidad computacional del *cloud-*

computing para mejorar estas, y es debido a la dificultad de la estimación del coste de las simulaciones y el coste del uso de recurso. La evolución de la computación conduce a que las simulaciones distribuidas tengan dos temas de discusión importantes, la primera son los fundamentos de sistemas distribuidos y la segunda son las decisiones de diseño basadas en la evolución tecnológica.

1.3. Objetivos

Se tendrá en el foco el objetivo de la puesta en marcha un Cloud Híbrido y ejecutar simulaciones distribuidas en él, estas simulaciones tendrán partes que se ejecutarán en el Cloud Privado y otras en el Cloud Público.

Para conseguir este objetivo, el primer paso será la familiarización con la ejecución de simulaciones en un cluster on-premise de 48 nodos formados por *Raspberry Pi 4* sin uso de orquestadores. Para ello, es necesario comprender como funciona el simulador desarrollado por compañeros que trabajan en este mismo proyecto de investigación y comenzar a realizar ejecuciones con estas. Es este caso se estudió el uso del software desarrollado por Fidel Reviriego Navarro[8] y Álvaro Santamaría de La Fuente[3].

El siguiente paso es la selección de una herramienta completa de software libre que sea capaz de administrar un cluster tolerante a fallos, escalable y que permita orquestar tareas en un sistema Linux. Una vez seleccionada la herramienta se llevará a cabo su configuración y puesta en marcha en el Cloud Privado, y tras validar su correcto funcionamiento y manejo, se extenderá al entorno híbrido desplegando esta herramienta en el Cloud Público. Para la interconexión de los dos cloud será necesaria la configuración de diferentes recursos de red para habilitar una conexión segura y estable.

Se continuará con la automatización de todas las tareas de configuración y despliegue del paso anterior, y como último paso se tendrá la validación a través de la ejecución de simulaciones. Estas simulaciones correrán en un entorno híbrido heterogéneo, parte de la simulación correrá en el cluster on-premise (arquitectura ARM) y la otra parte correrá en el cluster de GCP (arquitectura AMD).

1.4. Metodología

Al principio se planteó la idea de aplicar Scrum[10] para hacer uso de las metodologías ágiles, sin embargo, tras un par de semanas de trabajo, se observó que la metodología que mejor se adaptaba a este proyecto era el iterativo. La razón por la que se cambió de metodología es debido a que el trabajo requiere volver a fases anteriores del proceso continuamente, esto es a causa del carácter investigativo y exploratorio del proyecto.
Por otro lado, se han mantenido reuniones con los tutores cada semana, o cada dos semanas como máximo, para evaluar el avance del proyecto, su estado, problemas y futuros objetivos a cumplir.

1.5. Organización de la memoria

Se ha optado por una estructura en secciones de cumpla con las recomendaciones en la elaboración de Trabajo de Fin de Grado (TFG) pautados por la EINA[2], y que sirven como presentación resumida del trabajo realizado. Los anexos numerados alfabéticamente corresponden a anexos donde se detallan otros aspectos más de implementación.

El capítulo 2 comprende conceptos que aclaran temas que se tratan en el trabajo, y además una pequeña exploración de las tecnologías planteadas. Se justificará la necesidad de las herramientas de automatización y gestión de recursos y se hablará de estos software, detallando tanto las usadas como las descartadas, ya que no son objetos de estudio en las

materias de estudio de este grado.

En el capítulo 3 se contemplará el análisis y el diseño del sistema. En esta sección se plantea los diferentes problemas a solucionar para así tener una visión clara de lo que se necesita resolver. También se presenta un diagrama en alto nivel de la arquitectura de la infraestructura usada para conseguir la meta propuesta.

Continuando con el capítulo 4, se podrá leer como se ha implementado el cloud híbrido. En estas lineas se redacta como se ha administrado y configurado tanto el orquestador como el espacio de trabajo en cloud público y privado, también se habla de la automatización del despliegue, así como de las diferentes herramientas usadas para conseguir este objetivo.

En el capitulo 5 contiene lineas acerca de como se ha validado el sistema, los diferentes tests llevados a cabo, la evaluación del sistema y las pruebas realizadas para validar el sistema.

Para terminar, el último capítulo (número 6) comprende las conclusiones y apuntes acerca del trabajo futuro que puede llevarse a cabo por compañeros de este grado, y finalmente una pequeña valoración personal sobre el trabajo realizado.

Capítulo 2

Conceptos y Tecnologías

2.1. Planificación de recursos distribuidos

La planificación de recursos computacionales distribuidos hace referencia al algoritmo responsable de determinar el orden y la distribución de carga de ciertas tareas que son puestas en ejecución en un sistema. Hoy en día existe múltiples herramientas que ayudan a llevar a cabo estas tareas, sin embargo, cada una de estas tiene sus peculiaridades y se debe elegir la herramienta de manera cautelosa para que se adapte a las necesidades exigidas.

La herramienta de planificación de recursos computacionales es la encargada de desplegar la tarea sobre los recursos hardware del cluster, asignar recursos, balancear la carga, gestionar los fallos, orquestar y monitorizar, y, todo esto sin sobrecargar el sistema para que todo el potencial se destine a la tarea. Este planificador debe asegurar la conexión de la red, la escalabilidad, una latencia en comunicaciones razonable, e incluso tolerancia a fallos tanto del planificador como de las tareas planificadas. Sumado a esto se encuentra la integración de entornos heterogéneos lo que añade dificultad al proceso.

2.2. Modelización de recursos de administración

La automatización de tareas de configuración del sistema es uno de los pilares fundamentales de este proyecto. Se dispone de múltiples configuraciones, ya sea, a nivel de administración de sistemas como puede ser, configuración de DNS, VPN o conexiones de red, o a nivel de gestión de recursos como son la orquestación de tareas. Todo ello lleva a la necesidad de automatizar los procesos para evitar configuraciones manuales en cada una de las ejecuciones. Así mismo, esta mecanización servirá de apoyo para futuros compañeros que sigan este proyecto y quieran usar las configuraciones y el trabajo elaborado.

Hoy en día esta necesidad de automatizar ha crecido significativamente, llegando a niveles en los que el administrador de sistemas realiza la resolución del problema una única vez, y tras ello se automatiza para futuros usos y únicamente se supervisa que la automatización no falle. Esto reduce en gran medida las tareas a realizar por el ingeniero y permite centrarse en otras tareas como puede ser la supervisión de que todo el sistema funciona correctamente. Este concepto, que cada vez es más habitual en estos últimos años, es lo que se intenta alcanzar en el desarrollo de este proyecto.

Por otro lado, el uso de recursos de un proveedor de cloud genera la necesidad de tener un control de lo que se dispone en esta plataforma. Esto es necesario por dos razones, la primera porque el administrador del sistema debe conocer los recursos que tiene y mantener una traza de estos, y en segundo lugar, porque estos recursos tienen un coste económico y mantener el control de estos recursos ayudará a reducir el coste, por ejemplo borrando todos los recursos sin olvidarnos de nada. Para esta tarea será necesario el uso de una herramienta de modelización, despliegue y destrucción de recursos, que permitan tener estos medios bajo control.

Estos dos conceptos anteriores, aunque a priori parecen diferentes, en realidad son lo mismo y buscan el mismo fin, facilitar la automatización de despliegues y configuraciones.

2.3. Herramientas

2.3.1. Kubernetes

Kubernetes [1] es una herramienta de planificación de recursos que permite administrar cargas de trabajo y servicios y ofrece una administración centrada en contenedores. Esta herramienta orquesta la infraestructura de cómputo, redes y almacenamiento, y fue diseñada con el objetivo de ofrecer un medio para poder desplegar un sistema de componentes, y brindar herramientas que faciliten este despliegue, escalado y administración. Esta herramienta destaca sobre todo por el enfoque a tolerancia de fallos que ofrece en los servicios que despliega, llegando a ofrecer herramientas muy útiles para gestionar esta tolerancia.

Cabe destacar que Kubernetes no es una Plataforma como Servicio (PaaS), este opera a nivel de contenedores y no a nivel de hardware, aunque ofrece características de PaaS como balanceo de carga o monitorización. Otro detalle a destacar, es que Kubernetes no despliega un código fuente ni compila una aplicación, por lo que para solventar esto se debería incluir el código fuente o aplicación en un contenedor para ser desplegado por Kubernetes.

Finalmente, cabe mencionar la dificultad de la gestión de la red. Hace que el tema de la conexión sea una tarea compleja exigiendo la implementación del modelo de red[7] ya que no lo trae por defecto.

2.3.2. Nomad

Nomad[2] es un orquestador de cargas de trabajo que permite desplegar, administrar y escalar cualquier aplicación, ya sea código fuente, trabajos en lote o contenedores. Nomad no balancea la carga de las peticiones a los nodos, para ello exige el acople de otro software como Nginx[3]. Tampoco permite descubrir los servicios para acceder a ellos, para ello se necesita Consul[4], una herramienta de la misma empresa (e.g. para ver el estado de los servidores es necesaria esta herramienta). Un punto destacable de Nomad es su instalación que solo requiere de la descarga de un binario que debe ubicarse en la máquina y ya se puede comenzar la puesta en marcha de servicios.

2.3.3. Slurm

Slurm[5] (Simple Linux Utility for Resource Management) es un sistema de código abierto, tolerante a fallos y escalable, que permite gestionar tareas y clústeres. Hay tres ideas en las que se centra esta herramienta:

- Proporciona una API que habilita la ejecución de tareas en un cluster tanto para administradores como usuarios.
- Orquesta tareas de manera que se aprovechen los recursos del cluster al completo, resolviendo conflictos de reparto de recursos a través de la gestión de colas de tareas con prioridad.
- Asigna a los usuarios acceso exclusivo o no exclusivo a nodos y sus recursos durante el tiempo que dura la ejecución de las tareas.

[1] https://kubernetes.io/es/
[2] https://www.nomadproject.io/
[3] https://www.nginx.com/
[4] https://www.consul.io/
[5] https://slurm.schedmd.com/

Cuando se habla de arquitectura en Slurm, esta dispone de un controlado centralizado que monitoriza los recursos disponibles en el cluster y las tareas que son ejecutadas en esta misma. Este controlador puede adquirir la propiedad de tolerancia a fallos a través de la configuración de un nodo controlador esclavo que adquirirá responsabilidad cuando el controlado maestro falle. Por otro lado, los nodos de cómputo del cluster corren un demonio que se encarga de recibir las tareas, ejecutarlas y devolver los resultados.

Existe también la posibilidad de configurar un demonio de base de datos en Slurm que se encargaría de almacenar datos de cálculos y datos sobre el cluster para un posible futuro estudio de los mismos. La configuración de este demonio se recomienda si se dispone de replicación en el nodo controlador, ya que se deben almacenar datos necesario para mantener la replicación.

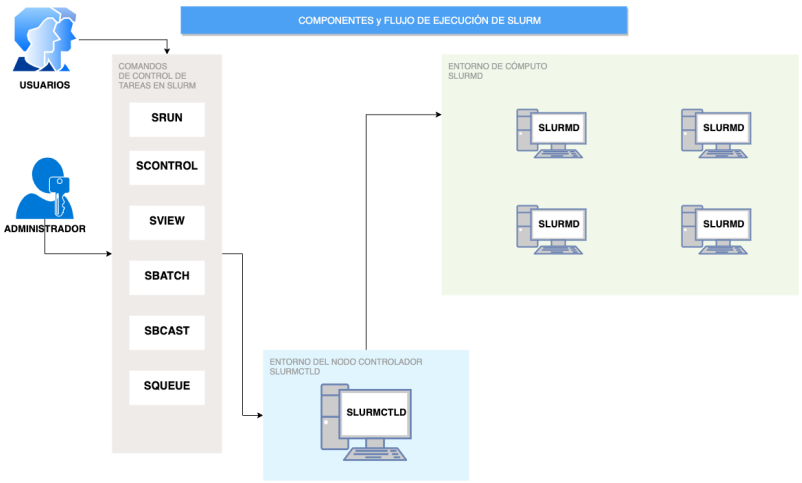


Figura 2.1: Esquema de los componentes y flujo de trabajo de Slurm

2.3.4. Ansible

Ansible[6] es una herramienta de modelización y automatización de recursos de administración de sistemas de tareas en el ámbito de las tecnologías de información. Con Ansible se puede automatizar el aprovisionamiento de software, la gestión de configuraciones y el despliegue de aplicaciones. Es una herramienta muy usada por administradores de sistemas y equipos de DevOps para gestionar servidores, configuraciones y aplicaciones de manera ágil y paralela.

La operativa de Ansible consiste en aplicar las configuraciones a través de SSH, estas configuraciones son acciones descritas en YAML. Además, las configuraciones se aplican en máquinas aisladas y no son paralelizadas.

En la figura 2.2 se puede observar la dinámica del funcionamiento de Ansible. Existe un nodo de gestión que accede a un inventario de *hosts*, es decir, al inventario de máquinas a las que aplicar el *Playbook* que es una configuración, declarado en lenguaje YAML, de los cambios que se deben aplicar en cada nodo del inventario.

[6]https://www.ansible.com/

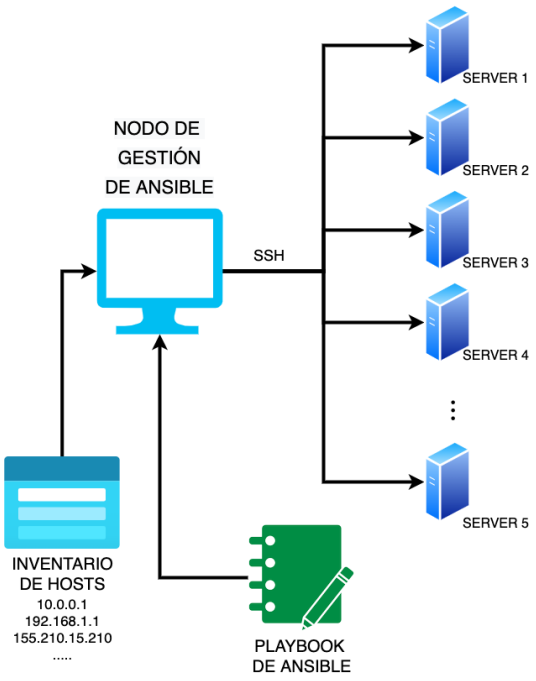


Figura 2.2: Esquema de los componentes y flujo de trabajo de Ansible

2.3.5. Terraform

Ansible también se usa para el aprovisionamiento de proveedores cloud, sin embargo, su uso no está tan extendido ni ofrece tan buen soporte como la de Terraform[7], que realiza esta tarea con una operativa mejorada, y además, ofrece una proceso más sencillo para la creación y destrucción de los diferentes recursos cloud instanciados. Terraform facilita la infraestructura como código permitiendo aprovisionar el entorno cloud de manera reutilizable y mantenible. La gestión de estos recursos se realiza a través de ficheros estructurados. Terraform usa una sintaxis general para definir los recursos y ejecuta estas sentencia para alcanzar un objetivo, la disponibilidad de estos recursos en el entorno cloud. Por tanto, Terraform es un lenguaje declarativo, ya que lo que se específica en las sentencias es lo que se consigue desplegar en la plataforma cloud.

Una gran ventaja de Terraform es que permite relacionar datos de los recursos creados gracias a su API, es decir, no aplica las configuraciones de manera aislada. Cuando creamos un recurso y luego otro, podemos acceder a contenido del primero desde el segundo. Esta ventaja facilita mucho la creación de ciertos flujos de trabajo (e.g. crear un subred y luego, crear un nodo en esa subred sin conocer la IP de esta porque se accede a través del primer recurso).

2.3.6. Proveedores cloud público

Hoy en día con el avance del Cloud cada vez son más los proveedores. Dentro de los más conocidos encontramos AWS (Amazon), GCP (Google) y Azure (Microsoft), cada proveedor tiene sus características que las diferencian de los demás y elegir uno u otro depende del proyecto que se quiera desplegar, de la familiaridad con el proveedor, del presupuesto, etc.
En el caso de GCP, las tareas que permite como proveedor se pueden encontrar creación de máquinas virtuales, creación de bases de datos, creación de recursos de red como firewalls, servidores VPN, servidores DNS, etc. Todos los ejemplos anteriores y muchos otros, son los posibles herramientas que nos proporciona este proveedor bajo un coste económico que generalmente suele ser un coste bajo demanda o uso.

[7] https://www.terraform.io/

2.3.7. Proveedores cloud privado

Otro proveedor que se tiene en este proyecto es el cluster de Raspberry Pis. Este cluster se ha se ha construido en el seno del equipo de investigación y está compuesto por 48 Raspberry Pis funcionando bajo el sistema Ubuntu 20.04. Además, incorpora un PC tradicional para labores de aislamiento en red privada y encaminamiento.
En concreto, los nodos del cluster está formado por dos tipos de equipos. Por un lado se tiene Raspberry Pis 4 arquitectura ARM con 4GB de memoria RAM LPDDR4 Quad Core Cortex-A72 a 1.5GHz, y por otro lado se tiene Raspberry Pis 4 arquitectura ARM con 8GB de memoria RAM LPDDR4 Quad Core Cortex-A72 a 1.5GHz.

Capítulo 3

Análisis y Diseño del sistema

3.1. Análisis del problema

La meta final que se tiene en este trabajo es conseguir un cloud híbrido **heterogéneo** en el que poder lanzar simulaciones de SED. Estas simulaciones podrán ser ejecutados en tres entornos posibles: Cloud Privado (cluster de Raspberry Pis), Cloud Público (nodos de SED) y Cloud Híbrido (mezclando los dos anteriores). Para conseguir esta meta será necesario pasar por diferentes fases.

Se necesita un planificador que ofrezca, principalmente, **una buena gestión de recursos, baja latencia de comunicaciones y escalabilidad**. Aunque **la escalabilidad no va a ser un objetivo de este proyecto** debido a que lo que se busca en este trabajo es una primera aproximación al entorno híbrido, no se debe olvidar esta característica puesto que se está preparando el terreno para un trabajo posterior que continúe progresando hacia el escalado. De nada serviría conseguir una aproximación híbrida con una herramienta que no sea capaz de escalar si no se va a poder continuar con el trabajo realizado en un futuro.

