Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Telemática

Trabajo Fin de Grado

Desarrollo de una nube híbrida de computación

Autor: Silviu Constantin Sofrone

Tutor: José Manuel Arco Rodríguez

2022

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería Telemática

Trabajo Fin de Grado

Desarrollo de una nube híbrida de computación

Autor: Silviu Constantin Sofrone

Tutor: José Manuel Arco Rodríguez

TRIBUNAL:

Presidente: García Herraiz, Antonio

Vocal 1º: Navarro Guillén, Andrés

Vocal 2º: Arco Rodríguez, José Manuel

FECHA: septiembre de 2022

Agradecimientos

A todos los amigos y docentes que me han acompañado en este camino y han influido en mi etapa académica.

Y en especial a mi familia, a Francisco y a Marcos, por vuestro apoyo incondicional y por forjarme como la persona que soy en la actualidad, vosotros me guiais hacia todos mis logros.

Índice

Resumen

En el presente trabajo se exploran diversas opciones para conformar una nube híbrida compuesta por, mínimo, una nube privada y una pública. Para la privada se ha utilizado la plataforma *open source* OpenStack. Con el fin de conectar y operar OpenStack con una nube pública, se ha intentado integrar el proyecto Omni con varios despliegues (OpenStack-Ansible y Devstack) encontrando numerosos problemas, algunos sin solución.

Para conformar una nube híbrida necesitamos control unificado de las nubes y comunicación entre sus máquinas virtuales. El control se ha realizado con Juju sobre OpenStack, AWS y GCP, pero la comunicación solo ha sido posible entre las nubes públicas mediante una *multi-cloud*.

Palabras clave: nube híbrida, OpenStack-Ansible, DevStack, Juju, *multi-cloud*.

Abstract

This paper explores various options for building a hybrid cloud consisting of at least one private and one public cloud. For the private cloud, the open source platform OpenStack has been used. In order to connect and operate OpenStack with a public cloud, we have tried to integrate the Omni project with several deployments (OpenStack-Ansible and Devstack) obtaining numerous problems, some of them unsolvable.

To form a hybrid cloud we need unified control of the clouds and communication between them. The control has been done with Juju on OpenStack, AWS and GCP, but communication has only been possible between the public clouds through a multi-cloud.

Keywords: hybrid cloud, OpenStack-Ansible, DevStack, Juju, multi-cloud.

Índice de imágenes

Glosario de acrónimos y abreviaturas

API	Application Programming Interface
ASN	Autonomous System Number
AWS	Amazon Web Services
BGP	Border Gateway Protocol
CIDR	Classless Inter-Domain Routing
CLI	Command Line Interface
CMP	Cloud Management Platform
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
EC2	Elastic Compute Cloud
GCP	Google Cloud Platform
GUI	Graphical User Interface
HA VPN	Highly Available Virtual Private Network
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
IAM	Identity and Access Management
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IKE	Internet Key Exchange
IP	Internet Protocol
IPSec	Internet Protocol Security
JAAS	Juju as a Service
JSON	JavaScript Object Notation
L3	Layer 3
LXC	LinuX Containers
ML2	Modular Layer 2
OS	Operating System
OSA	OpenStack-Ansible
OVS	Open Virtual Switch
RAM	Random Access Memory
RXTX	RX Technology Holdings Incorporated
SDK	Software Development Kit
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URL	Uniform Resource Locator
vCPU	Virtual Centralized Processing Unit
VGW	Virtual Private Gateway
VLAN	Virtual Local Area Networks
VM	Virtual Machine
VPC	Virtual Private Cloud
VPN	Virtual Private Network
VXLAN	Virtual Extensible Local Area Networks
YAML	Formato de serialización de datos legible por humanos

1. Introducción

La finalidad del proyecto consiste en desplegar una nube privada OpenStack sobre la que trabajar en la búsqueda de un método que nos permita integrarla con una nube pública. Para ello, es importante analizar qué proceso de instalación es el más adecuado para nuestro cometido. Dado que el objetivo final es conformar una nube híbrida, evitaremos un despliegue complejo. Será suficiente con un método capaz de configurar una nube sencilla, completamente funciona y escalable, en vistas a que se pueda replicar lo aprendido a un entorno de producción en trabajos futuros. Factores como la capacidad de restauración de la nube o facilidades con respecto a la implementación de la nube híbrida, deberán ser también tenidos en cuenta.

Una vez creada la nube OpenStack, es primordial verificar el correcto funcionamiento de esta. Comprobar las funcionalidades básicas de sus servicios y la capacidad de comunicación con el exterior, son pasos obligatorios antes de embarcarse en el estudio de las distintas opciones disponibles para la conexión con un proveedor de nube pública. La idea inicial es implementar el sistema Omni, encargado de integrar OpenStack con los principales proveedores de nube pública.

En el capítulo 2 “Conceptos”, se exponen información previa que deberemos tener en cuenta sobre OpenStack y el concepto de nube híbrida. Los dos siguientes capítulos, “OpenStack-Ansible (OSA)” y “Operaciones en la nube OSA”, embarcan todo el proceso de configuración y despliegue de la nube basada en el despliegue de OpenStack-Ansible, y las posteriores pruebas realizadas que verifican que funciona correctamente.

A partir del capítulo 5 “Nube híbrida OpenStack Omni”, da comienzo el estudio y las pruebas de la creación de la nube híbrida. En concreto, dicho capítulo trata la cuestión de implementar el proyecto Omni en nuestra nube OSA, sin embargo, este entono no cuenta con las condiciones necesarias para integrarlo con Omni. Como alternativa, se utiliza DevStack para la instalación de OpenStack. Tras superar varios problemas, al final no se consigue que Onmi funcione con Devstack y se siguen probando otras técnicas para desplegar la nube híbrida en el siguiente capítulo.

En el capítulo 6 “Nube híbrida con Juju”, se explican las tareas de conformado de una nube híbrida con la herramienta de gestión de aplicaciones Juju. Para ello, se establece un entorno compuesto por las nubes OpenStack, Google Cloud y Amazon Web Services, permitiendo el control unificado de sus recursos a través de Juju.

Para finalizar, si bien la nube híbrida con Juju nos dota de la capacidad de gestionar los recursos de cada nube desde un punto de control, aún nos queda resolver la cuestión referente a la comunicación entre sus máquinas virtuales. Para conectar OpenStack a cualquier nube pública es necesario configurar el router de la Universidad, al cual no tenemos acceso, por tanto, se ha procedido a conectar AWS con GCP mediante una multi-cloud. Es así como llegamos al último capítulo, “Multi-cloud AWS-GCP”, en el que, aprovechando la virtualización, crearemos una VPC en cada proveedor capaces de intercambiar tráfico a través de túneles VPN.

2. Conceptos

2.1. OpenStack

OpenStack [1] es un proyecto de computación en la nube con la función de diseñar y gestionar tanto nubes públicas como privadas (dependiendo de si la infraestructura y el conjunto de recursos que forman parte del entorno *cloud* están pensados para ser utilizados por el público general o por un cliente u organización, respectivamente) ofreciendo una infraestructura como servicio (IaaS). Esto quiere decir que está orientado a brindar todos los componentes necesarios de almacenamiento, redes, imágenes y máquinas virtuales para configurar una infraestructura *cloud*. Se trata de la plataforma de *cloud computing* de software libre más grande que existe en la actualidad dadas sus múltiples ventajas [2]:

- Elasticidad: capacidad de aumentar o disminuir la infraestructura según las necesidades cambiantes de nuestro entorno y en cuestión de segundos.
- Pago por uso: independientemente de la escala que tenga la infraestructura, siempre pagaremos únicamente por los recursos que estamos consumiendo en base a su tiempo de uso.
- Proyecto de código abierto: su naturaleza *open source* se consagra como una de sus principales distinciones. El hecho de contar con un producto de tal escala de forma gratuita, y tener la libertad de consultar y modificar su código, es un gran aliciente para los usuarios a la hora de decantarse por utilizar este sistema.
- Automatización: uno de los principales puntos fuertes de OpenStack recae en la facilidad con la que se pueden automatizar tareas. Esto es en parte debido a que cuenta con su propia interfaz de programación de aplicaciones, con la que cualquier desarrollador puede crear sus propias soluciones y compartirlas con la comunidad.
- Su comunidad: tenemos a nuestra disposición una comunidad muy extensa y activa, a la que acudir en busca de respuestas a errores y mejoras para nuestro entorno.

2.1.1 Servicios y métodos de despliegue

Estamos ante un servicio complejo, debido a que está formado por numerosas herramientas software *open source* [1] que se pueden instalar individualmente o en grupo. Actualmente cuenta con una amplia variedad de estos módulos, entre los que destacan:

- Nova (Compute): el corazón de OpenStack. Se encarga de la creación y despliegue de las máquinas virtuales a partir de los múltiples hipervisores con los que puede trabajar.
- Horizon (Dashboard): la interfaz gráfica desde la que interactuar con el resto de los servicios de la nube.
- Neutron (Network): se ocupa de la generar y gestionar las redes que forman el entorno OpenStack.
- Keystone (Identity): módulo de identidad que tiene la tarea de gestionar la autenticación de usuarios y políticas.
- Glance (Image): módulo de gestión de imágenes que proporciona, principalmente, las plantillas de los sistemas operativos que utilizarán las instancias al desplegarse.
- Cinder (Block Storage): provee los dispositivos de almacenamiento a nivel de bloque.
- Swift (Object Storage): almacena los denominados “objetos”, que se definen como identidades únicas que contienen información y que se encuentran al mismo nivel unos de otros.

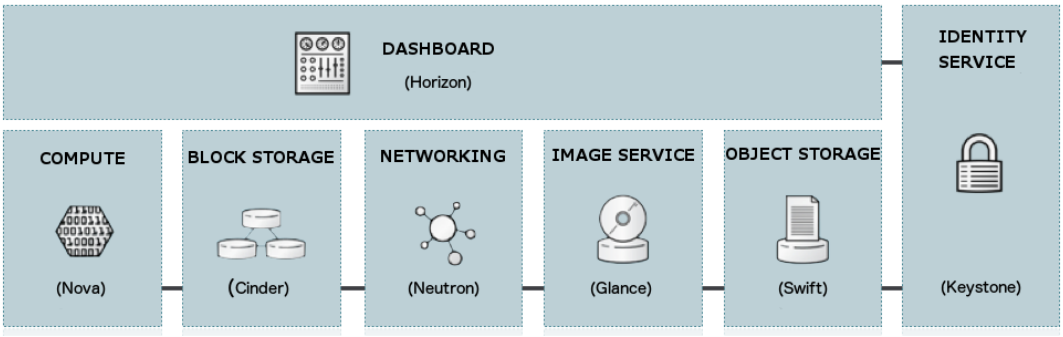

Ilustración 1 Servicios de OpenStack [1]

Todos estos servicios trabajan de manera conjunta para garantizar que OpenStack funciona correctamente. Es por ello por lo que optar por una instalación manual resulta en una tarea altamente intrincada, ya que para asegurar que al final del proceso todo está relacionado de forma apropiada, deberemos contar con conocimientos más que suficientes sobre la configuración de cada servicio. Si bien esta es la forma más indicada para aprender a crear configuraciones y como las diferentes partes de OpenStack se comunican entre sí, la comunidad ha desarrollado diferentes métodos para desplegar OpenStack [3] de una forma automatizada:

- Microstack: método de desarrollo sencillo, pero con ciertas limitaciones.
- DevStack: una de las mejores opciones para realizar un primer acercamiento, ya que, a partir de un simple script, nos permite generar un entorno OpenStack completo de forma rápida y sencilla. Pensado para ser desplegado en una máquina virtual.
- OpenStack-Ansible: método de despliegue basado en Ansible y contenedores LXC. OpenStack-Ansible aprovecha la simplicidad con la que Ansible puede configurar sistemas, implementar software y orquestar tareas, para desplegar los servicios de OpenStack en contenedores LXC a través de sus *playbooks*. Se tratan de archivos en formato *yaml* que sirven para definir el estado deseado de un sistema, de esta forma, resulta muy sencillo implementar y configurar un entorno OpenStack a partir de ellos.
- OpenStack-Charms: conjunto de herramientas *DevOps* que permite conformar una nube Openstack prácticamente a cualquier escala. Se basa en la herramienta de código libre Juju, con la que podemos crear y gestionar aplicaciones en diferentes entornos a partir de scripts en formato *yaml* capaces de realizar configuraciones de forma fácil y directa.
- Kolla & Kolla-Ansible: implementa OpenStack utilizando contenedores Kolla, orquestados a través de Ansible. Kolla se usa para construir imágenes Docker, mientras que Kolla-Ansible se encarga de aprovisionar dichas imágenes.
- OpenStack-Helm: el despliegue se realiza en contenedores usando la herramienta Helm, enfocada en la gestión de paquetes Kubernetes.

A priori, DevStack se presenta como la mejor opción para nuestro cometido, ya que nos permite instalar una nube privada OpenStack simple y completamente funcional en pocos pasos. Sin embargo, para este trabajo estudiaremos un proceso que nos proponga un grado de complejidad más equilibrado, como es el caso de OpenStack-Ansible.

2.2. Nube híbrida

Una nube privada basada en OpenStack cuenta con las características suficientes para que, en las manos adecuadas, resulte en una opción más que competente a la hora de hacer frente a una demanda de computación y procesamiento controlada y estudiada. Sin embargo, en

entornos de producción se puede dar el caso de que dicha demanda aumente más allá de las capacidades de un centro local de datos. Este es el momento indicado de recurrir a la nube pública de alguna empresa proveedora de *cloud*, con el objetivo de aumentar instantáneamente la capacidad para manejar el exceso a enfrentar. De esta forma surge la denominada nube híbrida [4], un entorno informático que permite compartir datos entre una nube privada y una pública, y de incrementar la capacidad de cómputo y almacenamiento de nuestro entorno.

Esta situación se ha convertido en un problema recurrente debido a la creciente demanda de recursos impulsada, principalmente, por el número cada vez mayor de dispositivos con acceso a Internet. Es por ello por lo que, numerosas empresas han agregado entre sus servicios, o directamente se han creado con este fin, la capacidad de escalar el entorno de un cliente a una nube híbrida. Si bien cada vez contamos con más opciones en el mercado, enfrentarse "desde cero" a esta cuestión tiene la dificultad añadida de que las soluciones disponibles son escasas. El libro "Hybrid Cloud for Architects" [4] nos ofrece una vista general sobre algunas soluciones a tener en cuenta:

- Nube híbrida basada en una CMP: la implementación se hace sobre una plataforma de gestión de la nube, o Cloud Management Platform (CMP). Se trata de un software que combina un conjunto de características o módulos capaces de gestionar diferentes entornos de nube basados en nubes privadas, públicas o una combinación de ambas. Algunos ejemplos de estas herramientas son ManageIQ, vRealize Suite o Right Scale CMP.
- Nube híbrida basada en contenedores: la configuración que nos presenta el libro consiste en el despliegue de un clúster de Kubernetes en cada nube, para posteriormente hacer que se comuniquen y trabajen entre sí mediante la función Kubernetes Federation. Dicha función otorga la capacidad de gestionar múltiples clústeres Kubernetes distintos sin la necesidad de "hablar" directamente con cada uno.
- Soluciones de nubes híbridas prediseñadas: con el objetivo de reducir el coste, la complejidad y el riesgo que supone conformar una nube híbrida, algunas empresas se han embarcado en la tarea de crear soluciones que permitan la integración entre entornos *cloud* o, directamente, ofrecen una nube híbrida previamente construida. Tenemos como ejemplos Azure Stack y OpenStack Omni, siendo este último objeto de prueba para nuestro entorno.

3. Despliegue con OpenStack-Ansible (OSA)

OpenStack-Ansible [5] es un método de despliegue mediante el cual podemos levantar un entorno OpenStack a partir de Ansible y contenedores LXC. Los servicios de OpenStack se despliegan en contenedores LXC a través de los playbooks de Ansible, manteniendo un aislamiento total entre dichos servicios y los componentes alojados en los nodos destino donde se ha realizado el despliegue.

Este método destaca con respecto al resto gracias a que aprovecha:

- **Las ventajas de Ansible:** si bien existen métodos de despliegue que utilizan herramientas de configuración automática parecidas, como *Chef* o *Puppet*, Ansible destaca por encima de ellas debido a que no se requiere de la instalación de ningún cliente o agente, sino que utiliza SSH para configurar cada *endpoint*. De hecho, Ansible sólo necesita que los hosts destino dispongan de SSH y Python. Esto hace que sea muy sencillo ejecutar sus playbooks para configurar sistemas, implementar software y orquestar tareas en entornos de cualquier tamaño y tipo.
- **El aislamiento de los contenedores LXC:** el hecho de tener los distintos servicios aislados en sus propios contenedores nos facilita las tareas de instalación de OpenStack, así como las de futuras actualizaciones.

Este enfoque está pensado para que nuestros proyectos, sin importar sus dimensiones, sean fáciles de instalar, escalar, actualizar, pero también de operar. Y es que la capacidad de escribir los playbooks contra las APIs de OpenStack y las CLI de Python, y la cantidad de módulos de Ansible para la gestión de Linux y OpenStack, han impulsado a la comunidad a desarrollar múltiples formas para automatizar operaciones frecuentes. Es por todo esto que OSA se distingue como un método especialmente apropiado para el despliegue de entornos de producción.

3.1. Resumen

3.1.1. Puentes de red

Se requiere de una configuración de red específica en los hosts destino para ejecutar OpenStack-Ansible. Los puentes de red [6] enumerados a continuación son necesarios para completar la instalación con éxito:

- **br-mgmt** (Gestión de contenedores): El puente *br-mgmt* se encarga de la gestión y la comunicación entre la infraestructura y los servicios de *OpenStack* desplegados en los contenedores.
- **br-vlan** (Proveedor de redes *OpenStack*): Este puente de capa 2 proporciona infraestructura tanto para redes con y sin (redes “flat”) etiqueta VLAN. Su función principal será la de conectar nuestra red física con la red interna de la nube privada OpenStack para qué, de esta manera, las instancias que lancemos puedan comunicarse con el exterior. El despliegue se encargará de realizar la conexión según los parámetros que le indiquemos.
- **br-vxlan** (Túnel de red *OpenStack*): La interfaz *br-vxlan* se utiliza en caso de que el entorno esté diseñado para permitir que los proyectos creen redes virtuales, así sus VMs

se comunicarán utilizando VXLAN. Proporciona la interfaz para el tráfico de red de túnel virtual encapsulado (VXLAN).