En una primera instancia Kubernetes se planteó como la herramienta de gestión de recursos y orquestación para ser usado en el proyecto. Tras estudiar esta idea más detenidamente se descartó con mucha rapidez debido que no cumplía una de las premisas que se exige en el proyecto y es que ofrezca una conectividad de red sencilla y con baja latencia. El problema reside en que Kubernetes tiene las redes internas aisladas, las dirección IP que asigna a los pods[1] y a los contenedores pueden cambiar a lo largo de su vida, no hay acceso entre microservicios y la capa de aplicación no es visible. Además un último elemento de peso que hace descartar esta herramienta es la cantidad de modificación de cabeceras de tramas de red que realiza, llevando a latencias altas debido a paquetes grandes, un alto tráfico en red y una demora extra a la hora de encapsular y des-encapsular las cabeceras.

Nomad también se consideró como herramienta para la orquestación de tareas en el cluster de *Raspberry Pis*. Sin embargo, los planificadores que ofrece no eran los adecuados para lanzar tareas que exigen de paralelización para un entorno HPC. Tras investigar en la documentación de la herramienta[4] se obtuvo la información de que Nomad permitía el uso de un orquestador personalizado, lo que encaminó hacia la programación de uno. Debido a la poca información del orquestador personalizado se decidió ponerse en contacto con la comunidad de Nomad para conseguir más información. Se consiguió que uno de los desarrolladores de Nomad contestase a las dudas que se les planteó [6], pero no se tuvo éxito. Una característica que tiene Nomad es la limitación de 6100 nodos[9]. Usar una herramienta cuyo propio equipo ha conseguido como mucho el uso de 6100 nodos nos está acotando el número de nodos a ser usados. Siendo que se está preparando un proyecto para ser escalado en un futuro no es de gran utilidad usar una herramienta que nos acota el escalado.

Finalmente, el planificador de tareas escogido para solventar el problema de orquestación

[1] Un pod es la unidad mínima de ejecución en Kubernetes

de recursos y tareas ha sido Slurm. La elección de esta herramienta se debe a que en un futuro se quiere simular tareas complejas y costosas, incluso se alcanzará el uso de un número de nodos considerablemente grandes acercándose a tareas de computación de altas prestaciones. Slurm ofrece esta posibilidad de escalar a un número de nodos grande por lo que es clave para el trabajo a futuro.

Por otro lado, si se observa los usuarios de Slurm podemos ver entidades gubernamentales, universidades, empresas, e incluso se hace uso para la administración de flujos de trabajo para varios proyectos pertenecientes a la lista TOP500[2]. Todo ello hace a Slurm una herramienta muy potente, con gran soporte y con grandes capacidades computacionales, ofreciendo gran granularidad para la gestión de recursos y la orquestación de tareas. Para terminar, Slurm facilita la arquitectura de red y además ofrece latencias bajas por lo que cumple las premisas planteadas al comienzo de este capítulo.

Para decantarse por un proveedor en este proyecto se ha optado por ver si alguno ofrece recursos o soporte para Slurm y el mejor candidato ha sido Google. Esto es debido a que desde Slurm ofrecen ciertas directrices[11] de como desplegar un proyecto en este proveedor, sin embargo, muchas de las indicaciones no son concretas, y algunos recursos que ofrecen no compilan correctamente, por lo que exige trabajo por parte del administrador. No obstante, esto hace a Google el proveedor a ser escogido. Además, tras haber trabajado previamente con los otros dos proveedores más conocidos, Azure y AWS, sin duda Google es el más fácil de usar, ya sea por su interfaz clara y ordenada, como el tema de permisos para los administradores de los recursos.

Continuando con el análisis, se dispone de un cluster de múltiples nodos prácticamente idénticos que van a necesitar disponer de las mismas configuraciones de red, permisos, usuarios, paquetes, etc. Para evitar realizar estas tareas repetidamente en todos los nodos, se hará uso de este software para la automatización. Con la ayuda de Ansible, solo se necesita resolver los problemas o realizar las configuraciones una única vez. Una vez conocidas las tareas que tenemos que replicar en todos los nodos, se automatiza y se aplica en todos los nodos simultáneamente en una única ejecución.

Por otro lado, en este trabajo se necesita hacer uso de recursos de un proveedor cloud para generar un espacio de trabajo híbrido. Ocurre un problema importante con estos recursos y es que suponen un coste económico que puede llegar a ser muy elevado si no se destruyen estos recursos una vez se hayan terminado de hacer uso de estos. Para mantener de manera organizada los recursos que se necesitan generar y usar, se hacen uso de herramientas de Infraestructure as a Service (IaaS) como Terraform. Terraform permite generar estos recursos y mantiene, en un fichero de estado, un listado con los medios creados, de esta manera, cuando se realice la destrucción de estos de manera automática, este sabe que medios quedan pendientes de destruir. Cuando se consigue eliminar todos los medios del fichero de estado, este queda vació indicando que ya no hay recursos pendientes.

3.2. Diseño del sistema y Arquitectura

Del planificador se hablará mas en detalle en el anexo A, pero es necesario conocer que contiene dos tipos de nodos, el nodo controlador y los nodos de cómputo, para comprender correctamente el diagrama de la figura 3.1. En esta figura se observa la arquitectura en alto nivel propuesta para la resolución de los problemas planteados en la sección 3.1. Se puede ver que se propone un cloud híbrido compuesto por dos elementos principales, por un lado el cluster on-premise y por otro lado el cluster GCP.

El cluster on-premise está formado por una subred interna aislada del exterior donde tendremos los nodos de cómputo y el nodo controlador. Estos nodos tienen salida al exterior gracias al router que es una máquina ubicada en la EINA que realiza tres funciones principales: router, túnel VPN y servidor DNS.

El cluster de GCP está bajo una Virtual Private Cloud que contiene una subred de nodos

[2]https://www.top500.org/

de cómputo y tres elementos principales que permiten el trafico entre los dos cluster: una zona DNS, una conexión VPN y unas reglas de firewall.

Es fundamental la interconexión de estos dos clústeres, para ello será necesario establecer un túnel VPN que asegure la conexión punto a punto desde un cluster al otro. También se establecerá una resolución de nombres en ambas dirección a través de la creación de servidor DNS.

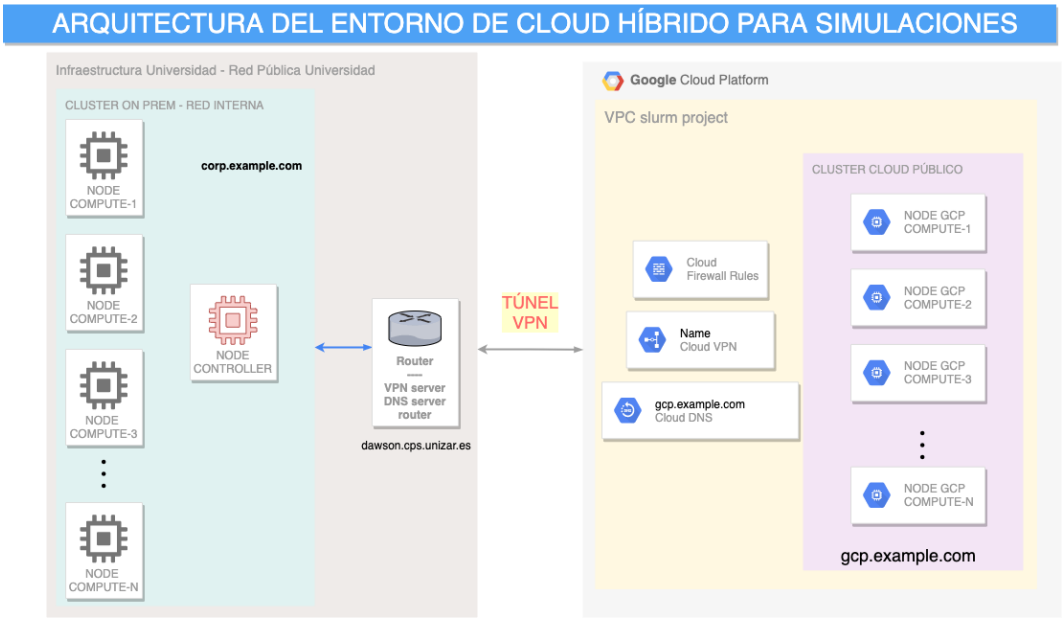


Figura 3.1: Diagrama en alto nivel de la arquitectura propuesta para el sistema.

Por último se hablará de la seguridad, para ello se ha puesto en marcha un firewall tanto en el router del cluster on-premise, como en el cluster de Google, con esto se consigue restringir el tráfico y filtrar únicamente los paquetes de red necesarios. Muchos de los recursos del proveedor tienen que estar expuestos para poder ser usados y esta exposición crea riesgos que hay que mitigar, y el firewall es un elemento que ayuda en este proceso.

Capítulo 4

Implementación del Cloud Híbrido

Es relevante mencionar que el sistema que se despliega se encuentra en un entorno heterogéneo. Muchos de los proyectos que se están desarrollando para realizar simulaciones con Slurm están bajo entornos homogéneos, es decir, múltiples máquinas de las mismas características. Sin embargo, en este caso, por un lado se tienen las máquinas del proveedor Google cuya arquitectura es AMD y que además son máquinas virtuales sobre servidores cuyas características no se conocen, y, por otro lado, se tienen las Raspberry Pis, que se basan en una arquitectura ARM.

En este proyecto se concede mucha importancia a configurar y administrar correctamente todos los elementos del sistema final a desplegar. Durante las siguientes lineas se podrá leer un resumen de las implementaciones realizadas y en los anexos citados se podrán ver las configuraciones detalladas de estas implementaciones.

4.1. Interconexión entre Cloud Privado y Público

La interconexión entre los dos entornos es fundamental para poder conectar los dos clústeres, sin esta conexión se tendrían nodos independientes sin comunicación con los del otro. Para la interconexión se ha tenido varias opciones un recurso que ofrece el proveedor de cloud llamado *cloud interconnect* y un túnel VPN convencional. La primera consiste en una conexión directa por cable a la infraestructura del proveedor, en este caso Google, sin embargo el coste de este recurso es 1000 veces superior al de la segunda opción, un túnel VPN.

La conexión se llevará a cabo por un túnel ya que es la opción más adecuada para el presupuesto y la más viable. Un túnel establece la conexión de un punto a otro y permite el tráfico de paquetes de red a través de un protocolo que se establece a la hora de configurar. En este trabajo, ese protocolo será IPSec, que asegura las comunicaciones sobre la capa IP autentificando y cifrando todo paquete que se encamine por este túnel.

Para terminar con la interconexión será necesario establecer un servidor DNS al que poder realizarle peticiones de resolución de nombres de dominio, de esta manera podemos referenciar a las máquinas a través del nombre y no por sus direcciones IP.

4.2. Cloud Privado

4.2.1. Router

El router que se usa para administrar el tráfico es una máquina ubicada en la EINA que, entre otras tareas, nos permite el acceso al cluster on-premise. Esta máquina realiza muchas funciones dentro del sistema que se configura durante este trabajo.

En primer lugar, y como el nombre indica, la función es de router para la subred de máquinas del cluster on-premise, con esto se consigue, entre otras cosas, asignación automática de IPs según la dirección MAC, redirección de paquetes y encaminamiento de tráfico.

Continuamos con la función del túnel VPN que es importante ya que gracias a este se puede tener el primer extremo de la conexión por Túnel IPSec cifrado desde las Raspberry Pis hasta los nodos de GCP, de esta manera se evita exponer los nodos al exterior y se puede tener los nodos en redes privadas.

Otro papel que realiza este equipo es de servidor DNS. Esta funcionalidad nos permite resolución de nombres directa entre los dos clústeres, de esta manera los usuarios y administradores pueden hacer uso de nombres de domino y olvidarse de memorizar direcciones IP.

Por último, otra tarea que realiza el router es de firewall. En el equipo se tiene configuradas unas reglas de firewall para permitir y denegar tráfico entrante y saliente, así como de redirigir paquetes.

Todas estas funcionalidades explicadas en estas lineas son claves para poder dar pie a un entorno híbrido, sin estas, no sería posible la conexión entre los dos entorno. Se podrá ver en detalle la implementación de estas funcionalidades en el anexo D.

4.2.2. Cluster on-premise

En el cluster on-premise se han realizado múltiples configuraciones en los nodos. Entre estas se encuentra la configuración DHCP, instalación de paquetes, configuración del SO, etc. No obstante, la más importante es la puesta en marcha de todo el sistema de Slurm ya que es lo que permitirá todo el funcionamiento del sistema de simulaciones. La administración de la herramienta Slurm ha involucrado a tareas de documentación, configuración, y administración de sistemas. El trabajo comienza con la documentación al detalle del funcionamiento de la herramienta y de su posible parametrización y ficheros de configuración.

Cuando se ha tenido claro como funciona esta herramienta se ha podido proceder a la configuración de este software. Esta configuración comienza con la administración del nodo controlador, y continua con la configuración de los nodos de cómputo. El nodo controlador corre el demonio llamado *Slurmctld* y es el encargado de mantener activo el cluster, monitorizar el resto de demonios de Slurm y recursos, acepta los *jobs* y asigna recursos en términos generales. En resumen, este nodo es el que gestiona y mantiene activo todo el cluster. Los otros tipos de nodos que se tienen, son los nodos de cómputo y estos corren el demonio *Slurmd* y son los encargados de monitorizar, aceptar, lanzar y matar los *jobs*.

Toda la infraestructura de Slurm tanto para el nodo controlador como el nodo de cómputo se estructura en base a un fichero de configuración que tienen todos los nodos, y que además debe ser exactamente igual para todos. Cada vez que se desee añadir un nuevo nodo, aumentar el tamaño de la RAM de un nodo o cambiar la ubicación de un fichero de *logging*, se debe modificar este fichero en todos los nodos.

Slurm permite agrupar los nodos del cluster en las llamadas **particiones**. Agrupar los nodos por particiones es útil a la hora de lanzar las tareas ya que da la opción de elegir dónde correrá la tarea. En la figura 4.1 podemos ver que el cluster desplegado dispone de tres particiones.

- **TFG:** Partición compuesta exclusivamente por nodos on-premise.
- **GCP:** Partición compuesta exclusivamente por nodos GCP.
- **MIX:** Partición compuesta por todos los nodos del cluster, tanto nodos on-premise, como nodos GCP.

Cuando aparece un asterisco al lado del nombre de la partición, esto indica que es la partición por defecto, es decir, donde se ejecutarán las tareas a menos que se le indique lo contrario. En nuestro caso la partición por defecto es **tfg.**

Continuado con información de Slurm, otro de los datos que podemos observar en la figura 4.1 es el estado de los nodos. Se ve como algunos de los nodos tiene el estado *IDLE*, que indica que está listo para procesar tareas, y, otros tienen el estado *DOWN* que indica que son nodos caídos o nodos no operativos.

```
ubuntu@slurmctld:~$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
tfg*         up      10:00      3  down* ubuntu[4-5,40]
tfg*         up      10:00     24   idle ubuntu[2-3,6-8,21-39]
gcp          up      10:00      4  down* gcp1.gcp.example.com,gcp2.gcp.example.com,
mix          up      10:00      7  down* gcp1.gcp.example.com,gcp2.gcp.example.com,
mix          up      10:00     24   idle ubuntu[2-3,6-8,21-39]
```

Figura 4.1: Salida del comando *sinfo* que muestra el estado del cluster de Slurm.

Para lanzar un programa en alguno de los nodos del cluster de Slurm es suficiente con el comando **srun script.sh**, o si se quiere ejecutar un único comando **srun hostname**, este comando lanza una tarea en **uno** de los nodos del cluster que estén listos para trabajar. Cuando se quiere lanzar un programa o comando en más de un nodo se usará la opción *-N*, tal que **srun –N 10 hostname**, y esto ejecutará el comando *hostname* en 10 nodos del cluster.

Para un conocimiento más profundo de la herramienta se recomienda consultar la sección A.4.

La instalación, configuración y puesta a punto de Slurm se encuentra automatizado con Ansible. Gracias a este herramienta no se tiene que ir nodo por nodo realizando configuraciones unitarias en cada máquina.
Siendo este un trabajo que se puede replicar y automatizar es ideal hacer uso de esta herramienta. Se ha creado un *Playbook* de Ansible que con un simple comando se lanza toda la instalación en cuantos nodos se necesite.

Modificar el fichero de configuración de Slurm en todos los nodos es una tarea pesada, para ello se ha desarrollado otro módulo de Ansible que se encarga de ello, y de esta manera con una única aplicación todos los nodos especificados reciben el reemplazo del fichero.

```
---
# Replace slurm.conf file
- name: Replace slurm.conf with new configuration
  become: true
  template:
    src: slurm.conf.j2
    dest: "/etc/slurm-llnl/slurm.conf"
    owner: ubuntu
    group: ubuntu
    mode: 0644
  register: replaceDone

#If it's a compute node then reload slurmd daemon
- name: Reload slurmd
  become: true
  service:
    name: "{{ slurmd_service }}"
    state: reloaded
  when: "'compute' in slurm_roles"

#If it's a controller node then reload slurmd daemon
- name: Reload slurmctld
  become: true
  service:
```

```
    name: "{{ slurmctld_service }}"
    state: reloaded
  when: "'controller' in slurm_roles"
```

Código 4.1: Código de Ansible que reemplaza el fichero de configuración y reinicia el demonio correspondiente.

En el anexo A se puede ver más detalladamente la configuración de esta herramienta, y la automatización de la misma se puede ver en la sección C.1.

4.3. Cloud Público

4.3.1. Configuración de recursos de la Virtual Private Cloud

La VPC de Google también necesita de cierta administración para conseguir establecer las conexiones de manera correcta. En primer lugar, se debe crear los nodos, para ello se necesita establecer una imagen con la misma configuración de Slurm que el cluster on-premise. Para continuar, se procede a la creación de una zona DNS en la VPC a la que poder dirigir peticiones para resolver los nombres de los nodos instanciados en este proveedor. Otra configuración necesaria es la de un Cloud VPN, un recurso del proveedor que permite establecer el otro extremo del Túnel IPSec del que se ha hablado en la sección 4.2.1. Como última tarea, se tiene la configuración del firewall, lo cual es muy importante cuando se habla de cloud público. La importancia del firewall reside en la cantidad de ataques de seguridad existentes en el cloud público por lo que establecer un firewall restrictivo es clave.