- **lxcbr0** (LXC interno): Puente configurado automáticamente por OpenStack-Ansible, necesario para proporcionar conectividad externa (típicamente Internet) a los contenedores LXC.
- **br-storage** (opcional): El puente *br-storage* permite el acceso y la comunicación entre los dispositivos de almacenamiento en bloque (Block Storage) y el resto de los servicios de *OpenStack*.

En nuestro caso configuremos la red IP de los puentes br-mgmt y br-vxlan y asignaremos una interfaz física al br-vlan. Podemos separar la gestión, que es básicamente el acceso y el uso de la API, a través de la red de gestión, a la vez que tenemos los túneles de red, ya sea para superposición (VXLAN) o para nuestra red de capa 2 (VLAN). Podemos configurar incluso una red opcional dedicada exclusivamente al almacenamiento.

3.1.2. Entorno utilizado

Ya hemos hablado sobre como las características de OpenStack-Ansible convierten a este en un proyecto muy adecuado para el despliegue de entornos de producción. Sin embargo, dado que nuestros objetivos son de docencia, no nos interesa implementar un entorno de este tipo, sino más bien uno de prueba. Por tanto, aprovecharemos el uso que hace OSA de los playbooks y roles proporcionados por Ansible para implementar y configurar un entorno *OpenStack* acorde a nuestras necesidades.

De este modo, nos apoyaremos en el ejemplo de entorno de prueba Yoga versión 24.1.0 dev53 [7] que nos ofrece la documentación oficial de OSA. Este ejemplo está compuesto por:

- **Un host de infraestructura**, dónde irán albergados los servicios de infraestructura instalados por el playbook *setup-infrastructure.yml* y servicios de OpenStack cómo keystone, glance, neutron, placement, horizon, entre otros. A su vez, será el host de despliegue encargado de ejecutar los playbooks de Ansible y de controlar los servicios de nuestra nube.
- **Un host de cómputo** encargado de proporcionar las instancias de cómputo de la nube a través de nova.
- **Un host de almacenamiento** dónde se encuentra instalado cinder, cuya función es la de aprovisionar recursos de almacenamiento en bloque (Block Storage) persistentes.

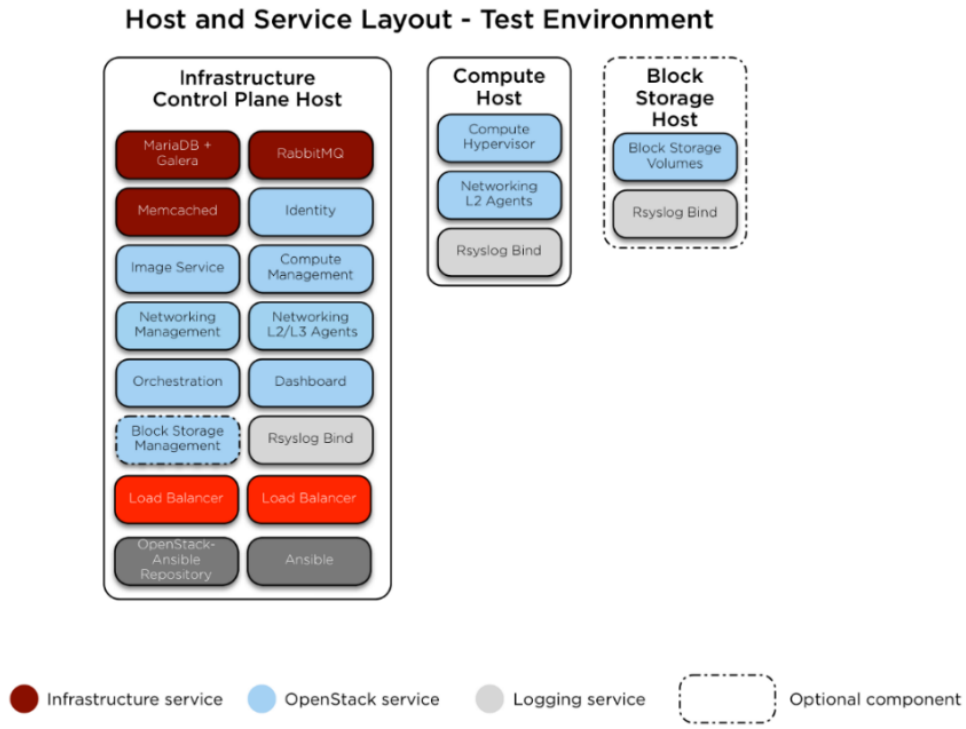

Ilustración 2 Esquema entorno de prueba [7]

Recordemos que el objetivo de nuestro sistema se limita a levantar una nube privada con la que, posteriormente, podamos conformar una nube híbrida. El uso que le daremos a sus instancias no va más allá de comprobar que poseen conectividad con el exterior (consolidando a su vez el correcto funcionamiento de la nube), no están destinadas a consumir recursos de almacenamiento. Por tanto, podemos prescindir del host de almacenamiento y, por consiguiente, de la red de almacenamiento, limitando así el número de hosts al mínimo posible.

3.1.3. Configuración de Red

Teniendo claro el tipo de entorno que vamos a utilizar, el número de hosts que lo conforma, así como el papel que desempeñará cada uno, debemos decidir el rango de direcciones que usarán las distintas redes que OSA requiere para su correcta instalación. En adición, el ejemplo de la documentación separa el tráfico de cada red por VLANs. Seguiremos dicha configuración ya que, como veremos más adelante, las VLANs son lo que nos permitirá asociar la misma interfaz para todos los puentes, aparte de ayudarnos a comprender mejor como está organizado el tráfico de nuestro sistema.

Con todo esto en cuenta, para nuestro entorno se utilizarán las siguientes asignaciones de CIDR y VLAN:

Red	CIDR	VLAN
Red de Gestión	10.0.120.0/22	10
Red de Túneles VXLAN	10.0.124.0/22	30

Dentro de dichos rangos de direcciones, las direcciones IP empleadas para cada host serán:

Nombre del Host	IP de Gestión	IP de Túnel VXLAN
infra1	10.0.123.11	10.0.125.11
compute1	10.0.123.12	10.0.125.12

*Los nombres infra 1 y compute1 hacen referencia a los hosts de infraestructura y cómputo, respectivamente, y son los nombres que utilizará OSA para identificar a cada uno.

Aclarados estos datos, procedemos a diseñar la arquitectura de red sobre la que vamos a desplegar el entorno OpenStack. En cuanto a la configuración de las interfaces de red, en nuestro caso, el uso de una sola interfaz será suficiente para soportar el tráfico de los servicios de OpenStack y las instancias lanzadas en nuestro nodo de cómputo. Por tanto, seguiremos el diseño proporcionado en la documentación de OSA [8] basado en la configuración de una única interfaz en cada host:

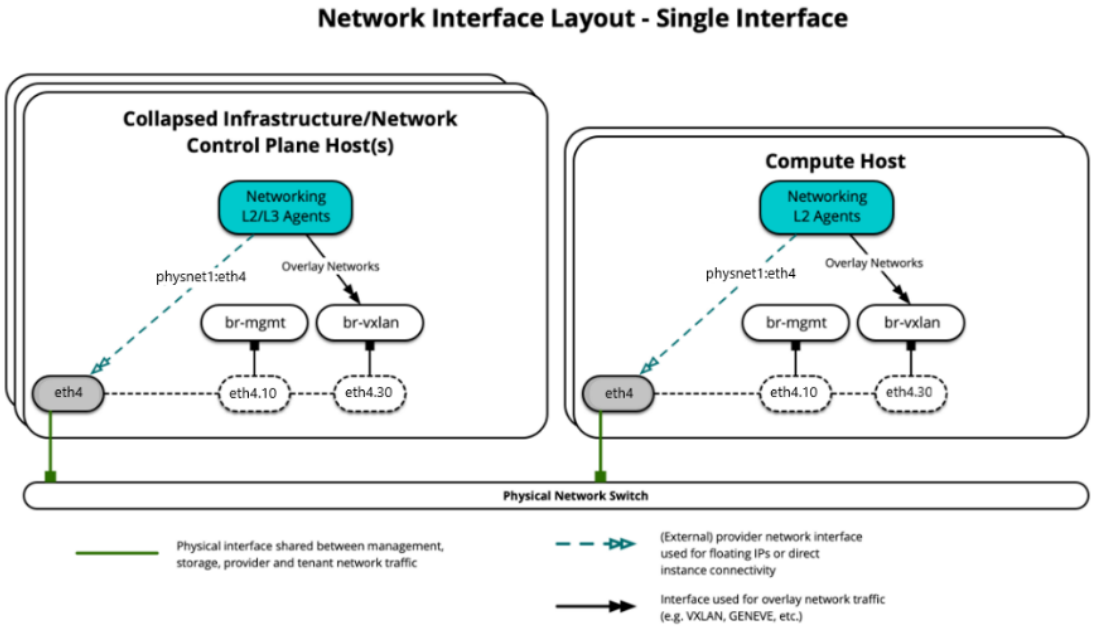

Ilustración 3 Configuración de red [8]

Como podemos observar en el diagrama, el peso de toda la carga de trabajo recae sobre la interfaz física eth4 de ambos nodos, que está sujeta a los puentes *br-mgmt* y *br-vxlan* según sus correspondientes VLANs. Sin embargo, en nuestro entorno, utilizaremos también la interfaz física eth1, la cual tendrá asignada una dirección del laboratorio del departamento de automática, donde se encuentran físicamente los ordenadores. Lo haremos así porque el rango de direcciones del laboratorio que tenemos a nuestra disposición para realizar este proyecto no es suficiente como para desplegar OSA sobre él. Concretamente contamos con 20 IPs pero, entre las direcciones que necesita el despliegue para configurar los contenedores y las que emplearemos más adelante para asignárselas a los dispositivos albergados en nuestra nube, este rango se nos queda muy corto. Este es el motivo por el que, en el apartado anterior, elegimos los rangos 10.0.120.0/22 y 10.0.124.0/22, ya que son rangos de direcciones privadas que no se entrometen en el del laboratorio. Por lo tanto, en nuestro entorno, realmente la salida a la red exterior se realiza por la interfaz eth1.

A partir del diagrama podremos comprender mejor la configuración de red de los archivos **/etc/network/interfaces** que utilizaremos para cada host (Ver anexo). En ellos, hemos configurado los puentes requeridos por OSA según el direccionamiento elegido anteriormente.

Ambos ficheros son muy parecidos y se apoyan en el fichero de configuración de red [7] del entorno de prueba. Un resumen de estos:

- Ambos hosts tienen asignada a la interfaz eth1 una dirección del laboratorio (infra1: 172.29.21.201 y compute1: 172.29.21.200), cada uno con salida a internet a partir de un router de la Universidad ubicado en la dirección 172.29.20.1.

- Encontramos declarados el puente de gestión y el puente VXLAN, con las direcciones IP que hemos elegido para ellos, así como las VLANs a las que están sujetas.
- También se ha declarado el puente br-vlan que, mediante la interfaz virtual eth12 que hemos creado, conectará nuestra red física con la de OpenStack.
- Adicionalmente, el puente de gestión del nodo de control cuenta con la dirección virtual 10.0.123.10, que posteriormente OSA utilizará para identificar el lado público de su balanceador de carga. Las funciones principales de esta dirección IP una vez completada la instalación, serán las de acceder a la API externa y a las interfaces web.

3.1.4. Configuración del despliegue

Una vez hayamos elegido el entorno que mejor se ajusta a nuestras necesidades y configurado la red sobre la que vamos a desplegarlo, podemos pasar a configurar las características del mismo. Todos los *playbooks* de Ansible [9] que se ocupan del proceso de instalación dependen de ciertos archivos de configuración, cuyas tareas se encargan de definir el entorno de destino. Básicamente se trata de tres archivos, y sus directivas de configuración (algunas obligatorias y otras opcionales) deben ser revisadas y modificadas antes de ejecutar los *playbooks*. Estos son:

- **openstack_user_config.yml**: se encarga de configurar la red de los hosts destino, así como de instalar el software correspondiente a cada uno. Más específicamente, indica en qué hosts se ejecutan los contenedores y servicios desplegados por OpenStack-Ansible, y define las interfaces y redes puente necesarias para relacionar las redes virtuales de la nube *OpenStack* y las físicas de dichos hosts.
- **user_variables.yml**: define opciones de despliegue globales y específicas para cada rol. Podremos elegir, por ejemplo, el método de instalación de los servicios de *OpenStack* (utilizando paquetes PIP o de distribución) u opciones de depuración.
- **user_secrets.yml**: contiene las contraseñas (generadas antes de ejecutar el despliegue) de todos los servicios.

3.1.5. Playbooks

Ahora sí estaremos preparados para iniciar el proceso de despliegue de OSA, el cual requiere la ejecución de tres *playbooks* [10] principales de *Ansible*:

- **setup-hosts.yml**: paso previo a la instalación de los servicios de *OpenStack* y de infraestructura, ya que su función es la de preparar los hosts de destino para dichas instalaciones. Básicamente se encarga de construir, o reiniciar, los contenedores (donde estarán albergados los servicios) en los hosts de destino, así como de instalar componentes comunes necesarios para dichos contenedores.
- **setup-infrastructure.yml**: instala los servicios de infraestructura: *Memcached*, el servidor de repositorios, *Galera*, *RabbitMQ* y *rsyslog*.
- **setup-openstack.yml**: *playbook* que se ocupa de instalar los servicios de *OpenStack*: Identity (keystone), Image (glance), Block Storage (cinder), Compute (nova), Networking (neutron), etc.

Cómo alternativa se puede usar el *playbook* "setup-everything.yml", el cual abarca todas las acciones de los tres *playbooks* citados.

3.2. Despliegue

Teniendo claro los conceptos básicos de OpenStack-Ansible, así como la configuración de red sobre la que vamos a realizar la instalación, estamos listos para preparar y ejecutar el despliegue. Para este cometido, la guía de despliegue oficial de OSA [11] distingue entre dos tipos de hosts:

- **Host de despliegue:** encargado de ejecutar los *playbooks* de Ansible.
- **Hosts destino:** nodos en los que OSA instalará los servicios de OpenStack y de infraestructura.

Recordemos que nuestro host de infraestructura, aparte de ocuparse de la tarea del despliegue, también contendrá servicios de OpenStack y de infraestructura. Por tanto, dicho nodo se podría considerar a su vez un host de destino. Es por ello por lo que, para este apartado, nos referiremos a él simplemente como host de despliegue, mientras que el host de cómputo conservará el mismo nombre.

3.2.1. Medios

Para la realización de este despliegue contamos con los siguientes dispositivos y características:

- Dos ordenadores Intel Core i7-9700K, 32GB RAM, 8 vCPUs
- Switch Netgear GS716T

Ambos ordenadores cuentan con el sistema operativo Ubuntu Server 20.04 LTS. El nombre del equipo del host de despliegue es “control”, mientras que el del host de cómputo es “ansible”, y comparten el mismo nombre de usuario, mi nombre.

3.2.2. Configuración del Switch

Antes de pasar a la preparación de los nodos, configuraremos dos puertos del Switch que tenemos a nuestra disposición con las VLANs necesarias. El manual del Switch Netgear GS716T [12] nos indica que podemos acceder al dispositivo mediante un navegador web con la dirección IP por defecto del conmutador (192.168.0.239). Para ello, primero debemos conectar el *switch* a la red de nuestro puesto de trabajo y al equipo desde el que vamos a administrar el conmutador y, desde este, configurar una IP en la misma subred que la dirección IP por defecto (en este caso 192.168.0.0/24):

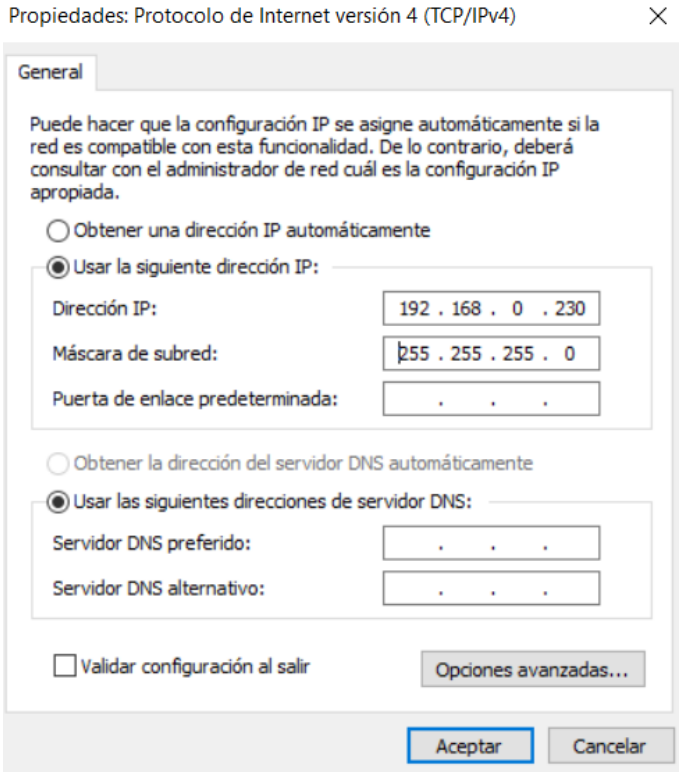

Ilustración 4 Configuración de IP en la subred del Switch

Una vez hecho esto, podremos acceder a la interfaz web introduciendo en el navegador la dirección IP por defecto del conmutador y la contraseña “password”. Dentro de la interfaz web del Switch, accedemos al apartado “VLAN” y seleccionamos la opción del estándar “IEEE 802.1Q VLAN”. Seguido, pinchamos en el desplegable “VLAN Management” y seleccionamos la opción “Add new VLAN”, ingresamos el identificador de la VLAN a configurar en el apartado “VLAN ID”, elegimos los puertos que recibirán dicha configuración pinchando sobre ellos y, por último, confirmamos los cambios seleccionando “Apply”. En nuestro conmutador utilizaremos los puertos 5 y 6 para las etiquetas VLAN 10 y 30, referentes a la red de gestión y a la red VXLAN respectivamente:

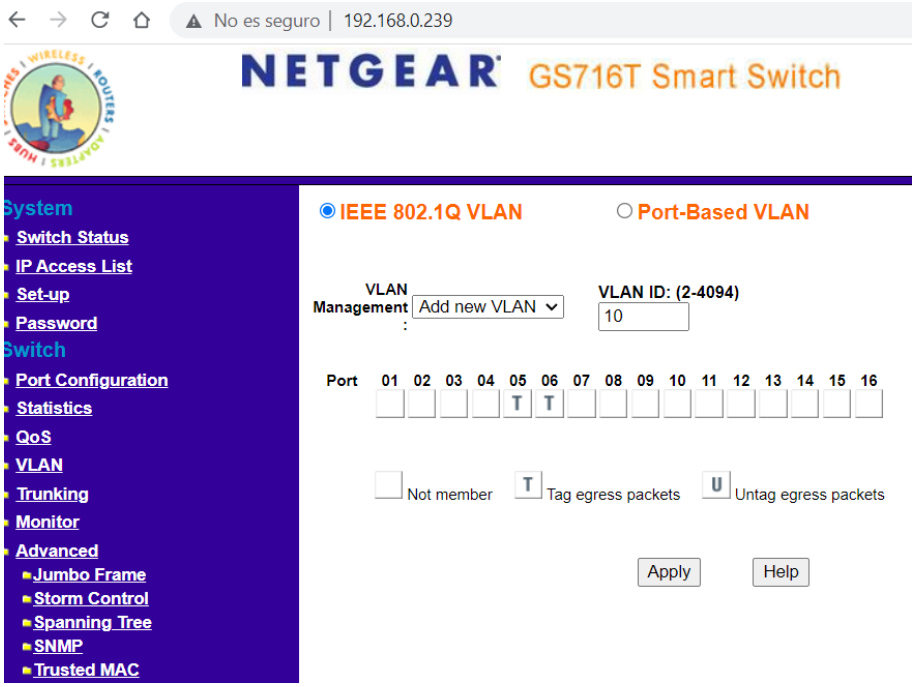

Ilustración 5 Configuración del Switch (1)

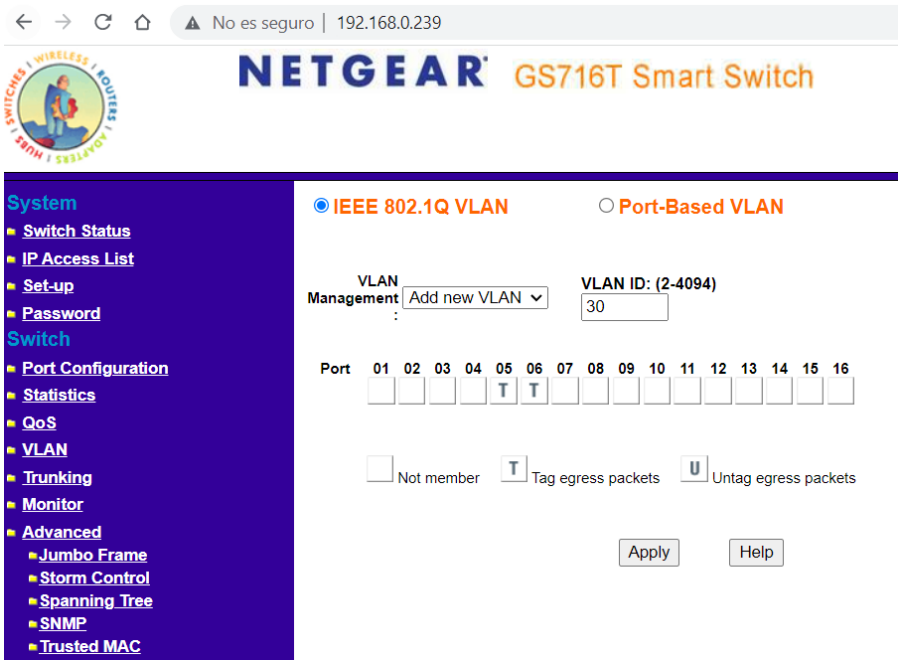

Ilustración 6 Configuración del Switch (2)

Cabe la posibilidad de que al conmutador se le haya asignado dinámicamente otra IP a través de DHCP, y que no podamos acceder a la interfaz web a partir de la dirección IP por defecto que nos indica la documentación. En caso de que no seamos capaces de identificar la nueva dirección, y asegurándonos previamente de que no vamos a interferir en la configuración del Switch de otra persona, podemos reiniciar de fábrica el conmutador. De esta forma, se reiniciará la dirección 192.168.0.239 en el dispositivo.

3.2.3. Preparar el host de despliegue

Ya estamos listos para comenzar con las tareas de preparación del despliegue. Comenzaremos con el nodo de despliegue, instalando algunos paquetes de software adicionales:

1. Actualizamos las listas de paquetes disponibles en las fuentes definidas en el sistema:

```
$ sudo apt update
```

2. Actualizamos los paquetes del sistema y el kernel:

```
$ sudo apt dist-upgrade
```

3. Reiniciamos el sistema.
4. Procedemos a instalar los siguientes paquetes de software:

```
$ sudo apt install bridge-utils build-essential git openssh-server python3-dev sudo
```

Configurar claves SSH

A parte de ser una herramienta fundamental para que Ansible pueda conectarse a los hosts del entorno y realizar el proceso de instalación, también utilizaremos SSH para conectarnos a ambos nodos con nuestro ordenador personal (a partir de una dirección IP del laboratorio que nos asigna la Universidad) mediante la aplicación *MobaXterm*. Trabajar desde este programa nos facilitará tareas como la transferencia de archivos entre los nodos y nuestro ordenador personal, acceso a varios monitores a la vez y creación de túneles SSH, entre otras.

De momento en este apartado nos ocuparemos simplemente de crear un par de claves en el host de despliegue. Para ello, lanzamos el comando “ssh-keygen” y seguido pulsamos “Enter” para elegir el subdirectorio “.ssh/” del directorio personal como lugar para almacenar el par de claves (**id_rsa** e **id_rsa.pub**). Por último, siguiendo las recomendaciones de la documentación, no les asignaremos una contraseña, en vistas a reducir la interacción del usuario durante las operaciones de Ansible.

Configurar la red

En este paso simplemente tendremos que copiar el contenido del fichero que preparamos en pasos anteriores para la red del host de despliegue en **/etc/network/interfaces**.

Instalar el código fuente y las dependencias

Primero, atendiendo a la documentación oficial del despliegue, en la configuración del nodo infraestructura [13], debemos clonar en el directorio **/opt/openstack-ansible** una versión estable del repositorio Git de OpenStack-Ansible:

```
$ sudo git clone -b 24.0.1 https://opendev.org/openstack/openstack-ansible /opt/openstack-ansible
```

A continuación, nos ubicamos en el directorio **/opt/openstack-ansible**, y ejecutamos el script **bootstrap-ansible.sh**:

```
$ sudo scripts/bootstrap-ansible.sh
```

Este script instala Ansible, descarga los *playbooks* utilizados para el despliegue, prepara el host con todos los roles de Ansible que componen el entorno OpenStack y crea la herramienta openstack-ansible, que proporciona al sistema las variables necesarias para que coincidan con las de los ficheros **/etc/openstack_deploy/user_*.yml** como argumentos en los *playbooks* de Ansible.

3.2.4. Preparar el host de cómputo

En el nodo de cómputo debemos actualizar también los paquetes del sistema y el kernel:

```
$ sudo apt update

$ sudo apt dist-upgrade
```

Reiniciamos el sistema e instalamos los paquetes de software adicionales:

```
$ sudo apt install bridge-utils debootstrap openssh-server \

 tcpdump vlan python3
```

Configurar claves SSH

En este punto vamos a terminar la configuración de las claves SSH para poder conectarnos de forma remota al host de cómputo desde el de despliegue. Para ello, tenemos que copiar la clave pública (**id_rsa.pub**), albergada en el nodo de despliegue, al nodo de cómputo. La forma más rápida de hacerlo es mediante el comando "ssh-copy-id". Para utilizarlo, simplemente tendremos que indicar el nombre de usuario destino y la dirección del host destino:

```
$ sudo ssh-copy-id silviu@172.29.21.200
```

El comando "ssh-copy-id" copia las claves públicas SSH en "~/.ssh", pero OSA utilizará el directorio "/root/.ssh". Por tanto, en el nodo de cómputo, moveremos directamente la clave de "~/.ssh" a dicha dirección:

```
$ sudo su

$ cat .ssh/authorized_keys >> /root/.ssh/authorized_keys
```

Configurar la red

Al igual que en el apartado anterior, copiamos el contenido del fichero de configuración de la red del host de cómputo en **/etc/network/interfaces**.