Si se desea comprender al detalle como se han llevado a cabo estas implementaciones se recomienda leer el anexo B.

4.3.2. Desarrollo de la automatización del despliegue en Cloud

Crear los recursos manualmente en el proveedor es un proceso que puede llevar mucho tiempo, y además existe una gran probabilidad que alguno de ellos se configure erróneamente, nos confundamos en algún parámetro o incluso olvidarnos de algún recurso. Para solventar esto, se hace uso de la herramienta de Terraform para declarar los recursos que se necesitan, sus parámetros de configuración y el estado deseado. De esta manera se declara una única vez los recursos, se aplica el módulo de Terraform y este se autentifica en el proveedor y crea los recursos tal y como se le ha pedido. Posteriormente podemos eliminar todos y cada uno de estos recursos con otro simple comando, ya que Terraform guarda el estado del sistema desplegado en un fichero y es capaz de rastrear los cambios que ha habido en este sistema.

```
resource "google_compute_instance" "default" {
  count        = length(var.node_names)
  name         = var.node_names[count.index]
  machine_type = var.machine_type
  zone         = var.zone


  boot_disk {
    initialize_params {
      type  = "pd-ssd"
      image = var.instance_image_source
    }
  }

  network_interface {
    network    = var.vpc_network
    subnetwork = var.vpc_subnetwork

    access_config {
      network_tier = "STANDARD"
```

```
    }
  }

  hostname = "${var.node_names[count.index]}.${var.gcp_dns_domain
      }"

  service_account {
    email  = data.google_service_account.default.email
    scopes = ["cloud-platform"]
  }

  metadata_startup_script = file("${path.module}/startup.sh")
}
```

Código 4.2: Instanciación de máquinas virtuales en el proveedor Google haciendo uso del módulo programado de Terraform.

En la sección C.3 se explica al detalle como funciona esta herramienta y como se ha llevado a cabo el desarrollo del módulo de automatización.

Capítulo 5

Validación y Pruebas

5.1. Validación

Para validar la automatización del despliegue de los recursos en GCP cuando se usa un módulo de Terraform se ha usado Chef Inspec[1]. Este framework nos permite realizar tests para comprobar el estado del sistema desplegado. El funcionamiento consiste en especificar el estado que se desea de tener tras el despliegue de recursos y una vez desplegado se lanza este test para comprobar que los recursos se han creado correctamente. Por otro lado, para asegurarse de que se han destruido correctamente los recursos y que no se han quedado en el proveedor generando un gasto, se ha creado un test para comprobar que los recursos que se quieren desplegar no están presentes en el proveedor.

En cambio, para validar el modulo de Ansible se han creado test para lanzar los Playbooks. Estos tests hacen uso del módulo programado de Ansible para configurar el sistema operativo objeto, y configurar los paquetes y herramientas que se deseen. Concretamente, lo que se busca con el test es comprobar que se ha podido instalar Slurm, tanto en nodos de cómputo como en el nodo controlador, y además se ha podido configurar los ficheros y permisos necesarios.

5.2. Pruebas

Para demostrar que el sistema desplegado sirve para el propósito por el que en un principio fue diseñado, se han llevado a cabo dos tipos de pruebas que se redactan en las siguientes secciones. Por un lado se han desarrollado pruebas preliminares en el entorno de simulación desplegado para demostrar su potencial. Una vez validado el sistema se han puesto en marcha en un entorno real de simulación distribuida con el software desarrollado por el compañero *Álvaro Santamaría de la Fuente* en su TFG[3].

5.2.1. Pruebas en entorno de prueba

En estas pruebas se ha explotado el potencial del cluster de Slurm, llevando a cabo tareas en los tres entornos posibles (Cloud Público, Cloud Privado, Cloud Híbrido). Además estas pruebas sirven para dejar documentación de las posibles ejecuciones que se pueden hacer con Slurm en el sistema desplegado.

Entre las pruebas, destacar ejecuciones realizadas en el cluster on-premise aprovechando todos los recursos posibles y sometiendo varias tareas a la vez para ver como Slurm gestiona la asignación de recursos y la cola de tareas. Como en estas pruebas solo se busca ver el potencial de Slurm asignando recursos y gestionando esta cola de tareas, el trabajo ejecutado consistía en un programa que lanzaba un comando para ver el nombre del equipo, y, a continuación dormía durante 30 segundos simulando que realiza un trabajo.
Por otro lado, también se ha realizado la misma prueba en el cluster de google, y por último, en el cluster híbrido mezclado nodos de ambos tipos.

[1] https://docs.chef.io/inspec/

Una de las tareas que demuestra que la infraestructura planteada es correcta consiste en la **prueba de paralelización de tareas**. Esta prueba intenta verificar que Slurm es capaz de gestionar los recursos de los nodos adecuadamanete. Para ello se pone en marcha 4 tareas:

```
#!/bin/bash
#fichero batchTask.sh

srun -l --partition=gcp --job-name=cloud   -N 4 './script.sh' & #A
srun -l --partition=tfg --job-name=onprem  -N 10'./script.sh' & #B
srun -l --partition=mix --job-name=hybrid  -N 6 './script.sh' & #C
srun -l --partition=mix --job-name=hybrid2 -N 6 './script.sh' & #D
```

- A: Cluster GCP usando 4 nodos.
- B: Cluster on-premise usando 10 nodos.
- C: Cluster híbrido (mezcla de las anteriores) usando 6 nodos.
- D: Cluster híbrido (mezcla de las anteriores) usando 6 nodos.

Se someten las 4 tareas simultáneamente y se espera que Slurm gestione los nodos de tal forma que si existen recursos pueda lanzar mas de una tarea por nodo, y efectivamente es capaz de hacerlo y gestiona los recursos de tal manera que ejecuta mas de una tarea paralelamente.

Otra prueba a recalcar es la ejecución de tareas exclusivas, es decir, que si una tarea exclusiva está ejecutándose en un nodo, Slurm no puede alojar otra tarea paralela en el mismo nodo. Se lanza la misma prueba anterior pero con la opción de exclusividad y se observa exitosamente que Slurm es capaz de gestionar este problema.

Las conclusiones obtenidas son que el sistema funciona correctamente y Slurm realiza una gestión de recursos muy buena, incluso permitiendo múltiples parametrizaciones para personalizar esta gestión de recursos.

5.2.2. Pruebas en un entorno real de simulación distribuida

Una vez se ha comprobado que el entorno funciona correctamente, se ha pasado a probar el simulador distribuido. Para ello, en primer lugar, se ha distribuido una copia del binario del simulador en todos los nodos, teniendo en cuenta que los nodos de Google necesitan el binario en arquitectura AMD y los del cluster on-prem las necesitan en arquitectura ARM. Una vez se dispone del binario y las redes a simular, se han realizado varias pruebas de simulaciones recogiendo los datos correspondientes para ser analizados a posteriori.

De todas las pruebas realizadas, una muy interesante que cabe destacar es la de lanzar múltiples **simulaciones en paralelo en varios entornos cloud**. Esta prueba consiste en verificar que al lanzar una simulación en Cloud Privado, otra en Cloud Público y otra en Cloud Híbrido, Slurm es capaz de gestionar los recursos para que no haya colisiones y conflictos a la hora de compartir estos medios. Tras lanzar esta prueba y observar como Slurm es capaz de encolar (figura 5.1) las tareas en la cola de procesos y dejar pendiente aquellas que todavía no tienen recursos disponible, se ha podido dar por validado este sistema y esta prueba.

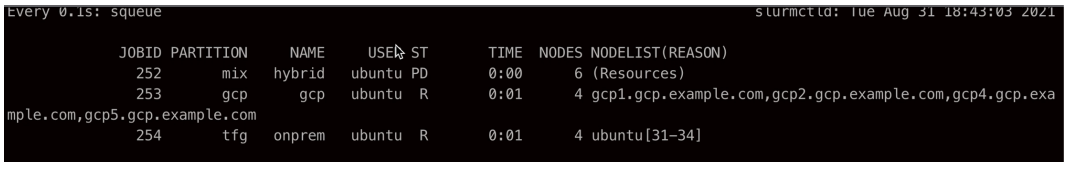


Figura 5.1: Cola de procesos de Slurm con tareas paralelas en varios entornos cloud.

Capítulo 6

Conclusiones

Para terminar esta memoria se redactan las siguientes lineas sobre las conclusiones obtenidas durante el desarrollo de este proyecto, también se darán nociones del posible trabajo a futuro, y para finalizar, una valoración personal que se ha alcanzado tras acabar el trabajo.

6.1. Conclusiones

Tras muchas horas de trabajo, personalmente, puedo decir que se ha logrado el objetivo. Se ha podido **automatizar el proceso de despliegue de simulaciones en Cloud Privado, Cloud Público y Cloud Híbrido**, y además, se ha automatizado el proceso de creación de recursos de estos tres clouds.

Después de analizar múltiples herramientas se pudo elegir Slurm como la herramienta de orquestación y planificación de recursos distribuidos que cumple con las premisas expuestas. No solo se ha conseguido su configuración sino que también se ha automatizado su proceso de instalación y configuración.

Este software cumple con los requisitos de baja latencia en comunicaciones, una gestión de la red sencilla, es escalable y gestiona los recursos del sistema de manera adecuada. No solo gestiona los recursos como queremos sino que permite múltiples parametrizaciones y configuraciones para mejorar y adaptar a nuestro trabajo esta gestión, por lo que puedo concluir diciendo que la herramienta es la adecuada para este y futuros trabajos.

En cuanto al entorno híbrido, también se da por conseguido este objetivo. Se ha podido **establecer y automatizar la interconexión del Cloud Privado con el Cloud Público**, y los recursos necesarios para crear el entorno híbrido.

El reto de conectar los nodos del cluster de GCP con los nodos del cluster on-premise ha sido interesante y se ha podido solucionar haciendo uso de recursos convencionales como son los túneles VPN y un servidor DNS. Es cierto que la velocidad de conexión entre los dos clústeres no es la mejor posible, pero es la que se acomoda al presupuesto y limitaciones.

Por último, el objetivo de lograr llevar a cabo pruebas en un entorno real de simulaciones distribuidas ha sido conseguido exitosamente. Se han llevado a cabo simulaciones, que **ejecutan parte en Cloud Privado y parte en Cloud Público**, con resultados de eficiencia muy buenos para los TFG de otros compañeros del proyecto, por lo que con este logro se ha podido validar el sistema desplegado.

6.2. Trabajo a futuro

Como trabajo a futuro se plantea conseguir la instanciación de máquinas de Google bajo demanda, es decir, si las tareas que haya en la cola del cluster son elevadas y la cantidad de recursos que haya libres son escasas, instanciar automáticamente nodos en Google para generar más recursos. Incluso se podría conseguir que si una tarea requiere de más nodos de los disponibles en el cluster híbrido, instanciar más nodos en Google según el trabajo y los requerimientos.

Por último, como he mencionado antes, Slurm es una herramienta muy compleja que permite mucha flexibilidad a la hora de configurar el cluster y sería un buen proyecto a futuro, documentar y probar todo este potencial ya que puede ser clave para mejorar las simulaciones distribuidas de SED.

6.3. Valoración Personal

El despliegue de un sistema preparado para simulaciones en un entorno híbrido ha supuesto un nuevo reto. Tras estos cuatro años de mi formación como ingeniero he estudiado diferentes sistemas distribuidos y he visto algunos otros en Cloud Público, sin embargo, el reto de realizar el despliegue de un sistema distribuido en un entorno híbrido haciendo uso de Cloud Público y Privado se convertía en algo nuevo.

Personalmente este trabajo me ha gustado mucho, si es verdad que ha creado ciertos momentos de gran dificultad debido a que me enfrentaba a un reto nuevo y con un tema de investigación poco evolucionado, pero esto también ha sido interesante para fomentar nuevas capacidades. Me gustaría destacar que el no tener un trabajo previo en el que basarse hacía que el reto fuese más interesante a la vez que complicado. Al adentrarme en este proyecto sabía de antemano de que se trataba de un trabajo de investigación por lo que tenía claro que habría momentos complicados en los que se podría topar con caminos sin salida, no se pudo evitar y hubo que pasar por este bache, sin embargo, esto no frenó el proceso y finalmente se consiguieron cumplir los objetivos.

Sin ninguna duda estoy contento con el trabajo realizado, crear un entorno híbrido para computación de altas prestaciones es un logro muy importante, incluso más cuando he podido colaborar en un proyecto de investigación, y no solo eso, sino que se ha aportado trabajo en un artículo publicado en la GECON 2021[14] y que además ha sido elegido como mejor articulo de la conferencia. Con este trabajo puedo aportar una base para otras personas que decidan adentrarse en un proyecto de simulación distribuida en un entorno híbrido heterogéneo. Sin duda alguna recomendaría a futuros alumnos adentrarse en este proyecto, y sobre todo, con los profesores que han guiado este trabajo, ya que sus conocimientos han sido claves para poder avanzar y realizar un buen proyecto.

Bibliografía

[1] Ali et al. Asgary. A Drive-through Simulation Tool for Mass Vaccination during COVID-19 Pandemic. https://pubmed.ncbi.nlm.nih.gov/33182336/.

[2] EINA. Elaboración De La Propuesta Y La Memoria Del Tfg/Tfm. https://eina.unizar.es/sites/eina.unizar.es/files/archivos/secretaria/20210706_instrucciones_tfg_tfm.pdf.

[3] Álvaro Santamaría de la Fuente. "Diseño e implementación de un simulador distribuido de eventos discretos con mecanismos de balanceo de carga". Trabajo de fin de Grado. Universidad de Zaragoza, 2021.

[4] Hashicorp. Scheduling in Nomad. https://www.nomadproject.io/docs/internals/scheduling/scheduling.

[5] Red Hat. BIND 9 Configuration Reference. https://bind9.readthedocs.io/en/latest/reference.html.

[6] Hayk Kocharyan. Custom scheduler documentation. https://discuss.hashicorp.com/t/custom-scheduler-documentation/22624.

[7] Kubernetes. Installing Addons. https://kubernetes.io/docs/concepts/cluster-administration/addons/.

[8] Fidel Reviriego Navarro. "Diseño e implementación de un simulador distribuido de alta escala de sistemas de eventos discretos". Trabajo de fin de Master. Universidad de Zaragoza, 2021.

[9] Nomad. The Two Million Container Challenge. https://www.hashicorp.com/c2m.

[10] Scrum ORG. Scrum. https://www.scrum.org/.

[11] SchedMd. Slurm on GCP Platform. https://github.com/SchedMD/slurm-gcp.

[12] Vladimir Smirnov. How to set up a VPN between strongSwan and Cloud VPN. https://cloud.google.com/community/tutorials/using-cloud-vpn-with-strongswan.

[13] Onur Ulgen y Ali Gunal. Simulation in the Automobile Industry. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.91.4817&rep=rep1&type=pdf.

[14] José Ángel Bañares Unai Arronategui y Álvaro Santamaría et al. José Manuel Colom Hayk Kocharyan. Workload Evaluation in Distributed Simulation of DESs. September, 2021.

Anexo A

Configuración de Slurm

A.1. Nodo controlador - Slurmctld

Para la configuración del nodo controlador de Slurm se comienza con la instalación de Slurm. Para ello se hará uso del módulo de Ansible que instala y configura los directorios. A continuación se explica lo que hace este módulo cuando se trata del nodo controlador el que está siendo configurado.

A.1.1. Configuración del controlador usando el módulo de Ansible

1. Crea un grupo y un usuario llamado **slurm**.
2. Actualiza la cache de APT[1].
3. Instala los paquetes de Slurm que corresponden al nodo controlador, que en este caso es *slurm-wlm*.
4. Crea el directorio */var/spool/slurm-llnl/slurmctld* donde se guardará el estado del demonio *Slurmctld*, y concede permisos al usuario Slurm creado previamente.
5. Crea el directorio */var/log/slurm-llnl/* donde se ubicará el fichero de logging del demonio
6. Instala los paquetes restantes necesarios para Slurm *(slurm-wlm-doc y slurm-client)*.
7. Copia el fichero de configuración **slurm.conf** y **cgroup.conf** necesarios para el funcionamiento del software.
8. Configura Munge, un software de autenticación que usa Slurm por debajo para llevar a cabo las conexiones entre nodos. Al ser el nodo controlador, este copia la clave **munge.key** que se usa para la autenticación y la ubica en la máquina que lanza el Playbook de Ansible. Esto se hace así ya que los nodos de cómputo necesitan esta clave para interactuar con el controlador y esta debe ser la misma para todos.

A.1.2. Configuración de red

Se modificará el servidor DNS para que haga uso del que se encuentra ubicado en el router. Para ello se editará el fichero */etc/systemd/resolved.conf* y se asegurará de estar presente las siguientes lineas:

```
DNS=192.168.1.254
Domains=corp.example.com
```

Código A.1: Configuración del DNS para el nodo controlador en el fichero '/etc/systemd/resolved.conf'

[1]APT es el gestor de paquetes de ubuntu https://help.ubuntu.com/kubuntu/desktopguide/es/apt-get.html

A.1.3. Validación de la configuración

Cuando se hayan completado los pasos se validará que el software está funcionando. Para ello, se comprobará el estado del demonio con el siguiente comando **$ sudo service slurmctld status** y se asegurará que no hay ningún error.
Otra comprobación que se puede realizar es ejecutando el comando **$ sudo sinfo**. Este comando nos devuelve el estado del cluster, si la salida no muestra ningún error, en ese caso el cluster está correcto (ver figura A.1).

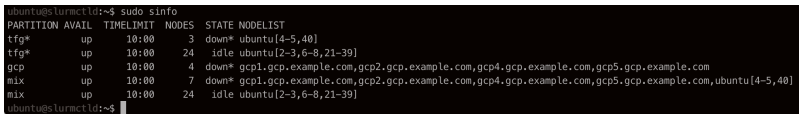


Figura A.1: Salida del comando *sudo sinfo* en un cluster completamente configurado.