3.2.5. Configuración inicial del despliegue

Este es el momento de revisar y modificar los archivos de configuración referentes al entorno sobre el que vamos a realizar la instalación:

1. En el host de despliegue, copiamos el contenido del directorio **/opt/openstack-ansible/etc/openstack_deploy** al directorio **/etc/openstack_deploy**:

   ```
   $ sudo cp -r /opt/openstack-ansible/etc/openstack_deploy/. /etc/openstack_deploy
   ```

2. Nos ubicamos en **/etc/openstack_deploy**.

3. Copiamos el contenido del fichero de ejemplo **openstack_user_config.yml.example** a **/etc/openstack_deploy/openstack_user_config.yml**:

```
$ sudo cp openstack_user_config.yml.example openstack_user_config.yml
```

4. Revisamos y modificamos el archivo **openstack_user_config.yml** acorde a la configuración que queremos para nuestro despliegue (Ver anexo):

    - En la sección **cidr_networks** indicamos los rangos de direcciones IP asociados a las redes de gestión (container) y VXLAN (tunnel).
    - Completamos la sección **used_ips** con un listado de direcciones IP en uso de los rangos definidos en la sección anterior. De esta forma, evitamos conflictos durante procesos de generación automática de direcciones IP.
    - En **global_overrides** configuraremos el balanceador de carga. Aquí indicaremos que la dirección 10.0.123.10 se utilizará como IP en el lado público del balanceador (**external_lb_vip_address**), mientras que la IP 10.0.123.11 hace referencia a la dirección interna del balanceador (**internal_lb_vip_address**). También debemos indicar el nombre del puente de gestión en la subsección **management_bridge**.
    - Aún dentro de **global_overrides**, debemos configurar las redes de los contenedores LXC en la subsección **provider_networks**. En este caso, se deben configurar la red de gestión, VXLAN y las redes "flat" (no etiquetadas) y VLAN (etiquetadas). Los significados de algunos de los puntos de esta sección son:
        - **type**: indica el tipo de red que vamos a declarar ('raw', 'vlan', 'vxlan' o 'flat').
        - **container_bridge**: nombre del puente a utilizar para esta red en los hosts de destino.
        - **ip_from_q**: nombre de la red de la sección **cidr_networks** que indica el rango de direcciones IP a utilizar.
        - **range**: rango de identificadores VXLAN o de etiquetas VLAN, dependiendo de si el tipo de red es 'vxlan' o 'vlan'.
        - **net_name**: nombre de la red para los tipos 'flat' o 'vlan'.
        - **container_type**: mecanismo que conecta las interfaces de los contenedores con el puente en los hosts de destino para esta red (normalmente "veth").
        - **container_interface**: la interfaz del contenedor a utilizar para la conexión.
        - **host_bind_override**: interfaz que se utilizará para conectar el puente "br-vlan" de nuestra red física con el de la nube OpenStack. Aquí es dónde utilizaremos la interfaz virtual eth12 que declaramos en el fichero de configuración de red.
        - **group_binds**: lista de grupos de Ansible que contienen esta red.
    - Por último, indicamos en qué hosts se desplegarán los servicios de infraestructura y OpenStack. Debemos indicar el nombre de la sección que hace

referencia al servicio, o grupo de servicios, que queremos instalar, seguido del nombre y la dirección IP del host destino. Algunos ejemplos:

- **shared-infra_hosts**: despliega servicios de infraestructura (el clúster de base de datos Galera SQL, RabbitMQ y Memcached).
- **identity_hosts**: hace referencia al servicio de Keystone.
- **network_hosts**: despliega los servicios de Neutron.
- **compute_hosts**: despliega el servicio de computación Nova. Esta será la única sección que se desplegará en el nodo de cómputo.

- Adicionalmente, en caso de requerir más servicios para nuestro entorno, en los ficheros ubicados en **etc/openstack_deploy/conf.d** podemos encontrar ejemplos de todos los que podemos utilizar.

5. En el archivo **user_variables.yml** (ver anexo) simplemente activaremos la opción de depuración y detalles "debug", y dejaremos la opción por defecto "source" como método de instalación. Dicho método especifica que los servicios OpenStack se instalen por medio de paquetes PIP.

Cabe destacar que mantenemos una copia de seguridad de **/etc/openstack_deploy/** en el repositorio https://github.com/sofrone7/openstack-ansible, en caso de que sea necesario restaurar dicho directorio.

Creación de las credenciales de los servicios

Como comentábamos en puntos anteriores, antes de iniciar el despliegue tenemos que generar las contraseñas de los servicios. Para ello, nos situamos en **/opt/openstack-ansible** y lanzamos el script **pw-token-gen.py**. Este paso creará credenciales aleatorias para cada uno de los servicios de infraestructura y OpenStack en el archivo que deseemos, en este caso **/etc/openstack_deploy/user_secrets.yml**:

```
$ sudo ./scripts/pw-token-gen.py --file /etc/openstack_deploy/user_secrets.yml
```

3.2.6. Ejecutar el despliegue

Comprobación de la integridad de los *playbooks*

1. Verificar la sintaxis de los archivos YAML del paso anterior que configuramos en el directorio **/etc/openstack_deploy**. Para ello, haremos uso del programa en línea YAML Lint (http://www.yamllint.com/). Su funcionamiento es muy sencillo, simplemente tenemos que copiar el contenido de los ficheros, pinchamos en "Go" y la página nos indicará si la sintaxis es válida o no.
2. Nos dirigimos a **/opt/openstack-ansible/playbooks** y comprobamos la sintaxis de los tres *playbooks* a ejecutar mediante los siguientes comandos:

```
silviu@control:/opt/openstack-ansible/playbooks$ sudo openstack-ansible setup-inf
rastructure.yml --syntax-check
Variable files: "-e @/etc/openstack_deploy/user_secrets.yml -e @/etc/openstack_de
ploy/user_variables.yml "
[WARNING]: Unable to parse /etc/openstack_deploy/inventory.ini as an inventory
source

playbook: setup-infrastructure.yml

EXIT NOTICE [Playbook execution success] ***********************************
==============================================================================
silviu@control:/opt/openstack-ansible/playbooks$ sudo openstack-ansible setup-hos
ts.yml --syntax-check
Variable files: "-e @/etc/openstack_deploy/user_secrets.yml -e @/etc/openstack_de
ploy/user_variables.yml "
[WARNING]: Unable to parse /etc/openstack_deploy/inventory.ini as an inventory
source
[WARNING]: Could not match supplied host pattern, ignoring: all_lxc_containers

playbook: setup-hosts.yml

EXIT NOTICE [Playbook execution success] ***********************************
==============================================================================
silviu@control:/opt/openstack-ansible/playbooks$ sudo openstack-ansible setup-ope
nstack.yml --syntax-check
Variable files: "-e @/etc/openstack_deploy/user_secrets.yml -e @/etc/openstack_de
ploy/user_variables.yml "
[WARNING]: Unable to parse /etc/openstack_deploy/inventory.ini as an inventory
source
[WARNING]: Could not match supplied host pattern, ignoring: zun

playbook: setup-openstack.yml

EXIT NOTICE [Playbook execution success] ***********************************
==============================================================================
```

Ilustración 7 Verificación de sintaxis de los playbooks

Ejecutar los *playbooks*

Ya tenemos todo listo para lanzar los *playbooks* de OSA y dar comienzo al proceso de instalación de nuestra propia nube privada OpenStack. Nos ubicamos en el directorio **/opt/openstack-ansible/playbooks** y seguimos el siguiente orden de ejecución:

1. En primer lugar, ejecutamos el *playbook* de configuración de los hosts:

   ```
   $ sudo openstack-ansible setup-hosts.yml
   ```

   Confirmamos que la ejecución del *playbook* se ha completado satisfactoriamente con cero elementos inalcanzables o fallidos:

```
PLAY RECAP *********************************************************************
compute1                   : ok=125  changed=25   unreachable=0    failed=0    sk
ipped=31   rescued=0    ignored=0
infra1                     : ok=190  changed=39   unreachable=0    failed=0    sk
ipped=34   rescued=0    ignored=0
infra1_cinder_api_container-c2ebafd7 : ok=90   changed=48   unreachable=0    fail
ed=0    skipped=6    rescued=0    ignored=0
infra1_galera_container-acda48b8 : ok=90   changed=48   unreachable=0    failed=0
    skipped=6    rescued=0    ignored=0
infra1_glance_container-6e6a676c : ok=90   changed=48   unreachable=0    failed=0
    skipped=6    rescued=0    ignored=0
infra1_heat_api_container-06b1e95d : ok=90   changed=48   unreachable=0    failed
=0    skipped=6    rescued=0    ignored=0
infra1_horizon_container-f75d71f3 : ok=90   changed=48   unreachable=0    failed=
0    skipped=6    rescued=0    ignored=0
infra1_keystone_container-05a6c5bf : ok=90   changed=48   unreachable=0    failed
=0    skipped=6    rescued=0    ignored=0
infra1_memcached_container-510bf1b5 : ok=90   changed=48   unreachable=0    faile
d=0    skipped=6    rescued=0    ignored=0
infra1_neutron_server_container-08bd30d2 : ok=93   changed=48   unreachable=0
failed=0    skipped=6    rescued=0    ignored=0
infra1_nova_api_container-7fc29f55 : ok=90   changed=48   unreachable=0    failed
=0    skipped=6    rescued=0    ignored=0
infra1_placement_container-974e44a1 : ok=90   changed=48   unreachable=0    faile
d=0    skipped=6    rescued=0    ignored=0
infra1_rabbit_mq_container-bbca74ae : ok=90   changed=48   unreachable=0    faile
d=0    skipped=6    rescued=0    ignored=0
infra1_repo_container-260b6ab8 : ok=90   changed=48   unreachable=0    failed=0
  skipped=6    rescued=0    ignored=0
infra1_utility_container-23297593 : ok=90   changed=48   unreachable=0    failed=
0    skipped=6    rescued=0    ignored=0
localhost                  : ok=15   changed=0    unreachable=0    failed=0    sk
ipped=13   rescued=0    ignored=0



EXIT NOTICE [Playbook execution success] ***********************************
==============================================================================
```

Ilustración 8 Salida de la ejecución del playbook setup-hosts.yml

2. Lanzamos el segundo *playbook*, en este caso el referente a la configuración de la infraestructura:

```
$ sudo openstack-ansible setup-infrastructure.yml
```

Ilustración 9 salida de la ejecución del playbook setup-infrastrcuture.yml

3. Por último, ejecutamos el *playbook* de configuración de OpenStack:

```
$ sudo openstack-ansible setup-openstack.yml
```

Ilustración 10 Salida de la ejecución del playbook setup-openstack.yml

4. Operaciones en la nube OSA

4.1. Pruebas iniciales del funcionamiento de OpenStack

Una vez terminado el proceso de instalación sin errores, la documentación de OpenStack-Ansible nos recomienda realizar dos comprobaciones para verificar el correcto funcionamiento de la nube.