A.2. Nodo de cómputo on-premise - Slurmd

El nodo de cómputo tiene una configuración similar a la del controlador, pero difiere en algún paquete y en algún fichero de configuración.

A.2.1. Configuración del nodo de cómputo usando el módulo de Ansible

1. Crea un grupo y un usuario llamado **slurm**.
2. Actualiza la cache de APT[2].
3. Instala los paquetes de Slurm que corresponden al nodo de cómputo, que en este caso es *slurm-wlm*.
4. Crea el directorio */var/spool/slurm-llnl/slurmd* donde se guardará el estado del demonio *Slurmd*, y concede permisos al usuario Slurm creado previamente.
5. Crea el directorio */var/log/slurm-llnl/* donde se ubicará el fichero de logging del demonio
6. Instala los paquetes restantes necesarios para Slurm *(slurm-wlm-doc y slurm-client)*.
7. Copia el fichero de configuración **slurm.conf** y **cgroup.conf** necesarios para el funcionamiento del software.
8. Configura Munge. Como es el nodo de cómputo, en este caso lo que hará es copiar el *munge.key* genreado por el nodo controlador y lo ubicará en el direcotiro */etc/munge*

A.2.2. Configuración de red

Primero se modificará el servidor DNS para que haga uso del que se encuentra ubicado en el router. Para ello se editará el fichero */etc/systemd/resolved.conf* y se asegurará de estar presente las siguientes lineas:

```
DNS=192.168.1.254
Domains=corp.example.com
```

Se continuará con la configuracón del fichero */etc/hosts* donde se añadirá la ip y el nombre del nodo controlador. Por lo que nos aseguraremos que el fichero contenga la siguiente linea **192.168.1.11 slurmctld**. A pesar de tener un serividor DNS esto es necesario porque Slurm no tolera nombres DNS completos en su fichero de configuración *slurm.conf*, y tener el nombre del nodo controlador es vital para completar este fichero.

[2]APT es el gestor de paquetes de ubuntu https://help.ubuntu.com/kubuntu/desktopguide/es/apt-get.html

A.3. Nodos de cómputo en Google Cloud Platform (GCP)

La configuración de Slurm para los nodos de cómputo de GCP es igual que en la sección A.2.1 por lo que no se va a repetir esta configuración, sin embargo, hay diferencias en cuanto a la configuración de red que se verá ahora.

A.3.1. Configuración de red

Se comienza modificando el fichero */etc/hosts* incluyendo la siguiente linea **192.168.1.11 slurmctld** por las mismas razones que se ha visto en la sección A.2.2.

Se continuará configurando el servidor DNS en el fichero */etc/systemd/resolved.conf*. Será necesario añadir la siguiente linea **DNS=192.168.1.254**. Con esta linea conseguiremos que los nodos de GCP hagan uso del servidor DNS ubicado en el router de la universidad y de esta manera podrán resolver nombres de nodos.

A.4. Uso de Slurm

En esta sección se facilitarán diferentes comandos que han sido útiles en este trabajo, y se explicará su uso para que futuros alumnos les resulte más fácil su manejo. Esta sección se puede completar con la página de documentación de Slurm[3].

```
$ sinfo
```

Permite ver el estado del cluster, las particiones, los nodos y su estado.

```
$ scontrol show job 145
```

Permite ver detalles de la tarea con identificador 145, con este comando se puede ver que nodos se le han asignado, cual es su estado, si ha terminado o no, etc.

```
$ scontrol update NodeName=ubuntu22 State=DOWN
```

Permite cambiar el estado de un nodo. En este caso cambia el estado del nodod **ubuntu22** a **DOWN**, es decir, que establece que ese nodo está caído. Existe diferentes estados como *RESUME, IDLE, DRAINED...*, se recomienda consultar la documentación para más información.

```
$ sudo scontrol reconfigure
```

Notifica al controlador de Slurm que existe un cambio en el fichero *slurm.conf*, por lo que este demonio volverá a cargar la configuración del cluster.

```
$ srun -n 4 -N 2 hostname
```

Ejecuta el comando *hostname* en 2 nodo, pero este se ejecuta en 4 tareas diferentes. La opción *-n* indica que queremos 4 tareas y la opción *-N* indica que se exigen 2 nodos diferentes.

[3] https://slurm.schedmd.com/documentation.html

```
$ srun --partition=mix -N 10 hostname
```

Ejecuta en 10 nodos de la partición llamada *mix* el comando *hostname*.

```
$ srun --nodelist=ubuntu[21-24] -N 4 hostname
```

Ejecuta el comando *hostname* en 4 nodos, y además exigimos que los nodos a ser usados sean los del rango ubuntu21 al ubuntu24, por lo que Slurm está forzado a usar esos 4 nodos.

```
$ sbatch  simulacion_hybrid_5ramas.script
```

Sbatch es un comando que ejecuta un programa en bash. El contenido del programa es similar a las siguientes lineas:

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --output=output%j.txt
#SBATCH --error=error%j.txt
#SBATCH --job-name=gcp
#SBATCH --partition=gcp
#SBATCH --exclusive=user

srun /home/ubuntu/esimc 10ks3ramas 1000000
    '[10.0.0.27:20000,10.0.0.28:20000]'
```

A.5. Configuración de particiones

Crear particiones en Slurm es una tarea sencilla. Para crear particiones se debe acudir a la configuración del fichero *slurm.conf*. En este fichero se especifican los nodos que pertenecen al cluster y es ahi cuando debemos señalas las particiones deseadas.
En el código A.2 vemos como se especifican los nodos del cluster.
Se detallan tres grupos de nodos: ubuntu[2-8], ubuntu[21-40] y debian[1-10], cada uno con sus características hardware.
Acto seguido se especifican las particiones y vemos tres.

- Partición **tfg**: compuesto por los nodos ubuntu[2-8] y ubuntu[21-40].
- Partición **deb**: compuesto por los nodos debian[1-10].
- Partición **mix**: computes por ubuntu[2-8], ubuntu[21-40] y debian[1-10].

Código A.2: Especificación de particiones en Slurm

```
# COMPUTE NODES
NodeName=DEFAULT RealMemory=3000 TMpDisk=380
NodeName=DEFAULT State=UNKNOWN

NodeName=ubuntu[2-8] NodeAddr=192.168.1.[12-18] CPUs=4 Sockets=1
    CoresPerSocket=4 ThreadsPerCore=1
NodeName=ubuntu[21-40] NodeAddr=192.168.1.[21-40] CPUs=4 Sockets=1
     CoresPerSocket=4 ThreadsPerCore=1

NodeName=debian[1-10] RealMemory=1960 CPUs=2 Sockets=1
    CoresPerSocket=1 ThreadsPerCore=2
```

```
PartitionName=DEFAULT MaxTime=10 State=UP
PartitionName=tfg Nodes=ubuntu[2-8,21-40] Default=YES


PartitionName=deb Nodes=debian[1-10]
PartitionName=mix Nodes=ubuntu[2-8,21-40],debian[1-10]
```

Anexo B

Configuración del cloud privado virtual

En este anexo se verá como configurar diferentes recursos que han sido usados en Google Cloud Platform (GCP).

B.1. Nodos de Google Cloud Platform (GCP)

Para la configuración de los nodos de GCP se ha tenido que instanciar varias máquinas virtuales y realizar múltiples pruebas para llegar a conseguir la configuración ideal.

En primer lugar ha sido necesario crear una imagen base para ser usado en las máquinas virtuales instanciadas. Para conseguir la imagen el proceso ha consistido en crear una máquina virtual de cualquier tipo con SO *Ubuntu 20.04 LTS Minimal*. La elección de esta imagen es debido a que su tamaño es mínimo ya que no incluye paquetes innecesarios que suele traer Ubuntu consigo. Al ser una imagen mínima no incluye algunos paquetes necesarios (e.g. iputils-ping, traceroute...) por lo que se tendrá que añadir.
Tras elegir la imagen y configurar los paquetes necesarios se procede a configurar Slurm siguiendo los pasos de la sección A.3.
Cuando se haya configurado por completo la imagen se procederá a crear la imagen y publicarla en GCP para poder usarla más adelante con otras máquinas virtuales.

Otra de las configuraciones necesarias es el nombre DNS que tendrá esta máquina, esto es necesario para poder usarla en el servidor DNS y resolver el nombre sin tener que recordar la dirección IP de estas máquinas.

Por último, otro elemento importante es el programa de inicio que se lanza al creer la máquina virtual. El código de ese programa es el siguiente:

Código B.1: Programa de inicio que se ejecuta al crear la máquina virtual de GCP

```
#!/bin/bash

echo ${hostname} > /tmp/startup
echo "changing hostname" > /tmp/startup

domain="gcp.example.com"
echo "/etc/hosts before any change" > /tmp/startup
cat /etc/hosts > /tmp/startup
sed -e "2s/[a-z].*$/${hostname}\.gcp\.example\.com/" hosts

echo "/etc/hosts after sed" > /tmp/startup
cat /etc/hosts > /tmp/startup

echo "testing changes" > /tmp/startup
echo "Hostname -s: ${hostname -s}" > /tmp/startup
echo "Hostname -f: ${hostname -f}" > /tmp/startup
```

Este programa se encarga de coger el nombre del nodo y su dominio DNS y guardarlo en el fichero de */etc/hosts* para que de esta manera el nodo tenga configurado su nombre DNS.

B.2. Túnel VPN

Crear un túnel VPN es necesario para mantener una conexión directa entre el cluster on-premise y el cluster privado de GCP. Para ello, el primer paso será crear un túnel VPN Clásico siguiendo las indicaciones del proveedor cloud.

Para tener esta configuración en primer lugar es necesario una dirección IP externa para tener el extremo del entorno GCP al que conector el cluster on-premise. Tras tener la dirección IP se crea una puerta de enlace con esta dirección. A esta puerta de enlace será necesario asignarle reglas de reenvío necesarias para el trafico de paquetes a través del Túnel IPSec. Las reglas necesarias son las siguientes:

- Reenvío de paquetes por puerto 4500 a través del protocolo UDP.
- Reenvío de paquetes por puerto 500 a través del protocolo UDP.
- Reenvío de paquetes a través del protocolo ESP.

Para finalizar se necesita configurar una ruta para redirigir el tráfico que llegue de la red del cluster on-premise a la subred existente en nuestro cluster privado. Una vez configurado todo debemos tener una pantalla similar a la de la imagen B.1 y debemos ver un verificado al lado del nombre del túnel si la conexión es correcta entre ambos extremos.

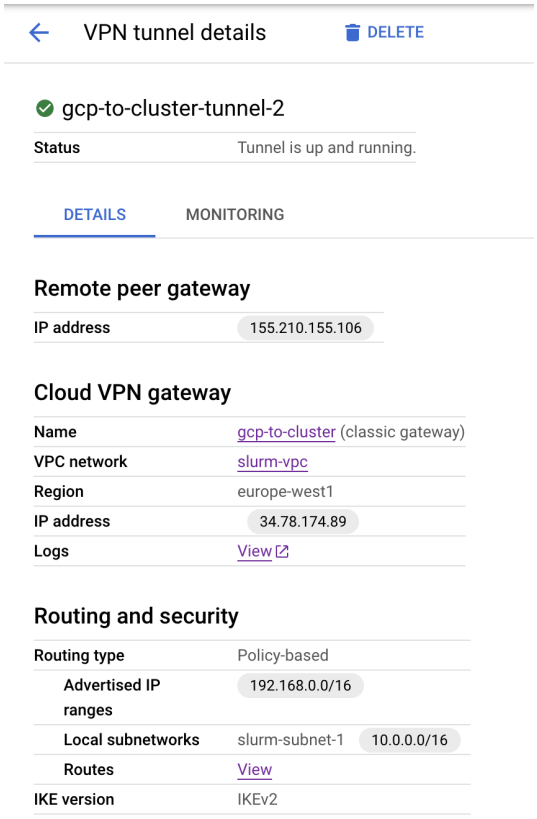


Figura B.1: Configuración del Túnel VPN en GCP.

B.3. DNS

Tras tener configurado el servidor DNS en el router como se indica en la sección D.3 es necesario configurar una zona DNS en la VPC. Con esta zona de reenvío se mantiene el registros de nombres y direcciones DNS de los nodos del proveedor.
Un último elemento necesario para configurar el DNS en la VPC es una regla que permita

tráfico entrante a la zona, de esta manera se permite consultas DNS desde el cluster on-premise. Tras terminar todo, debemos observar una zona DNS como la de la figura B.2.

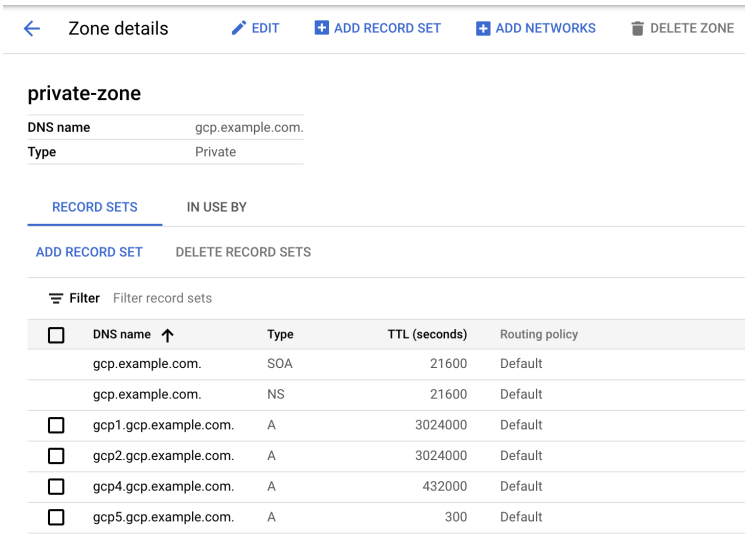


Figura B.2: Configuración de la zona DNS en GCP.

B.4. Firewall

El firewall es necesario para fortalecer la seguridad del sistema como se ha visto en la sección 4.3.1. Se configurarán nueve reglas que se listarán y explicarán a continuación:

- Permitir tráfico saliente de los nodos de cómputo al nodo controlador a través del puerto 6817 por los protocolos TCP y UDP hacia el resto de nodos de cómputo y los nodos del cluster on-premise.
- Permitir tráfico entrante del nodo controlador y los nodos del cluster on-premise a los nodos de cómputo a través del puerto 6818 por los protocolos TCP y UDP.
- Permitir peticiones DNS entrantes del cluster on-premise al cluster de GCP por el puerto 53 a través del protocolo TCP y UDP.
- Permitir tráfico ICMP entrante desde cualquier dirección.
- Permitir conexiones SSH a través del puerto 22 y 3389 por el protocolo TCP desde *'35.235.240.0/20'* que es la dirección del cliente de interfaz de GCP.
- Permitir conexiones SSH a través del puerto 22 por el protocolo TCP desde la red del cluster on-premise, el router y los nodos de cómputo de GCP.
- Permitir conexiones SSH por puerto 22 y protocolo TCP desde cualquier dirección a aquellas máquinas virtuales de GCP que lleven el *tag* **ssh-from-all**.
- Permitir tráfico TCP entrante desde los nodos de cómputo y los nodos del cluster on-premise para permitir el intercambio de mensajes que necesita el simulador implementado en Rust
- Permitir tráfico TCP saliente desde los nodos de cómputo y los nodos del cluster on-premise para permitir el intercambio de mensajes que necesita el simulador implementado en Rust

Con estas reglas configuradas en la VPC tendremos la configuración de la red necesaria para poder tener tráfico entre los dos cluster con seguridad.

Name	Type	Targets	Filters	Protocols / ports	Action	Priority	Network
allow-slurm-outbound	Egress	Apply to all	IP ranges: 192.168.1.0/24, 10.0.0.0/24	tcp:6817 udp:6817	Allow	1000	slurm-vpc
rust-tcp-out	Egress	Apply to all	IP ranges: 192.168.1.0/24, 10.0.0.0/24	tcp	Allow	1000	slurm-vpc
allow-dns	Ingress	Apply to all	IP ranges: 192.168.1.0/24	tcp:53 udp:53	Allow	1000	slurm-vpc
allow-icmp	Ingress	Apply to all	IP ranges: 0.0.0.0/0	icmp	Allow	1000	slurm-vpc
allow-ingress-from-iap	Ingress	Apply to all	IP ranges: 35.235.240.0/20	tcp:22, 3389	Allow	1000	slurm-vpc
allow-slurm	Ingress	Apply to all	IP ranges: 192.168.1.0/24, 10.0.0.0/24	tcp:6818 udp:6818	Allow	1000	slurm-vpc
allow-tcp	Ingress	Apply to all	IP ranges: 192.168.1.0/24, 192.168.0.0/24, 155.210.155.0/24, 85.251.75.0/	tcp:22	Allow	1000	slurm-vpc
allow-tcp-ssh	Ingress	ssh-from-all	IP ranges: 0.0.0.0/0	tcp:22	Allow	1000	slurm-vpc
rust-2000	Ingress	Apply to all	IP ranges: 192.168.1.0/24, 10.0.0.0/24	tcp	Allow	1000	slurm-vpc
slurm-vpc-allow-http	Ingress	http-server	IP ranges: 0.0.0.0/0	tcp:80	Allow	1000	slurm-vpc
slurm-vpc-allow-https	Ingress	https-server	IP ranges: 0.0.0.0/0	tcp:443	Allow	1000	slurm-vpc

Figura B.3: Listado de reglas de firewall en GCP.

Anexo C

Automatización

C.1. Instalación y configuración de Slurm

Automatizar la instalación y configuración de Slurm es muy útil debido a que se tienen múltiples nodos que requieren de este software. Para esta automatización se ha usado Ansible, un framework que permite automatizar tareas que se podría realizar de manera manual, como por ejemplo configuración e instalación de software y, creación y manipulación de ficheros. Para llevar a cabo el desarrollo del role de Ansible se ha usado el software de versiones GitHub[1], todos los cambios se han publicado y se han mantenido documentados bajo la organización del grupo de investigación dentro del repositorio *ansible-slurm*[2].

C.1.1. Funcionamiento de Ansible

En primer lugar, se va a explicar como funciona Ansible.

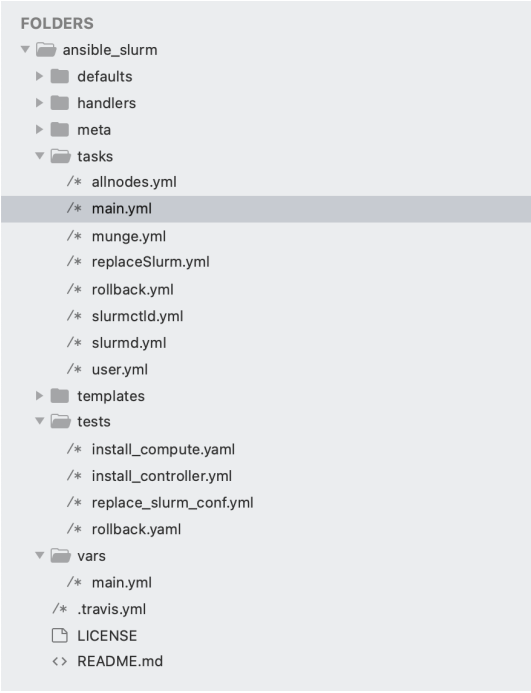


Figura C.1: Estructura de ficheros de Ansible

En la figura C.1 se observa la estructura de ficheros de Ansible es clave entender esto para comprender como funciona este software.

En primer lugar, se tiene el directorio *defaults* donde se definen variables que se usarán en este role. La forma de declararlo es usando la sintaxis de *yml* como se observa en C.1.

[1] https://github.com/
[2] https://github.com/simbots-swarm/ansible_slurm

Código C.1: Declaración de variables en el fichero de *defaults* para un role de Ansible