Verificación de la API

Si listamos los contenedores existentes en el host de despliegue, podremos observar todos que contienen los servicios de OpenStack e infraestructura desplegados por los *playbooks* **setup-infrastructure.yml** y **setup-openstack.yml**:

```
silviu@control:~$ sudo lxc-ls
[sudo] password for silviu:
infra1_cinder_api_container-c2ebafd7     infra1_galera_container-acda48b8         infra1_glance_container-6e6a676c
infra1_heat_api_container-06b1e95d       infra1_horizon_container-f75d71f3        infra1_keystone_container-05a6c5bf
infra1_memcached_container-510bf1b5      infra1_neutron_server_container-08bd30d2 infra1_nova_api_container-7fc29f55
infra1_placement_container-974e44a1      infra1_rabbit_mq_container-bbca74ae      infra1_repo_container-260b6ab8
infra1_utility_container-23297593        ubuntu-focal-amd64
```

Ilustración 11 Listado de contenedores

Para comprobar el funcionamiento de la API accederemos al contenedor de "utilidades". Este contenedor proporciona un entorno CLI para la configuración de la nube y la realización de pruebas. Podemos identificarlo fácilmente haciendo uso del comando "grep":

```
silviu@control:/opt/openstack-ansible/playbooks$ sudo lxc-ls | grep utility
infra1_utility_container-23297593        ubuntu-focal-amd64
```

Ilustración 12 Localización del contenedor de utilidades

Accedemos a él con el siguiente comando:

```
silviu@control:~$ sudo lxc-attach -n infra1_utility_container-23297593
[sudo] password for silviu:
root@infra1-utility-container-23297593:/#
```

Ilustración 13 Acceso al contenedor de utilidades

Una vez dentro del contenedor, podremos encontrar un archivo **openrc** en el directorio raíz. Se trata de un pequeño script *bash*, creado durante el despliegue de OSA, que se encarga de establecer las variables de credenciales de nuestra nube OpenStack como variables de entorno. De esta forma, podremos utilizar el sistema de línea de comandos del contenedor como el CLI de OpenStack.

Ejecutamos el script **openrc** junto a un comando de OpenStack que utilice una o más APIs como, por ejemplo, un listado de todos los usuarios de la nube:

```
root@infra1-utility-container-23297593:/# . ~/openrc
root@infra1-utility-container-23297593:/# openstack user list --os-cloud=default
+----------------------------------+--------------------+
| ID                               | Name               |
+----------------------------------+--------------------+
| 90093a3b2cbe451fb471e0678bbb928d | admin              |
| dff7871426a44ec9852b5600160f0fbf | placement          |
| 47873c6545b1494bb93950d71a31bb48 | glance             |
| 183251e79bf44561a51f9735ce469551 | cinder             |
| 9695a626bc7643c6b4548a244f37f957 | nova               |
| df9bcb2f2f6b4211b48f40960ff0983d | neutron            |
| 56bec37e196c449c872b479e01f71d47 | heat               |
| 75c20e90a9624f37b85702e1d3707dd4 | stack_domain_admin |
```

Ilustración 14 Listado de los usuarios de la nube OSA

O el comando **openstack endpoint list**, que muestra un listado de las URLs en las que la API recibe solicitudes de los distintos servicios de OpenStack:

```
root@infra1-utility-container-23297593:/# openstack endpoint list
+----------------------------------+----------+--------------+---------------+---------+-----------+-----------------------------------------------+
| ID                               | Region   | Service Name | Service Type  | Enabled | Interface | URL                                           |
+----------------------------------+----------+--------------+---------------+---------+-----------+-----------------------------------------------+
| 15d1e2e219bc4c4285ec64da4dd2a4e6 | RegionOne | glance       | image         | True    | admin     | http://10.0.123.11:9292                       |
| 1fc8c07705174557ab6d2c7097ef5036 | RegionOne | placement    | placement     | True    | admin     | http://10.0.123.11:8780                       |
| 3b33e57b25604832a0abc2687c256c0d | RegionOne | heat-cfn     | cloudformation | True    | internal  | http://10.0.123.11:8000/v1                    |
| 4bfac02a1b0f4cbaa142b06aabec81c8 | RegionOne | neutron      | network       | True    | admin     | http://10.0.123.11:9696                       |
| 4c21fc5907be486ebb1543246e4133ed | RegionOne | placement    | placement     | True    | internal  | http://10.0.123.11:8780                       |
| 5c5e829463314a31993416fa3350092e | RegionOne | glance       | image         | True    | internal  | http://10.0.123.11:9292                       |
| 7491541e20744ce49994886df58bc680 | RegionOne | heat         | orchestration | True    | public    | https://10.0.123.10:8004/v1/%(tenant_id)s     |
| 75159d4bc9dc419caf023582097a13e9 | RegionOne | heat-cfn     | cloudformation | True    | admin     | http://10.0.123.11:8000/v1                    |
| 77b00de249564a52b265454e082f07fe | RegionOne | cinderv3     | volumev3      | True    | admin     | http://10.0.123.11:8776/v3/%(tenant_id)s      |
| 7c6415ad9ff147e0bce097e5fd3ae598 | RegionOne | heat-cfn     | cloudformation | True    | public    | https://10.0.123.10:8000/v1                   |
| 7e1f9d39f33a4a29a684ea8d8ded528a | RegionOne | placement    | placement     | True    | public    | https://10.0.123.10:8780                      |
| 85566adcfe3f4d359a38e4f5dee935f6 | RegionOne | cinderv3     | volumev3      | True    | public    | https://10.0.123.10:8776/v3/%(tenant_id)s     |
| 8c80a5ec9b724487adabb874c350e69e | RegionOne | keystone     | identity      | True    | internal  | http://10.0.123.11:5000                       |
| 9622ef53f47a4ddb971a9b30bd554d0d | RegionOne | heat         | orchestration | True    | internal  | http://10.0.123.11:8004/v1/%(tenant_id)s      |
| 98d9b71c0ef9493a8f2d1004090d75e4 | RegionOne | keystone     | identity      | True    | admin     | http://10.0.123.11:5000                       |
| c52bcf51e2a3428fbe8b5400645da56e | RegionOne | glance       | image         | True    | public    | https://10.0.123.10:9292                      |
| cd22cdbcd52347a1a6d3c356a5137464 | RegionOne | cinderv3     | volumev3      | True    | internal  | http://10.0.123.11:8776/v3/%(tenant_id)s      |
| d0017073d4ed48b5b4ca5c69dba9939f | RegionOne | neutron      | network       | True    | public    | https://10.0.123.10:9696                      |
| d8463c89c7e146658ce38bc7ddbda59f | RegionOne | heat         | orchestration | True    | admin     | http://10.0.123.11:8004/v1/%(tenant_id)s      |
| ebd6af51d65e461fb32e4d0fe0a16bce | RegionOne | neutron      | network       | True    | internal  | http://10.0.123.11:9696                       |
| ed69db9c4d29408aa0c176457375013f | RegionOne | nova         | compute       | True    | internal  | http://10.0.123.11:8774/v2.1                  |
| f7703d55065c40d5b2631c9bfe12573d | RegionOne | nova         | compute       | True    | admin     | http://10.0.123.11:8774/v2.1                  |
| fac4a7f544ca4ec1ba6854f88fcbd7e1 | RegionOne | nova         | compute       | True    | public    | https://10.0.123.10:8774/v2.1                 |
| fc55f3f05788470f93825edba69aad8a | RegionOne | keystone     | identity      | True    | public    | https://10.0.123.10:5000                      |
+----------------------------------+----------+--------------+---------------+---------+-----------+-----------------------------------------------+
```

Ilustración 15 Listado de los endpoints de la nube OSA

Observamos que todos los *endpoints* están habilitados.

Por último, para comprobar que el host de cómputo (en la dirección 10.0.123.12) se ha integrado correctamente con el entorno, podemos verificar que se identifica como un nodo de computación válido al ejecutar el comando **openstack hypervisor list**:

```
root@infra1-utility-container-23297593:/# openstack hypervisor list
+----+---------------------+-----------------+-------------+-------+
| ID | Hypervisor Hostname | Hypervisor Type | Host IP     | State |
+----+---------------------+-----------------+-------------+-------+
|  1 | ansible             | QEMU            | 10.0.123.12 | up    |
+----+---------------------+-----------------+-------------+-------+
```

Ilustración 16 Listado de los nodos de computación de la nube OSA

Acceso al servicio Horizon

El acceso al cuadro de mandos Horizon se realiza a través de la dirección IP externa del balanceador de carga. Debemos recordar que dicha dirección fue definida en la opción **external_lb_vip_address** (10.0.123.10) del archivo **openstack_user_config.yml**.

Dado que el host de despliegue cuenta con un Ubuntu Server, emplearemos nuestro ordenador personal para entrar al *Dashboard* desde su navegador web. Sin embargo, nuestra máquina no puede conectarse a la dirección 10.0.123.10 del host de despliegue, ya que sólo podemos llegar a la red 172.29.20.0/22 del laboratorio. Para conseguirlo, haremos uso de la opción de MobaXterm que nos permite crear túneles SSH.