```
uninstall_pacakges:
  - munge
  - slurm-client
  - slurm-wlm-basic-plugins
  - slurm-wlm-doc
  - slurm-wlm
  - slurmctld
  - slurmd
  - libmunge2


#directorios
slurmctld_state_dir: '/var/spool/slurm-llnl/slurmctld'
slurmctld_log_dir: '/var/log/slurm-llnl/'
slurmctld_pid_file: '/run/slurmctld.pid'
```

Continuamos con el directorio de *handlers* que contiene tareas que serán lanzadas cuando se le notifique. Un caso de uso es cuando falla la instalación de un paquete se puede lanzar un *handler* para eliminar los ficheros que ha dejado este paquete en la máquina.

El siguiente directorio de la figura C.1 que cabe destacar es el de *tasks*. Este directorio es muy importante porque contiene las tareas que se lanzan en las ejecuciones. Estas tareas se agrupan en ficheros manteniendo una cierta lógica. Por ejemplo, el fichero de *munge.yml* contiene tareas relacionadas con la configuración e instalación de Munge.

Continuamos con el directorio de *templates* que contiene las plantillas de los ficheros en formato Jinja, estas plantillas son usadas, por ejemplo, para copiar ficheros en la máquina.

Por último, se dispone del directorio de *tests*, aquí van los *Playbook* de Ansible que son creados para ejecutar las tareas. A continuación se ve un ejemplo de *Playbook* que lanza la configuración del nodo controlador.

Código C.2: Playbook de Ansible para lanzar la configuración del nodo controlador

```
---

- name: Installing controller
  hosts:
    - 192.168.56.13
  vars:
    slurm_roles: ['controller']
    slurm_munge_key_dest: './'
  roles:
    - ansible_slurm
```

En las lineas del código C.2 se lleva a cabo la declaración de un *Playbook*. En primer lugar, se le da un nombre, luego se especifican los *hosts*, es decir, las máquinas donde aplicar estos cambios. Se continua con las variables, en este caso se le pasan dos. La primera variable, llamada *slurm_rules* tiene valor *controller* para indicar que la instalación es en un nodo controlador y de esta manera saber que tareas aplicar o no. La segunda variable, *slurm_munge_key_dest* que tiene valor `./`, especifica donde ubicar el archivo *munge.key* tras terminar la configuración de este nodo. Por último, se tiene el atributo *roles* y aquí es donde se especifica el role que se usará, en nuestro caso *ansible_slurm* que es el role que se ha programado.

En la figura C.2 se observan las lineas principales que ejecutan la instalación de Slurm. Las primeras 10 lineas contienen declaraciones que se verá en la sección C.2. Continua con la configuración de un *rollback*, es decir, un restablecimiento que se llevará a cabo en el caso de que el *Playbook* incluya la palabra *'rollback'*. Se observa que se laza primero la ejecución de un fichero llamado *user.yml* que es el encargado de la creación del usuario. A continuación

```
- name: Replace slurm.conf file
  include_tasks: replaceSlurm.yml

- name: Stopping slurm_roles
  fail:
    msg: "THIS FAIL IS INTENTIONED, ROLE STOPPED AFTER REPLACING FILE"
  when: "replaceDone is defined"

- name: Rolling Back
  include_tasks: rollback.yml
  when: rollback is defined

- name: Stopping slurm_roles
  fail:
    msg: "THIS FAIL IS INTENTIONED, ROLE STOPPED BECAUSE OF ROLLBACK"
  when: "rollbackDone is defined"

- name: Include user creation task
  include_tasks: user.yml

- name: Include slurmctld node configuration
  include_tasks: slurmctld.yml
  when: "'controller' in slurm_roles"

- name: Include slurmd node configuration
  include_tasks: slurmd.yml
  when: "'compute' in slurm_roles"

- name: Include common configuration for all nodes
  include_tasks: allnodes.yml

- name: Check slurmctld is enabled and running
  become: yes
  service:
    name: "{{ slurmctld_service }}"
    enabled: yes
    state: started
  when: "'controller' in slurm_roles"

- name: Check slurmd is enabled and running
  become: yes
  service:
    name: "{{ slurmd_service }}"
    enabled: yes
    state: started
  when: "'compute' in slurm_roles"

- name: Change pid file permission
  become: true
  file:
    path: "{{ slurmctld_pid_file }}"
    owner: "slurm"
    group: "root"
    mode: 0764
    state: file
  when: "'controller' in slurm_roles"

- name: Change pid file permission
  become: true
  file:
    path: "{{ slurmd_pid_file }}"
    owner: "slurm"
    group: "root"
    mode: 0764
    state: file
  when: "'compute' in slurm_roles"
```

Figura C.2: Fichero principal del role de Ansible

se ejecutará el fichero *slurmctld.yml* o *slurmd.yml* según si estamos trabajando con el nodo controlador o el nodo de cómputo. Finalmente se lanza el fichero *allnodes.yml* que contiene configuraciones comunes sin diferenciar el nodo.
En las siguientes lineas se verán ejemplos del contenido de algunos de los ficheros mencionados anteriormente.

Código C.3: Creación de usuario y grupo usando Ansible.

```
---
- name: Create slurm  grooup
  become: true
  group:
    gid: "1110"
    name: "slurm"
    state: "present"

- name: Create slurm user
  become: true
  user:
    name: "slurm"
    uid: "1110"
    group: "slurm"
    home: "/var/lib/slurm"
    shell: "/bin/bash"
```

En el ejemplo C.3 de código muestra como se crearía un grupo llamado Slurm con *gid* 1110, y un usuario llamado Slurm con *uid* 1110 haciendo uso de las declaraciones en yml de Ansible.

Código C.4: Ejemplo de instalación de un paquete y creación de un directorio

```
---

- name: Install Slurm controller package
  become: yes
  package:
    name: "{{ slurm_packages.slurmd }}"
    cache_valid_time: 3600
    force_apt_get: yes
    state: "{{ 'latest' if slurm_upgrade == 'yes' else 'present'
       }}"

- name: Create slurmd spool directory
  become: yes
  file:
    path: "{{ slurmd_spool_dir }}"
    owner: slurm
    group: root
    mode: 0755
    state: directory
```

En el código C.4 se observa unas declaración que instalan los paquetes que indica la variable *slurm_packages.slurmd* y acto seguido crea un directorio para el usuario *Slurm* con permisos 0755 ubicado en el valor de la variable *slurmd_spool_dir*.

C.1.2. Como lanzar la instalación automática

En primer lugar, es necesario tener configurado el fichero **oculto** con nombre *.ansible.cfg* ubicado en el *home* del usuario con el siguiente contenido.

Código C.5: Fichero de configuración del perfil de Ansible

```
[defaults]
inventory = ~/ansible/hosts
roles_path = ~/ansible
remote_user = ubuntu
```

En las lineas del código C.5 se configuran 3 elementos principales.

- **inventory:** Ubicación del fichero de *hosts*(nodos con los que vamos a trabajar).
- **roles_path:** Ubicación de los roles de Ansible.
- **remote_user:** Usuario remoto al que nos conectamos con Ansible.

Lo segundo que se tiene que tener configurado es el fichero de *hosts* que tendrá la forma que se observa en el código C.6

Código C.6: Fichero de hosts para Ansible

```
all:
  hosts:
    192.168.1.23
  children:
    controller:
      hosts:
        192.168.1.11
    computePrimeraPila:
      hosts:
        192.168.1.[12:18]
```

Se puede definir los *hosts* de diferentes maneras, la primera es usando la declaración **hosts** y declarar nodos unitarios. La segunda, es agruparlos con un nombre, por ejemplo, se observa el grupo *computePrimeraPila* y contiene los nodos cuya IP va desde 192.168.1.12 hata la 192.168.1.18.

Para continuar con la instalación automática, lo siguiente que será necesario es definir el *Playbook*. En el código C.2 se ha visto como definirlo para el nodo controlador, por lo que ahora se verá para el nodo de cómputo y un ejemplo de como aplicar un *rollback*.

Código C.7: Playbook de Ansible para lanzar la configuración del nodo de cómputo

```
---

- name: Installing compute
  hosts:
    - 192.168.1.14
    - 192.168.1.15
  vars:
    slurm_roles: ['compute']
    slurm_munge_key: './path/to/munge.key'
  roles:
    - ansible_slurm
```

En las lineas de C.7 primero se le da un nombre y se especifica los nodos donde aplicar los cambios. A continuación, se declaran las variables para especificar que es un nodo de cómputo y se le indica donde se encuentra el fichero *munge.key*. Por último, se recalca que role usar.

Código C.8: Playbook de Ansible para lanzar un *rollback* en un nodo.

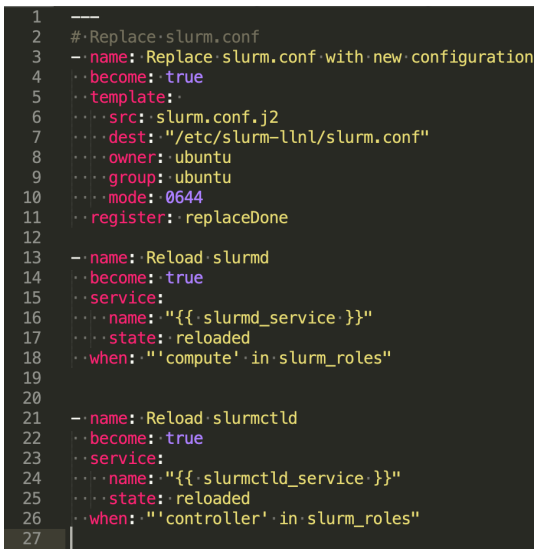
```
---

- name: Uninstalling slurm
  hosts:
    - 192.168.1.13
  vars:
    rollback: yes
  roles:
    - ansible_slurm
```

De nuevo se le da un nombre y un nodo, se le indica con la variable *rollback* que sí se quiere llevarlo a cabo y se marca que role se usa.

C.2. Sustitución del fichero de configuración de Slurm

Se necesita automatizar esta tarea ya que sino exige de un proceso largo para llevarlo a cabo. El proceso consiste en modificar el fichero, enviarlo a todos los nodos del cluster, moverlo al directorio correspondiente y reiniciar los demonios correspondientes. Para automatizar todo esto se tiene una simple tarea de Ansible (ver figura C.3). En esta tarea se usa plantillas de Jinja 2 para reemplazar valores en el fichero antes de copiarlo en el nodo destino. En las lineas del código C.9 se ve como se pasan diferentes variables que serán los que se sustituyan en el fichero.



```
---
#·Replace·slurm.conf
-·name:·Replace·slurm.conf·with·new·configuration
··become:·true
··template:·
····src:·slurm.conf.j2
····dest:·"/etc/slurm-llnl/slurm.conf"
····owner:·ubuntu
····group:·ubuntu
····mode:·0644
··register:·replaceDone

-·name:·Reload·slurmd
··become:·true
··service:
····name:·"{{·slurmd_service·}}"
····state:·reloaded
··when:·"'compute'·in·slurm_roles"


-·name:·Reload·slurmctld
··become:·true
··service:
····name:·"{{·slurmctld_service·}}"
····state:·reloaded
··when:·"'controller'·in·slurm_roles"
```

Figura C.3: Código de Ansible para reemplazar el fichero de configuración de Slurm

El Playbook de Ansible correspondiente para llevar a cabo el proceso de reemplazo es el del código C.9

Código C.9: Reemplazar fichero slurm.conf con un Playbook

```
- name: Replacing slurm.conf from templates
  hosts:
    - 192.168.1.13
  vars:
    slurm_roles: ['compute']
    gcp_nodes_dns_name: 'gcp0.gcp.example.com,gcp1.gcp.example.com
        ,gcp2.gcp.example.com,gcp3.gcp.example.com'
    gcp_node_cpu: '2'
    gcp_nodes_sockets: '1'
    gcp_node_cores_per_sockets: '1'
    gcp_nodes_threads: '2'
  roles:
    - ansible_slurm
```

C.3. Aprovisionamiento automático de recursos en cloud con Terraform

Como se ha visto en el anexo B, la configuración de la VPC resulta larga, tediosa y complicada en algunos sentidos. Una vez se tiene completada la configuración no es necesario volver a llevarla a cabo a mano y esto se puede automatizar para evitar problemas haciendo uso de Terraform.
Este módulo de Terraform se ha desarrollado y se ha mantenido un control de versiones haciendo uso de GitHub. Todo el código necesario se encuentre en el repositorio bajo la organización del grupo de investigación [3].

C.3.1. Funcionamiento de Terraform

Terraform tiene una operativa muy sencilla, la automatización que ofrece no tiene complejidad alguna. Primero se debe acudir a la web de Terraform[4] donde se detallan todos los recursos de la plataforma GCP que podemos usar. En segundo lugar, se documenta acerca de cada recurso que se desea instanciar y se procede a parametrizar este recurso. En los ejemplos de esta sección se verá que muchos de los recursos hacen uso de variables, esto se

[3] https://github.com/simbots-swarm/terraform-google-slurm
[4] https://registry.terraform.io/providers/hashicorp/google/latest/docs

debe a que se ha parametrizado para tener ubicado en un único fichero las variables que se usan en cada módulo.

Se debe comprender la estructura de ficheros para poder llevar a cabo su desarrollo. Esta estructura comprende de 4 ficheros:

- **main.tf** Aquí se encuentran los recursos a desplegar
- **outputs.tf** Este fichero contiene valores de salida que se devuelven al lanzar el módulo
- **variables.tf** Este fichero contiene variables que se usan en *main.tf*. La razón de uso de un *varibales.tf* es debido a que es más sencillo parametrizar desde un fichero único que ir modificando uso por uso de cada variable en todo el *main*.
- **versions.tf** Este fichero contiene información sobre la versión del proveedor y la versión de Terraform.

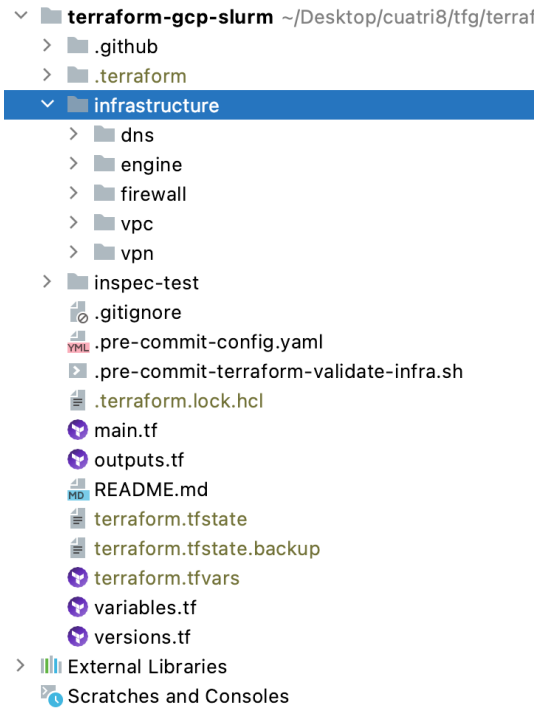


Figura C.4: Estructura de directorios del módulo de Terraform

En la figura C.4 se observa la estructura de directorios de Terraform para este módulo. Se dispone del directorio principal llamado **terraform-google-slurm**. Este nombre sigue el convenio marcado por el software que consiste en *terraform-proveedor-nombreDeLoQueAutomatizamos*.
Bajo la raiz se encuentra el directorio **infrastructure** que contiene módulos con recursos de GCP. Se ha hecho esta abstracción ya que estos módulos pueden ser usados independientemente por lo que tiene sentido este desacople.
Finalmente se tiene bajo la raíz los 4 tipos de ficheros mencionado unas lineas más arriba. En este caso el fichero *main.tf* contiene la instanciación de cada uno de los módulos que se encuentran bajo *infrastructure* por lo que es el encargado de crear todo el sistema a desplegar.
En el código C.10 se puede ver dos ejemplos de como el fichero *main.tf* principal llama a los módulos pasándoles unas variables para lanzar su instanciación.

Código C.10: Fichero *main.tf* encargo de desplegar todo el sistema

```
module "vpc" {
  source          = "./infrastructure/vpc"
  network_name    = var.network_name
  subnetwork_ip   = var.subnetwork_ip
```

```
  subnetwork_name = var.subnetwork_name
}


module "firewall_rules" {
  source = "./infrastructure/firewall"

  vpc_network           = module.vpc.vpc_network_name
  on_prem_ip_cidr       = var.on_prem_nodes
  dawson_network        = var.dawson_network
  raspberry_pi_network  = var.raspberry_pi_network
  gcp_subnet_ip_cidr    = module.vpc.vpc_subnetworks_ip_cidr

}
```

C.3.2. Creación de recursos para la automatización

VPC

Automatizar la Virtual Private Cloud (VPC) requiere la declaración de dos recursos en Terraform, la primera es la red que se usará y una subred donde se encontrarán los recursos. En el código C.11 se observa la declaración de estos recursos.

Código C.11: Creación de la red y subred del VPC

```
resource "google_compute_network" "vpc_network" {
  name                    = var.network_name
  auto_create_subnetworks = false
}


resource "google_compute_subnetwork" "vpc_subnet" {

  name          = var.subnetwork_name
  ip_cidr_range = var.subnetwork_ip
  network       = google_compute_network.vpc_network.name
  region        = var.region
}
```

Firewall

Una vez se tienen las redes y subredes creadas es el momento de crear las reglas de firewall para brindar la mayor seguridad posible. Se procede a automatizar la creación de las nueve reglas de firewall detalladas en la sección B.4. A continuación se muestra la declaración de dos de ellas para observar un ejemplo de declaración.

Código C.12: Regla de firwall para permitir tráfico ICMP entrante.

```
resource "google_compute_firewall" "allow_icmp" {
  name          = "allow-icmp"
  network       = var.vpc_network
  description   = "Allow ICMP traffict to gcp nodes from any IP"
  source_ranges = ["0.0.0.0/0"]
  priority      = "1000"

  # no targets applied because we want this on each node in the
      vpc
  direction = "INGRESS"

  allow {
    protocol = "icmp"
```

```
  }
}
```

Código C.13: Regla de firewall para permitir tráfico saliente por puerto 6817 para el demonio slurmd

```
resource "google_compute_firewall" "allow_slurm_outbound" {
  name               = "allow-slurm-outbound"
  network            = var.vpc_network
  description        = "Allow outgoing traffic from slurmd to
      slurmctld through 6817 port"
  destination_ranges = [var.on_prem_ip_cidr, var.
      gcp_subnet_ip_cidr]
  priority           = "1000"
  # no targets applied because we want this on each node in the
      vpc
  direction = "EGRESS"

  allow {
    protocol = "tcp"
    ports    = ["6817"]
  }

  allow {
    protocol = "udp"
    ports    = ["6817"]
  }
}
```

VPN

La VPN sigue una dinámica parecida. Se crean los recursos necesario de la sección B.2 como se ve en el código C.14

Código C.14: Declaración de una IP y creación de un túnel

```
resource "google_compute_address" "vpn_static_ip" {
  name   = "${var.project_name}-vpn-static-ip"
  region = var.region
}
...
resource "google_compute_vpn_tunnel" "tunnel1" {
  name                    = "${var.project_name}-tunnel"
  peer_ip                 = var.on_prem_tunnel_peer_ip
  shared_secret           = var.pke_secret
  region                  = var.region
  local_traffic_selector = [var.vpc_tunnel_subnetwork]

  target_vpn_gateway = google_compute_vpn_gateway.target_gateway.
      id

  depends_on = [
    google_compute_address.vpn_static_ip,
    google_compute_forwarding_rule.fr_esp,
    google_compute_forwarding_rule.fr_udp500,
    google_compute_forwarding_rule.fr_udp4500,
  ]
}
```

Nodos de cómputo en GCP

Crear los nodos de manera manual puede ser fácil cuando se tiene uno o dos nodos, pero cuanto se pasa a tener más es un proceso largo y pesado. Automatizar esta tarea con terrafomr es muy simple.

Código C.15: Creación de 4 nodos de GCP con Terraform

```
resource "google_compute_instance" "default" {
  count        = length(var.node_names)
  name         = var.node_names[count.index]
  machine_type = var.machine_type
  zone         = var.zone


  boot_disk {
    initialize_params {
      type  = "pd-ssd"
      image = var.instance_image_source
    }
  }

  network_interface {
    network    = var.vpc_network
    subnetwork = var.vpc_subnetwork

    access_config {
      # Ephemeral IP
      network_tier = "STANDARD"
    }
  }

  hostname = "${var.node_names[count.index]}.${var.gcp_dns_domain
     }"

  service_account {
    # Google recommends custom service accounts that have cloud-
        platform scope and permissions granted via IAM Roles.
    email  = data.google_service_account.default.email
    scopes = ["cloud-platform"]
  }

  metadata_startup_script = file("${path.module}/startup.sh")
}
```

En el código C.15 se ve como con una declaración de un recurso se pueden instanciar 4 nodos de GCP. Esto se debe a que se hace uso de un variable **count** cuyo valor es la longitud de un *array* que contiene los nombres de los nodos, por lo que si declaramos 20 nodos, este recurso se creará 20 veces.

DNS

Para el DNS se necesita una zona DNS, una política que permita el tráfico entrante y guardar los registros de pares nombre e IP en la zona DNS. Usando Terraform y haciendo uso de la variable count es sencillo, como se ve en el código C.16.

Código C.16: Creación de la zona DNS y creación de los registros en la zona DNS

```
resource "google_dns_managed_zone" "private_zone" {
  name        = "${var.project_name}-private-zone"
  dns_name    = "${var.gcp_dns_domain}."
  description = "DNS private zone por GCP nodes"

  visibility = "private"
```

```
  private_visibility_config {
    networks {
      network_url = var.vpc_network_id
    }
  }
}

resource "google_dns_record_set" "resource_recordset" {
  count        = length(var.node_names)
  managed_zone = google_dns_managed_zone.private_zone.name
  name         = "${data.google_compute_instance.node[count.index
      ].name}.${var.gcp_dns_domain}."
  type         = "A"
  rrdatas      = [data.google_compute_instance.node[count.index].
      network_interface.0.network_ip]
  ttl          = 86400
}


resource "google_dns_policy" "inbound_policy" {
  name                      = "${var.project_name}-inbound-policy"
  enable_inbound_forwarding = true
  enable_logging            = false

  networks {
    network_url = var.vpc_network_id

  }
}
```

Anexo D

Administración y configuración del Router del Cloud Híbrido

D.1. Configuración del Router del Cloud Híbrido

Para comenzar hay que habilitar la función de router, para ello es necesario ejecutar el comando **$ sysctl –w net.ipv4.ip_forward=1**, con esto permitir el reeenvío de paquetes.

Se continuará con la creación de una red NAT a través de *iptables.* Para ello es suficiente con establecer una regla de tipo *POSTROUTING* que reenvíe por la interfaz de red deseada los paquetes dirigidos a la subred de los nodos del cluster on-premise, en este caso, 192.168.0.0/24.
Cabe destacar que para que esta funcionalidad se desempeñe correctamente, es necesario disponer de una IP pública en el router y que esta esté expuesta hacia el exterior.

Por último, se ha establecido la asignación de direcciones IP automática en función de su dirección MAC a los nodos del cluster on-premise. Para ello se han creado entradas en el fichero */etc/dhcp/dhcpd.conf* como las del código D.1. Estas entradas asignan a un *host* una dirección fija en base a su dirección MAC.

Código D.1: Asignación automática de direcciones IP según la MAC

```
host r42 {
  hardware ethernet DC:A6:32:23:4D:25;
  fixed-address 192.168.1.12;
}
host r43 {
  hardware ethernet DC:A6:32:23:A1:9D;
  fixed-address 192.168.1.13;
}
```

D.2. Red VPN

Configurar un túnel VPN[12] es necesario para poder llevar a cabo una conexión por Túnel IPSec desde el Cloud Privado del cluster on-premise al Cloud Público del cluster de GCP. Para llevar a cabo esta tarea se ha hecho uso del software Strongswan[1].

Tras instalar el software a través del comando **sudo apt install strongswan strongswan-pki** procedemos a configurar en primer lugar el fichero de secretos. Para ello accedemos al fichero */var/lib/strongswan/ipsec.secrets.inc* y lo rellenamos con el siguiente contenido: **@IP_GATEWAY_GCP : PSK "nuestraClaveSecreta"**.
Una vez configurado el secreto pasamos a configurar el fichero que establece la conexión. Se necesita rellenar el fichero */var/lib/strongswan/ipsec.conf.inc* con el contenido de la figura D.1.

[1] https://www.strongswan.org/

```
conn %default
    ikelifetime=600m # 36,000 s
    keylife=180m # 10,800 s
    rekeymargin=3m
    keyingtries=3
    keyexchange=ikev2
    mobike=no
    ike=chacha20poly1305-sha512-curve25519-prfsha512,aes256gcm16-sha384-prfsha384-ecp384,aes256-sha1-modp1024,aes128-sha1-modp1024,3des-sha1-modp1024!
    esp=chacha20poly1305-sha512,aes256gcm16-ecp384,aes256-sha256,aes256-sha1,3des-sha1!
    authby=psk

conn net-net
    leftcert=/etc/ipsec.d/certs/server-cert.pem
    left=155.210.155.106 # In case of NAT set to internal IP, e.x. 10.164.0.6
    leftid=155.210.155.106
    leftsubnet=192.168.0.0/16
    leftauth=psk
    right=34.78.174.89
    rightid=34.78.174.89
    rightsubnet=10.0.0.0/16
    rightauth=psk
    type=tunnel
    # auto=add - means strongSwan won't try to initiate it
    # auto=start - means strongSwan will try to establish connection as well
    # Note that Google Cloud will also try to initiate the connection
    auto=start
    # dpdaction=restart - means strongSwan will try to reconnect if Dead Peer Detection spots
    #                 a problem. Change to 'clear' if needed
    dpdaction=restart
```

Figura D.1: Fichero de configuración del túnel VPN y del Túnel IPSec.

Entre otras cosas, en la configuración que se ve en la figura D.1 se especifican los extremos del túnel, los tipos de autenticación que aceptamos, subredes a las que les brindamos este servicio, etc.
Cuando se ha completado esta configuración se reinicia el demonio con el comando **sudo service strongswan restart** y se comprueba que no hay errores en la salida de **sudo service strongswan status**, si todo va bien veremos algo como la figura D.2

```
● strongswan-starter.service - strongSwan IPsec IKEv1/IKEv2 daemon using ipsec.conf
     Loaded: loaded (/lib/systemd/system/strongswan-starter.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2021-08-18 16:33:19 CEST; 4 weeks 1 days ago
   Main PID: 17204 (starter)
      Tasks: 18 (limit: 38315)
     Memory: 7.1M
     CGroup: /system.slice/strongswan-starter.service
             ├─17204 /usr/lib/ipsec/starter --daemon charon --nofork
             └─17221 /usr/lib/ipsec/charon

Sep 17 09:58:45 dawson ipsec[17221]: 12[ENC] generating INFORMATIONAL response 450 [ ]
Sep 17 09:58:45 dawson charon[17221]: 13[NET] received packet: from 34.78.174.89[500] to 155.210.155.106[500] (57 bytes)
Sep 17 09:58:45 dawson ipsec[17221]: 12[NET] sending packet: from 155.210.155.106[500] to 34.78.174.89[500] (57 bytes)
Sep 17 09:58:45 dawson charon[17221]: 13[ENC] parsed INFORMATIONAL request 451 [ ]
Sep 17 09:58:45 dawson charon[17221]: 13[ENC] generating INFORMATIONAL response 451 [ ]
Sep 17 09:58:45 dawson charon[17221]: 13[NET] sending packet: from 155.210.155.106[500] to 34.78.174.89[500] (57 bytes)
Sep 17 09:59:15 dawson charon[17221]: 11[NET] received packet: from 34.78.174.89[500] to 155.210.155.106[500] (57 bytes)
Sep 17 09:59:15 dawson charon[17221]: 11[ENC] parsed INFORMATIONAL request 452 [ ]
Sep 17 09:59:15 dawson charon[17221]: 11[ENC] generating INFORMATIONAL response 452 [ ]
Sep 17 09:59:15 dawson charon[17221]: 11[NET] sending packet: from 155.210.155.106[500] to 34.78.174.89[500] (57 bytes)
hayk@dawson:~$
```

Figura D.2: Estado del demonio de strongswan que muestra que hay tráfico en ambos sentidos.

D.3. Servidor DNS

Para la configuración del servidor DNS[5] se ha usado **Bind**[2] un software que permite la configuración de un servidor DNS de manera sencilla en Ubuntu. Este servidor se configura en el router, para ver la configuración necesaria se detallan las siguientes lineas.

En primer lugar debemos instalar el software, para ello aplicamos el comando **sudo apt-get install bind9 bind9utils bind9-doc**. Una vez disponemos del software configuramos la herramienta como en la figura D.3.
En esta configuración se observa que se define dos *forwarders*. Estos son servidores DNS a los que redirigimos peticiones, el primer, con dirección 10.0.0.17, es el servidor del cluster GCP y, el segundo, es el servidor DNS de Google. Por otro lado, al final del fichero definimos unas opciones necesarias como es permitir la recursividad y consultas desde cualquier dirección.

[2]https://www.isc.org/bind/

```
options {
        directory "/var/cache/bind";

        // If there is a firewall between you and nameservers you want
        // to talk to, you may need to fix the firewall to allow multiple
        // ports to talk.  See http://www.kb.cert.org/vuls/id/800113

        // If your ISP provided one or more IP addresses for stable
        // nameservers, you probably want to use them as forwarders.
        // Uncomment the following block, and insert the addresses replacing
        // the all-0's placeholder.

         forwarders {
                10.0.0.25;
                8.8.8.8;
         };

        //========================================================================
        // If BIND logs error messages about the root key being expired,
        // you will need to update your keys.  See https://www.isc.org/bind-keys
        //========================================================================
        dnssec-validation no;
        dnssec-enable no;

        # config by Hayk
        recursion yes;
        allow-query { any; };
        allow-recursion { any; };
        # done

        listen-on-v6 { any; };
};
```

Figura D.3: Configuración de las opciones de bind9.

Una vez configurada la herramienta, pasamos a configurar una zona de reenvío DNS. En este caso vamos añadir una zona que es a la que damos soporte y es la que aparece al final del fichero de la imagen D.4. La zona se llama **gcp.example.com** y es de tipo *forward*, ya que reenvía las peticiones a la zona DNS del cluster GCP (ver sección B.3). Esta zona es necesaria para redirigir las peticiones DNS con este dominio a la zona DNS de GCP.

```
// prime the server with knowledge of the root servers
zone "." {
        type hint;
        file "/usr/share/dns/root.hints";
};

// be authoritative for the localhost forward and reverse zones, and for
// broadcast zones as per RFC 1912

zone "localhost" {
        type master;
        file "/etc/bind/db.local";
};

zone "127.in-addr.arpa" {
        type master;
        file "/etc/bind/db.127";
};

zone "0.in-addr.arpa" {
        type master;
        file "/etc/bind/db.0";
};

zone "255.in-addr.arpa" {
        type master;
        file "/etc/bind/db.255";
};

zone "gcp.example.com"  {
        type forward;
        forwarders {
                10.0.0.25;
        };
};
```

Figura D.4: Definición de las zonas DNS de reenvio.

Continuamos con la configuración de la zona DNS a la que si que damos soporte de resolución

de nombres (ver figura D.5). La zona tiene el nombre de **corp.example.com** y simplemente le especificamos donde se encuentra el fichero que contiene los registros de nombres e IPs. En las figuras D.6 y D.7 se pueden ver los detalles de personalización de los registros DNS.

```
//
// Do any local configuration here
//

// Consider adding the 1918 zones here, if they are not used in your
// organization
//include "/etc/bind/zones.rfc1918";

zone "corp.example.com" {
        type master;
        file "/etc/bind/corp.example.com";
        allow-update { none; };
};
zone "1.168.192.in-addr.arpa" {
        type master;
        file "/etc/bind/corp.example.com.reverse";
        allow-update { none; };
};
"/etc/bind/named.conf.local" 18L, 387C
```

Figura D.5: Definición de las zonas DNS a las que damos soporte.

```
;
; BIND data file for corp.example.com
;
$TTL    604800
@       IN      SOA     dns1.corp.example.com.  root.corp.example.com (
                              3         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL

;Name Server Info
@       IN      NS      dns1

;IP address of Domain Name Server
dns1    IN      A       192.168.1.254;
;
@       IN      A       127.0.0.1
@       IN      AAAA    ::1

;Slurm Nodes
slurmctld       IN      A       192.168.1.11
dawson          IN      A       192.168.1.254
batch12         IN      A       192.168.1.12
ubuntu21        IN      A       192.168.1.21
~
~
~
~
"/etc/bind/corp.example.com" 25L, 476C
```

Figura D.6: Definición de zona de reenvío.

```
;
; BIND reverse data file for local loopback interface
;
$TTL    604800
@       IN      SOA     corp.example.com. root.corp.example.com. (
                              1         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;Domain Name Server info
@       IN      NS      dns1.corp.example.com.
dns1    IN      A       192.168.1.254

;Reverse lookup for Domain Name Server
254     IN      PTR     dns1.corp.example.com

;Slurm Nodes
11      IN      PTR     slurmctld.corp.example.com
12      IN      PTR     batch12.corp.example.com
21      IN      PTR     ubuntu21.corp.example.com

;
@       IN      NS      localhost.
1.0.0   IN      PTR     localhost.
~
~
~
~
"/etc/bind/corp.example.com.reverse" 25L, 563C
```

Figura D.7: Definición de zona de reenvío inverso.

Una vez se tiene estos ficheros listos se puede reiniciar el servidor DNS y dar soporte a las máquinas que lo deseen. Para reiniciar es suficiente con ejecutar el comando **sudo service named restart**.

D.4. Firewall

Se ha tomado la decisión de configurar un firewall en el router para aumentar la seguridad de este servidor ya que está expuesto al exterior a través de una dirección IP pública. Las reglas configuradas se verán en el código D.2 y D.3

Código D.2: Reglas manuales firewall para filtrar y controlar el tráfico en el router

```
#conexiones SSH
sudo ufw allow 22/tcp

#Tunel IPSec
sudo ufw allow 500/udp
```

```
#Tunel IPSec
sudo ufw allow 4500/udp

#consultas DNS
sudo ufw allow 53/tcp

#consultas DNS
sudo ufw allow 53/udp

#Permitir consultas DNS desde GCP
sudo ufw allow from 35.199.192.0/19 to 192.168.1.254 port 53/udp

#Permitir consultas DNS desde el cluster de GCP
sudo ufw allow from 10.0.0.0/16 to 192.168.1.254 port 53/udp

#Permitir trafico DNS saliente
sudo ufw allow out 53
```

Código D.3: Reglas de firewall para el Túnel IPSec

```
# nat Table rules
*nat
-A POSTROUTING -s 10.0.0.0/16 -o enp3s0 -m policy --pol ipsec --
    dir out -j ACCEPT
-A POSTROUTING -s 10.0.0.0/16 -o enp3s0 -j MASQUERADE

COMMIT

*mangle
-A FORWARD --match policy --pol ipsec --dir in -s 10.0.0.0/16 -o
    enp3s0 -p tcp -m tcp --tcp-flags SYN,RST SYN -m tcpmss --mss
    1361:1536 -j TCPMSS --set-mss 1360
COMMIT

*fitler
-A ufw-before-forward --match policy --pol ipsec --dir in --proto
    esp -s 10.0.0.0/16 -j ACCEPT
-A ufw-before-forward --match policy --pol ipsec --dir out --proto
     esp -d 10.0.0.0/16 -j ACCEPT
```

En el código D.2 se pueden observar reglas para permitir tráfico entrante y saliente para el DNS, tráfico del Túnel IPSec y una regla para permitir conexiones SSH. Mientras que en el código D.3 se observan reglas de firewall dedicadas mas para el Túnel IPSec, estas son las encargadas de redirigir, aceptar y filtrar el tráfico entrante y saliente del túnel.

Anexo E

Validación y pruebas

La evaluación del sistema es una parte imprescindible para dar luz verde a un sistema. En este trabajo no ha sido menos y se han llevado a cabo dos tipos de evaluaciones. En primer lugar, se ha validado el módulo de Terraform y el role de Ansible.

E.1. Validación del módulo de Terraform

Para validar el módulo de Terraform se ha usado el software de Chef Inspec[1]. Este software permite declarar el estado final que se desea del sistema y posteriormente lanzar la tarea para ver si el estado buscado es igual al estado que se encuentra en el sistema desplegado. *Inspec* hace uso del lenguaje de programación *Ruby* para el desarrollo de sus tests.

A continuación veremos partes del código del los tests elaboradoras para validar el sistema.

E.1.1. Validar la creación de recursos

Código E.1: Validar el estado de la VPC

```
control "Check VPC status" do
  impact 1.0
  title "Ensure the VPC network and subnet are deployed"
  describe google_compute_network(project: gcp_project_id, name:
      vpc_network) do
    it { should exist }
    its ('subnetworks.count') { should be >= 1 }
  end

  describe google_compute_subnetwork(project: gcp_project_id,
     region: gcp_project_region, name: vpc_subnetwork) do
    it { should exist }
    its('ip_cidr_range') { should eq '10.0.0.0/16' }
    its('network') { should match vpc_network }
  end
end
```

En las lineas del código E.1 se observa un test que comprueba dos cosas. La primera es que exista una red y que tenga un subred en ella. Y, la segunda es que exista una subred cuyo rango de direcciónes IP sea igual que '10.0.0.0/16'.

Código E.2: Validar el estado del firewall

```
control "Check Firewall Rules" do
  impact 0.8
  title "Ensure the project has the required firewall rules"

  describe google_compute_firewalls(project: gcp_project_id) do
```

[1] https://docs.chef.io/inspec/

```
    its('count') { should be > 8 }
  end

  describe google_compute_firewall(project: gcp_project_id, name
      : 'allow-slurm-outbound') do
    its('direction') { should eq "EGRESS" }
    its('destination_ranges') { should =~
        ["192.168.1.0/24","10.0.0.0/16"] }
    it { should allow_port_protocol("6817", "tcp") }
    it { should allow_port_protocol("6817", "udp") }
  end
```

Otra validación que se realiza es la del firewall, como se observa en el código E.2. Aquí se comprueba que haya más de 8 reglas que firewall (9 son las reglas que exigimos que haya) y, el otro test que se hace es comprobar que existe una regla de firwall de tipo *EGRESS* que permita tráfico a las subredes "192.168.1.0/24z "10.0.0.0/16.ª través del puerto 6817 por TCP y UDP.
Una vez ejecutados los test obtenemos una salida como la de la figura E.1

```
✔  Check nodes status: Ensure the project has the required nodes configured
   ✔  google_compute_instances count is expected to eq 5
   ✔  Instance gcp0 is expected to exist
   ✔  Instance gcp0 machine_type is expected to match "e2-small"
   ✔  Instance gcp0 status is expected to eq "RUNNING"
   ✔  Instance gcp1 is expected to exist
   ✔  Instance gcp1 machine_type is expected to match "e2-small"
   ✔  Instance gcp1 status is expected to eq "RUNNING"
   ✔  Instance gcp2 is expected to exist
   ✔  Instance gcp2 machine_type is expected to match "e2-small"
   ✔  Instance gcp2 status is expected to eq "RUNNING"
   ✔  Instance gcp3 is expected to exist
   ✔  Instance gcp3 machine_type is expected to match "e2-small"
   ✔  Instance gcp3 status is expected to eq "RUNNING"
   ✔  Instance gcp4 is expected to exist
   ✔  Instance gcp4 machine_type is expected to match "e2-small"
   ✔  Instance gcp4 status is expected to eq "RUNNING"
✔  Check DNS status: Ensure the project has the required DNS resources
   ✔  ManagedZone terraform-test-private-zone is expected to exist
   ✔  ManagedZone terraform-test-private-zone dns_name is expected to cmp == "gcp.example.com."
   ✔  ManagedZone terraform-test-private-zone description is expected to cmp == "DNS private zone por GCP nodes"
   ✔  ManagedZone terraform-test-private-zone visibility is expected to eq "private"


Profile: Google Cloud Platform Resource Pack (inspec-gcp)
Version: 1.8.8
Target:  gcp://764086051850-6qr4p6gpi6hn506pt8ejuq83di341hur.apps.googleusercontent.com

     No tests executed.

Profile Summary: 5 successful controls, 0 control failures, 0 controls skipped
Test Summary: 68 successful, 0 failures, 0 skipped
```

Figura E.1: Salida por terminal al completar los tests que comprueban la creación del sistema.

Si alguno de los test fallase, en la figura E.1 veríamos lineas en rojo y el *Test Summary* nos indicaria cuantos han fallado y cuales.

E.1.2. Validar la destrucción de los recursos

Validar los recursos tras destruir todos los del proveedor es importante para comprobar que no se queda ningún recurso pendiente de destrucción o sin destruir. Para ello se han elaborado tests que comprueban los siguientes puntos:

- No hay ninguna red ni subred en la VPC.
- No hay reglas de firwall.
- No se han dejado reglas de reenvío, IPs o elementos relacionado con el túnel VPN.
- No existe ningún nodo en el proyecto.

- No quedan elementos relacionados con el DNS.

Código E.3: Validar que no hay redes ni subredes en la VPC

```
  control "Check VPC status" do
                                                                    # A
    unique ID for this control
  impact 1.0

    # The criticality, if this control fails.
  title "Ensure the VPC network and subnet are deployed"
                                # A human-readable title
  describe google_compute_network(project: gcp_project_id, name:
      vpc_network) do    # The actual test
    it { should_not exist }
  end

  describe google_compute_subnetwork(project: gcp_project_id,
    region: gcp_project_region, name: vpc_subnetwork) do
    it { should_not exist }
  end
end
```

En el código E.3 se observa un fragmento del test que comprueba que no hay ninguna red ni subred en la VPC.
Una vez destruidos los recursos, se lanzan los tests y se obtiene una salida como la de la figura E.2. Si alguno de los recursos se queda pendiente en el sistema el test fallaría e indicaría que recursos quedan pendientes.

```
✔  Check VPC status: Ensure the VPC network and subnet are deployed
   ✔  Network slurm-vpc is expected not to exist
   ✔  Subnetwork slurm-vpc-subnet-1 is expected not to exist
✔  Check Firewall Rules: Ensure the project has the required firewall rules
   ✔  google_compute_firewalls count is expected to >= 4
✔  Check VPN configuration: Ensure the project has the required VPN configuration
   ✔  Address terraform-test-vpn-static-ip is expected not to exist
   ✔  ForwardingRule terraform-test-gcp-to-cluster-rule-udp500 is expected not to exist
   ✔  ForwardingRule terraform-test-gcp-to-cluster-rule-udp4500 is expected not to exist
   ✔  ForwardingRule terraform-test-gcp-to-cluster-rule-esp is expected not to exist
   ✔  Route terraform-test-route1 is expected not to exist
   ✔  VpnTunnel terraform-test-tunnel is expected not to exist
✔  Check nodes status: Ensure the project has the required nodes configured
   ✔  google_compute_instances count is expected to eq 0
   ✔  Instance gcp0 is expected not to exist
   ✔  Instance gcp1 is expected not to exist
   ✔  Instance gcp2 is expected not to exist
   ✔  Instance gcp3 is expected not to exist
   ✔  Instance gcp4 is expected not to exist
✔  Check DNS status: Ensure the project has the required DNS resources
   ✔  ManagedZone terraform-test-private-zone is expected not to exist


Profile: Google Cloud Platform Resource Pack (inspec-gcp)
Version: 1.8.8
Target:  gcp://764086051850-6qr4p6gpi6hn506pt8ejuq83di341hur.apps.googleusercontent.com

     No tests executed.

Profile Summary: 5 successful controls, 0 control failures, 0 controls skipped
Test Summary: 16 successful, 0 failures, 0 skipped
```

Figura E.2: Salida por terminal al completar los tests que comprueban la destrución del sistema.

E.2. Validación del *role* de Ansible

Para validar el *role* de Ansible se ha probado con los *Playbooks* que se ven en el anexo C.1. Tras aplicar cada *Playbook* se obtiene una salida como la de la imagen E.3 donde se observa la traza de ejecución y da información de si algo ha fallado (En la última línea se ve una linea roja donde nos dice que ha fallado, sin embargo, si se sigue leyendo se puede observar que es un mensaje de fallo intencionado por el desarrollador). Sin embargo, en la figure E.4 se puede observar un fallo real al intentar llevar a cabo la instalación.

```
hayk@dawson:~/ansible/ansible_slurm/tests$ ansible-playbook rollback.yaml

PLAY [Uninstalling slurm] ******************************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************************************
ok: [192.168.1.40]

TASK [ansible_slurm : Replace slurm.conf file] *********************************************************************************************
skipping: [192.168.1.40]

TASK [ansible_slurm : Stopping slurm_roles] ************************************************************************************************
skipping: [192.168.1.40]

TASK [ansible_slurm : Rolling Back] ********************************************************************************************************
included: /home/hayk/ansible/ansible_slurm/tasks/rollback.yml for 192.168.1.40

TASK [ansible_slurm : Uninstalling slurm pacakges] *****************************************************************************************
ok: [192.168.1.40]

TASK [ansible_slurm : Delete slurm directory] **********************************************************************************************
changed: [192.168.1.40]

TASK [ansible_slurm : Delete slurmctld log directory] **************************************************************************************
ok: [192.168.1.40]

TASK [ansible_slurm : Delete slurmctld state directory] ************************************************************************************
ok: [192.168.1.40]

TASK [ansible_slurm : Delete slurmd log directory] *****************************************************************************************
ok: [192.168.1.40]

TASK [ansible_slurm : Delete slurm spool directory] ****************************************************************************************
ok: [192.168.1.40]

TASK [ansible_slurm : Stopping slurm_roles] ************************************************************************************************
fatal: [192.168.1.40]: FAILED! => {"changed": false, "msg": "THIS FAIL IS INTENTIONED. ROLE STOPPED BECAUSE OF ROLLBACK"}

PLAY RECAP *********************************************************************************************************************************
192.168.1.40               : ok=8    changed=1    unreachable=0    failed=1    skipped=2    rescued=0    ignored=0

hayk@dawson:~/ansible/ansible_slurm/tests$
```

Figura E.3: Salida por terminal al aplicar un Playbook de 'rollback'.

```
hayk@dawson:~/ansible/ansible_slurm/tests$ ansible-playbook install_compute.yaml

PLAY [Installing compute] ******************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************************
ok: [192.168.1.40]

TASK [ansible_slurm : Replace slurm.conf file] *********************************************************************************
skipping: [192.168.1.40]

TASK [ansible_slurm : Stopping slurm_roles] ************************************************************************************
skipping: [192.168.1.40]

TASK [ansible_slurm : Rolling Back] ********************************************************************************************
skipping: [192.168.1.40]

TASK [ansible_slurm : Stopping slurm_roles] ************************************************************************************
skipping: [192.168.1.40]

TASK [ansible_slurm : Include user creation task] ******************************************************************************
included: /home/hayk/ansible/ansible_slurm/tasks/user.yml for 192.168.1.40

TASK [ansible_slurm : Create slurm  grooup] ************************************************************************************
ok: [192.168.1.40]

TASK [ansible_slurm : Create slurm user] ***************************************************************************************
ok: [192.168.1.40]

TASK [ansible_slurm : Include slurmctld node configuration] ********************************************************************
skipping: [192.168.1.40]

TASK [ansible_slurm : Include slurmd node configuration] ***********************************************************************
included: /home/hayk/ansible/ansible_slurm/tasks/slurmd.yml for 192.168.1.40

TASK [ansible_slurm : Update APT Cache] ****************************************************************************************
fatal: [192.168.1.40]: FAILED! => {"changed": false, "msg": "Failed to update apt cache: unknown reason"}

PLAY RECAP *********************************************************************************************************************
192.168.1.40               : ok=5    changed=0    unreachable=0    failed=1    skipped=5    rescued=0    ignored=0
```

Figura E.4: Salida por terminal al aplicar un Playbook de 'install_compute' que acaba en error.

E.3. Pruebas en entorno de prueba

Para validar el sistema se ha llevado a cabo una serie de pruebas haciendo uso del cluster de Slurm, para ello se han lazando tareas tanto en el cluster on-premise, en el cluster de GCP, como en el cluster híbrido. Estas pruebas han sido varias, desde las más simples hasta algunas más complejas que explotan el potencial de Slurm. A continuación, se listan las pruebas ejecutadas con Slurm, la explicación de lo que hacen y el porque de esta prueba.

Comprobar los nodos on-premise.

Se prueba con el comando **$srun –N 24 hostname**. Con este comando se quiere probar que, realmente, los 24 nodos (el total de nodos activos en el momento de la prueba) se encuentran en buen estado y pueden realizar tareas. Al ser la partición por defecto la de los nodos on-premise no hay que decirle que lo queremos en esa partición.

Comprobar los nodos del cluster GCP.

Se prueba con el comando **$srun ––partition=gcp –N 4 hostname**. Le especificamos que queremos que ejecute en 4 nodos de la partición llamada *gcp* el comando hostname. El resultado debe ser que los 4 nodos devuelvan su nombre.

Comprobar la paralelización de tareas.

En este caso no se prueba con un comando sino que se hace uso de un programa en *bash* que ejecuta múltiples sentencias *srun* (ver código E.4).
Este código lanza 4 tareas. La primera en los 4 nodos de la partición GCP; la segunda en 10 nodos de la partición tfg (nodos on-premise); la tercera en 6 nodos mixtos, es deicr, en 6 nodos que pueden ser tanto de GCP como on-premise; y la cuarta de nuevo igual que la tercera.

Código E.4: Programa que somete múltiples tareas paralelas

```
#!/bin/bash
#fichero batchTask.sh

srun -l --partition=gcp  --job-name=cloud    -N 4  './script.sh' &
srun -l --partition=tfg  --job-name=onprem   -N 10 './script.sh' &
srun -l --partition=mix  --job-name=hybrid   -N 6  './script.sh' &
srun -l --partition=mix  --job-name=hybrid2  -N 6  './script.sh' &
```

Este programa (o script) se ejecuta con el comando **$sbatch batchTask.sh** y se encarga de someter 4 tareas de Slurm simultáneamente en diferentes nodos que correrán el contenido del fichero *script.sh*. Esta prueba trata de demostrar que Slurm es capaz de paralelizar tareas, asignar recursos en un mismo nodo a diferentes tareas y gestionar los recursos del cluster adecuadamente.

Comprobar la paralelización de tareas exclusivas.

Esta prueba es similar a la anterior, pero esta trata de probar que al someter tareas paralelas que son exclusivas no hay conflicto de nodos. Con exclusiva se refiere a que no pueden estar compartiendo nodo ya que puede dar problemas con el programa que se ejecuta en la tarea. En el código E.5 se observa el programa que lana 4 tareas paralelas exclusivas

Código E.5: Programa que somete múltiples tareas exclusivas paralelamente

```
#!/bin/bash

srun -l --partition=gcp  --exclusive=user --job-name=cloud    -N 4
    './script.sh' &

srun -l --partition=tfg  --exclusive=user --job-name=onprem   -N 10
    './script.sh' &

srun -l --partition=mix  --exclusive=user --job-name=hybrid   -N 6
    './script.sh' &

srun -l --partition=mix  --exclusive=user --job-name=hybrid2  -N 6
    './script.sh' &
```

E.4. Pruebas en entorno real de simulación distribuida

Una vez comprobado que el cluster está funcionando correctamente en la sección E.3, era hora de probar el simulador de SED distribuido con la infraestructura de Slurm. Para ello se han llevado a cabo estas pruebas.

- Lanzar el simulador en nodos on-premise.
- Lanzar el simulador en nodos GCP.
- Lanzar el simulador haciendo uso de nodos on-premise y nodos GCP.

Las pruebas en el entorno real se han llevado a cabo haciendo uso de programas *bash*, que se ejecutan con el comando **$sbatch** de Slurm, debido a la facilidad que supone la configuración del proceso que lanzará Slurm. Se podría haber hecho a través del comando **$srun** pero no ofrece la misma comodidad de parametrización. A continuación, se verán los tres programas *bash* para lanzar el simulador en tres entornos, así mismo, se ofrecerá la alternativa en formato *srun* para que se pueda contemplar esta posibilidad.

Lanzar el simulador en nodos on-premise.

Con el programa E.6 preparamos una tarea que se lanza en cuatro nodos de la partición *tfg* (cluster on-premise), estos cuatro nodos deben estar en el rango de máquinas con *hostname* *ubuntu31* hasta *ubuntu34*. Esta tarea se lanzará de manera exclusiva. Para lanzar el programa sería suficiente con ejecutar **$sbatch nombrePrograma.sh**. La alternativa en formato *srun* sería la siguiente:

```
$ srun -N 4 -o output%j.txt -e error%j.txt -p tfg
--nodelist=ubuntu[31-34] --exclusive=user
/home/ubuntu/esimc 10ks3ramas_on_prem 1000000
'[192.168.1.31:20000,192.168.1.32:20000,192.168.1.33:20000,
192.168.1.34:20000]'
```

Código E.6: Programa para lanzar el simulador en el cluster on-premise

```
#!/bin/bash
#SBATCH --nodes=4
#SBATCH --output=output%j.txt
#SBATCH --error=error%j.txt
#SBATCH --job-name=onprem
#SBATCH --partition=tfg
#SBATCH --nodelist=ubuntu[31-34]
#SBATCH --exclusive=user

# La siguiente linea corresponde a la forma en la que se lanza el
    simulador. Esta consiste en lanzar el ejecutable esimc junto a
    la red a simular, el n mero de ciclos y la lista de nodoip

srun /home/ubuntu/esimc 10ks3ramas_on_prem 1000000
    '[192.168.1.31:20000,192.168.1.32:20000,
192.168.1.33:20000,192.168.1.34:20000]'
```

Con el código de este simulador se tropieza ante un problema que nos limita el uso de la potencia de Slurm. Este problema consiste en que el simulador debe conocer de antemano los nodos donde va a correr la simulación, por lo que se debe especificar a Slurm los nodos que se quieren usar ya que sino la correspondencia de IPs que le pasamos a Slurm y la del nodo donde se ejecuta no sería correcta. La limitación que lleva este problema es que no se está dejando en mano de Slurm la asignación de nodos, sino que se le pide de antemano que sean ciertos nodos los que se van a usar.

Lanzar el simulador en nodos GCP.

En este caso se usa el código E.7 para lanzar el simulador en nodos del cluster de GCP. En este caso se usa un programa similar al de E.6 pero se puede observar que no se le especifican los nodos que queremos usar, esto se debe a que solamente existen **cuatro** nodos de GCP por lo que no se necesita indicarlo explícitamente.

Código E.7: Programa para lanzar el simulador en el cluster GCP

```
#!/bin/bash
#SBATCH --nodes=4
#SBATCH --output=output%j.txt
#SBATCH --error=error%j.txt
#SBATCH --job-name=gcp
#SBATCH --partition=gcp
#SBATCH --exclusive=user

# La siguiente linea corresponde a la forma en la que se lanza el
    simulador. Esta consiste en lanzar el ejecutable esimc junto a
    la red a simular, el n mero de ciclos y la lista de nodo:ip

srun /home/ubuntu/esimc 10ks3ramas 1000000 '[10.0.0.27:20000,
10.0.0.28:20000,10.0.0.29:20000,10.0.0.30:20000]'
```

En este caso la limitación de la prueba anterior sigue siendo la misma porque se usa el mismo simulador.

Lanzar el simulador haciendo uso de nodos on-premise y nodos GCP.

En esta último programa que se observa en el código E.8 se vuelve a la dinámica de la

primera prueba ya que de nuevo debemos especificarle los nodos que queremos usar. Esta especificación explícita es necesario ya que hacemos uso de la partición **mix** que incluye tanto nodos GCP como on-premise, por lo que, Slurm, podría asignar tanto solo nodos on-premise, solo nodos GCP (si hubiese el número suficiente), como una mezcla de las anteriores. Al ser una prueba que se necesita que haga uso de los dos tipos de nodos se le fuerza a que haga uso de los cuatro nodos de GCP y 2 nodos del cluster on-premise.

Código E.8: Programa para lanzar el simulador en el cluster híbrido

```
#!/bin/bash
#SBATCH --nodes=6
#SBATCH --output=hybrid.txt
#SBATCH --error=hybriderr.txt
#SBATCH --job-name=hybrid
#SBATCH --partition=mix
#SBATCH --nodelist=gcp1.gcp.example.com,gcp2.gcp.example.com,gcp4.
    gcp.example.com,gcp5.gcp.example.com,ubuntu[31-32]
#SBATCH --exclusive=user



# La siguiente linea corresponde a la forma en la que se lanza el
    simulador. Esta consiste en lanzar el ejecutable esimc junto a
    la red a simular, el n mero de ciclos y la lista de nodo:ip

srun /home/ubuntu/esimc 10ks5ramas 1000000 '[10.0.0.27:20000,
10.0.0.28:20000,10.0.0.29:20000,10.0.0.30:20000,192.168.1.31:20000,

192.168.1.32:20000]'
```

La limitación que se ha visto en las dos anteriores pruebas sigue existiendo al usar el mismo simulador. En el supuesto caso de que no se debiese especificar como parámetro de llamada al simulador las direcciones IP de los nodos, se podría ahorrar la especificación de los nodos y dejar en manos de Slurm la asignación de nodos. No obstante, esto no asegura que se hiciese uso de nodos GCP y on-premise mezclados.

Lanzar "paralelamente"simulaciones en varios entornos cloud.

Finalmente, esta prueba demuestra la "paralelización"de tareas con Slurm. Para ello se usa el código E.9 para lanzar las tres pruebas anteriores de manera paralela.
Lo que se intenta buscar con esta prueba es la **asignación y gestión de recursos por parte de Slurm**. Para conseguir esto se lanzan las tres simulaciones, on-premise, GCP e híbrido. Y se observa en la cola de procesos de Slurm como se gestionan los recursos.

En la figura E.5 se ve la cola de procesos al lanzar el programa E.9 y se pueden ver tres tareas en la cola.

- La primera tarea con *jobid* 252, es la tarea híbrida. Esta tarea se encuentra en un estado *PD*, es decir, *pending*, ya que está esperando recursos.
- La segunda tarea con *jobid* 253, es la tarea en GCP. Esta tarea se encuentra en un estado *R*, es decir, *ready*, ya que tiene ya nodos asignados y puede ser ejecutado.
- La tercera tarea con *jobid* 254, es la tarea en on-premise. Esta tarea, como el anterior, está en estado *R* ya que también tiene recursos para poder ser ejecutado.

La tarea en entorno híbrido está esperando recursos ya que comparte los nodos de GCP con la tarea de Cloud Público, por lo tanto, cuando la tarea de GCP termine, el híbrido puede empezar a ser ejecutado.
Si no se tuviese la limitación explicada en las pruebas anteriores y además el simulador programado pudiese gestionar varias instancias bajo una misma dirección IP, se podría paralelizar y lanzar las tres simulaciones a la vez corriendo en los mismos nodos y a la par. Esta tarea se deja pendiente para el trabajo a futuro que se puede continuar en este proyecto.

Código E.9: Paralelización de simulaciones con Slurm

```
#!/bin/bash

sbatch simulacion_onprem_3ramas.script &
sbatch simulacion_gcp_3ramas.script &
sbatch simulacion_hybrid_5ramas.script &
```

```
Every 0.1s: squeue                                                    slurmctld: Tue Aug 31 18:43:03 2021

             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
               252       mix   hybrid   ubuntu PD       0:00      6 (Resources)
               253       gcp      gcp   ubuntu  R       0:01      4 gcp1.gcp.example.com,gcp2.gcp.example.com,gcp4.gcp.exa
mple.com,gcp5.gcp.example.com
               254       tfg   onprem   ubuntu  R       0:01      4 ubuntu[31-34]
```

Figura E.5: Cola de procesos de Slurm con tareas paralelas.

Anexo F

Documentación de los recursos creados

La documentación es clave para que futuros compañeros que procedan a continuar en este proyecto tengan claro lo que se hizo y lo que pueden mejorar. Para ello se ha llevado a cabo documentación detallada de todos los recursos, programas y herramientas usados y creados. Esta memoria es el informe principal del trabajo y el método de documentación de todos los procesos seguidos por lo que es un buen medio para comprender y replicar el trabajo realizado.

A parte de la memoria se cuenta con otro tipo de documentación como la bibliografía (capítulo **??**) que da pie a paginas web o manuales de instalación y configuración de diferentes partes del trabajo.

Por otro lado, se ha documentado el módulo de Terraform (sección C.3) a través del uso de ficheros Markdown (un lenguaje de marcado ligero), una práctica recomendada por Terraform (figura F.1).

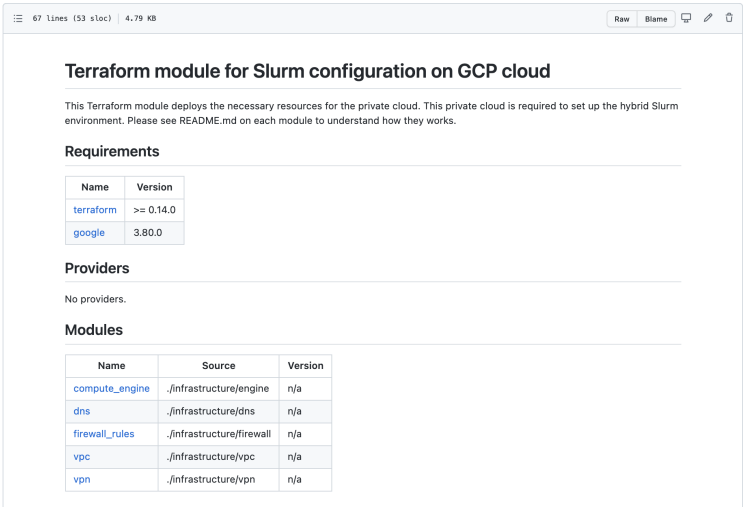
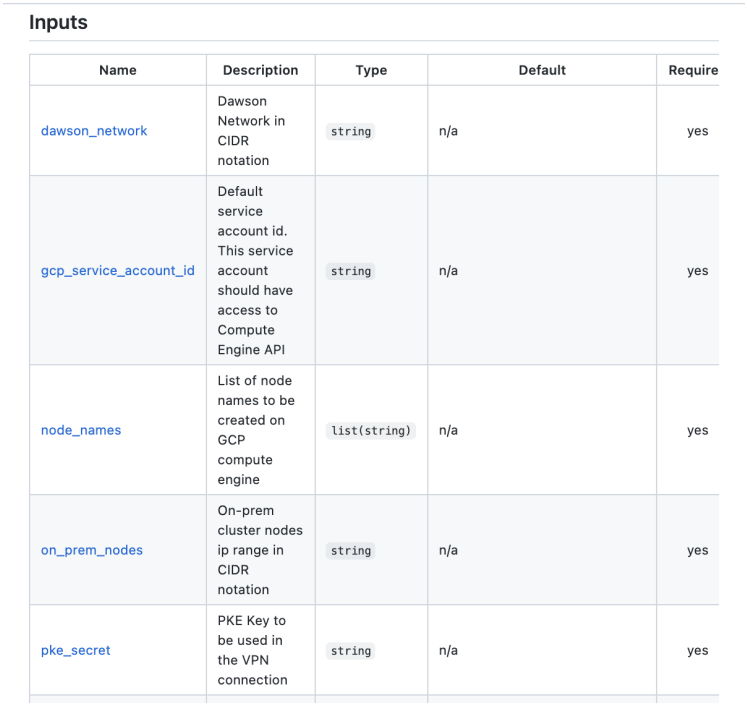


Figura F.1: Documentación del módulo de Terraform

Esta documentación se ha generado de manera automática a través del uso de un software llamado **pre-commit** [1]. Este software se encarga de que que antes de realizar un *push* al repositorio se comprueban unos requisitos y se pasen de manera exitosa. En el caso de este trabajo se ha pedido entre otras cosas que se genera una documentación automática de las variables de salida y entrada que se usan en el módulo de Terraform (ver figuras F.2 y F.3).

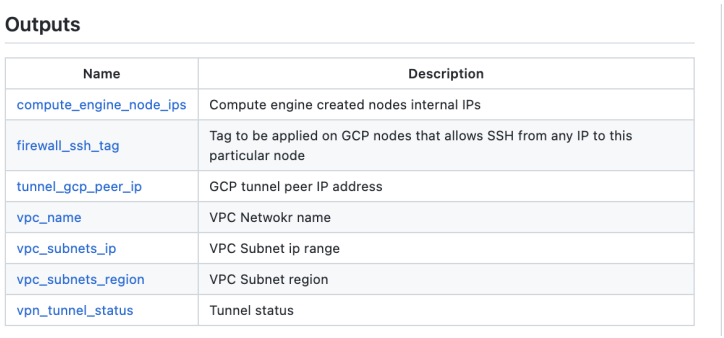
[1] https://pre-commit.com/



Inputs

Name	Description	Type	Default	Require
dawson_network	Dawson Network in CIDR notation	`string`	n/a	yes
gcp_service_account_id	Default service account id. This service account should have access to Compute Engine API	`string`	n/a	yes
node_names	List of node names to be created on GCP compute engine	`list(string)`	n/a	yes
on_prem_nodes	On-prem cluster nodes ip range in CIDR notation	`string`	n/a	yes
pke_secret	PKE Key to be used in the VPN connection	`string`	n/a	yes

Figura F.2: Documentación auto-generada de las variables de entrada del módulo de Terraform

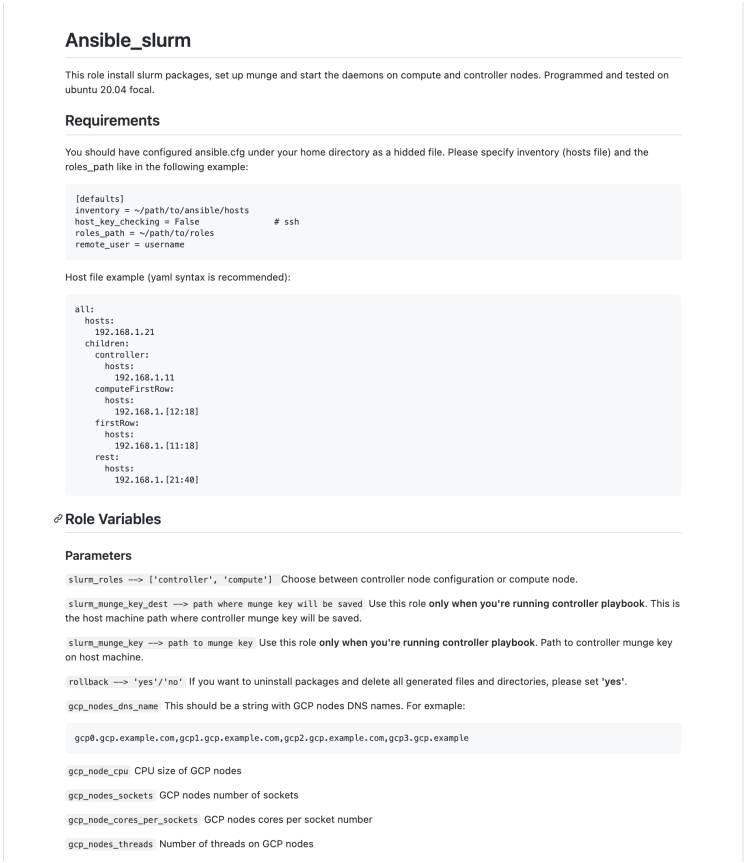


Outputs

Name	Description
compute_engine_node_ips	Compute engine created nodes internal IPs
firewall_ssh_tag	Tag to be applied on GCP nodes that allows SSH from any IP to this particular node
tunnel_gcp_peer_ip	GCP tunnel peer IP address
vpc_name	VPC Netwokr name
vpc_subnets_ip	VPC Subnet ip range
vpc_subnets_region	VPC Subnet region
vpn_tunnel_status	Tunnel status

Figura F.3: Documentación auto-generada de las variables de salida del módulo de Terraform

Con Ansible se ha llevado a cabo un proceso similar pero sin la automatización de la documentación. De nuevo en un fichero de Markdown (ver figura F.4) se ha detallado la explicación de como se usa este *role*, los diferentes ejemplos de *playbook* y las variables necesarias. Esto hace que la tarea de comprensión del *role* y su futuro uso sea más fácil para futuros usuarios.



Ansible_slurm

This role install slurm packages, set up munge and start the daemons on compute and controller nodes. Programmed and tested on ubuntu 20.04 focal.

Requirements

You should have configured ansible.cfg under your home directory as a hidded file. Please specify inventory (hosts file) and the roles_path like in the following example:

```
[defaults]
inventory = ~/path/to/ansible/hosts
host_key_checking = False               # ssh
roles_path = ~/path/to/roles
remote_user = username
```

Host file example (yaml syntax is recommended):

```
all:
  hosts:
    192.168.1.21
  children:
    controller:
      hosts:
        192.168.1.11
    computeFirstRow:
      hosts:
        192.168.1.[12:18]
    firstRow:
      hosts:
        192.168.1.[11:18]
    rest:
      hosts:
        192.168.1.[21:40]
```

Role Variables

Parameters

`slurm_roles --> ['controller', 'compute']` Choose between controller node configuration or compute node.

`slurm_munge_key_dest --> path where munge key will be saved` Use this role **only when you're running controller playbook**. This is the host machine path where controller munge key will be saved.

`slurm_munge_key --> path to munge key` Use this role **only when you're running controller playbook**. Path to controller munge key on host machine.

`rollback --> 'yes'/'no'` If you want to uninstall packages and delete all generated files and directories, please set **'yes'**.

`gcp_nodes_dns_name` This should be a string with GCP nodes DNS names. For exmaple:

```
gcp0.gcp.example.com,gcp1.gcp.example.com,gcp2.gcp.example.com,gcp3.gcp.example
```

`gcp_node_cpu` CPU size of GCP nodes

`gcp_nodes_sockets` GCP nodes number of sockets

`gcp_node_cores_per_sockets` GCP nodes cores per socket number

`gcp_nodes_threads` Number of threads on GCP nodes

Figura F.4: Documentación del role de Ansible.

Para el resto de configuraciones como Slurm, servidores DNS y VPN, y Túnel IPSec se puede tomar de referencia la documentación del proceso detallada en esta memoria.