El plan consiste en establecer una conexión SSH desde mi máquina local al host remoto (host de despliegue), donde se encuentra el servicio OpenStack Horizon. De esta forma, podremos acceder a este servicio desde el ordenador personal utilizando la dirección local (localhost) haciendo referencia al puerto mapeado a través de la conexión SSH. Para ello, dentro de MobaXterm, en la pestaña “Tools” pinchamos en “MobaSSHTunnel -> New SSH tunnel”. Seleccionamos “Local port forwarding” y observamos los siguientes campos a completar:

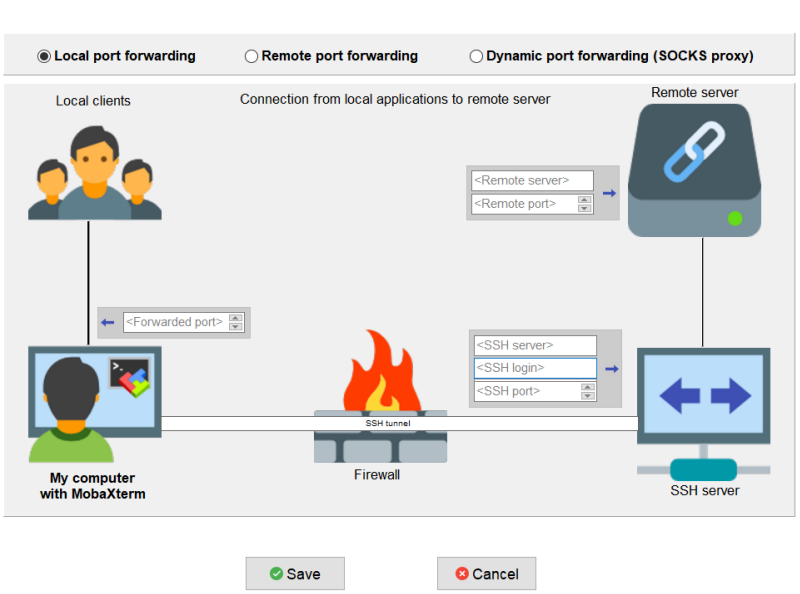

Ilustración 17 Campos del túnel SHH

- **Remote server:** la dirección IP por la que se accede a Horizon en el host remoto (10.0.123.10).
- **Remote port:** el puerto de Horizon en el host remoto (el *Dashboard* utiliza HTTPS en el puerto 443).
- **SSH server:** la dirección IP del laboratorio del host remoto a la que vamos a hacer SSH (172.29.21.201).
- **SSH login:** el nombre de usuario del host remoto (silviu).
- **SSH port:** el puerto SSH a utilizar (22).
- **Forwarded port:** el puerto que utilizaremos para acceder a Horizon desde el navegador de nuestra máquina local (443).

La configuración del túnel quedaría así:

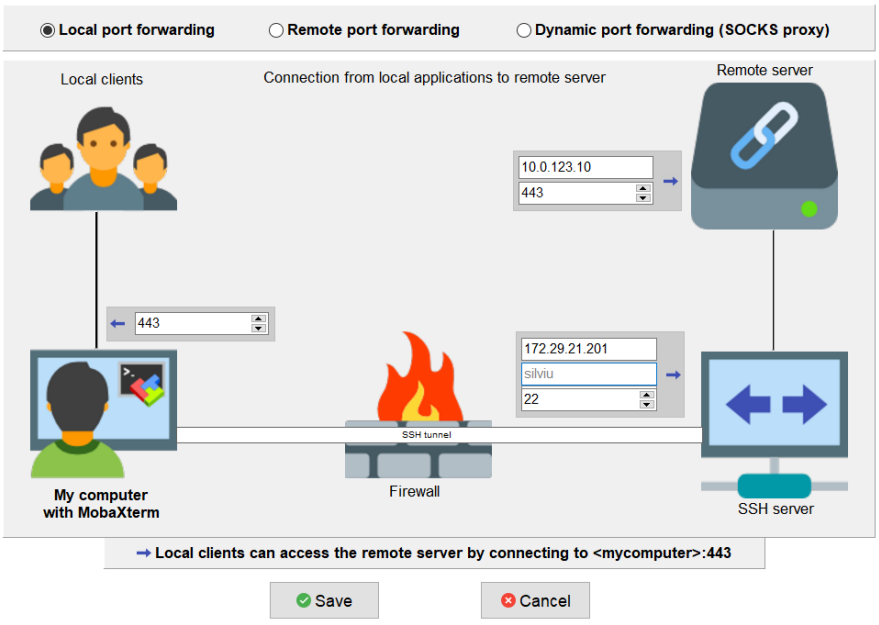

Ilustración 18 Configuración del túnel SSH

Al guardar los cambios podremos observar que un nuevo túnel SSH ha sido creado:

Name	Type	Start/stop	Forward port	Destination server	SSH server
	Local		443	10.0.123.10:443	silviu@172.29.21.201:22

Ilustración 19 Resultado del túnel SSH

Para activar el túnel simplemente tendremos que inicializarlo con la opción “Start” y podremos acceder a Horizon desde el navegador con la dirección https://localhost:443:

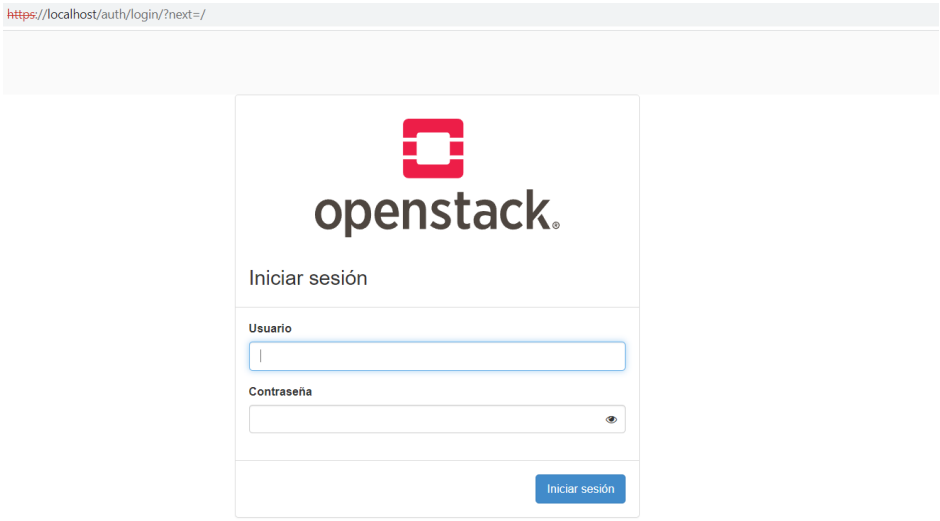

Ilustración 20 Acceso a Horizon

Nos autentificaremos con el nombre de usuario **admin** y la contraseña definida por la opción **keystone_auth_admin_password** en el fichero **user_secrets.yml**. Una forma sencilla de obtener la contraseña mencionada:

```
silviu@control:~$ sudo cat /etc/openstack_deploy/user_secrets.yml | grep keystone
_auth_admin_password
[sudo] password for silviu:
keystone_auth_admin_password: 118c211ab404105ef42466bd3ac34
```

Ilustración 21 Contraseña de Keystone

Una vez dentro, la interfaz web nos muestra un resumen de los recursos de la nube:

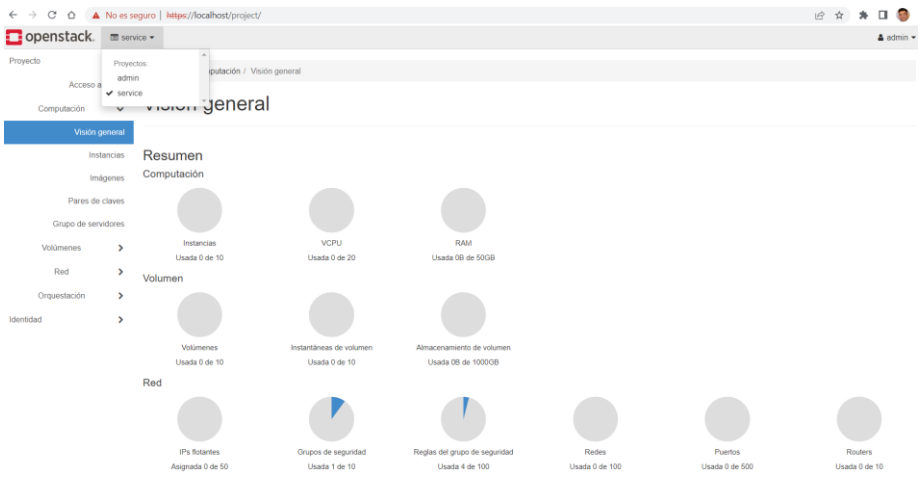

Ilustración 22 Resumen de los recursos de la nube

En la esquina superior izquierda veremos el nombre del proyecto en el que estamos ubicados. Si pinchamos sobre él veremos los dos proyectos predeterminados que OSA ha creado durante la instalación: **admin** y **service**. Seleccionamos **admin**, ya que de ahora en adelante trabajaremos con este proyecto.

4.2. Salida a Internet

Nuestra nube privada será de poca utilidad si su red se encuentra aislada. Debemos asegurarnos de que puede comunicarse con los dispositivos ubicados fuera del entorno de la nube, así como con Internet. Para ello, realizamos la tarea de crear una red virtual desde la que la nube OpenStack pueda tener salida al exterior a través de la red física del laboratorio.

Para comprender esta arquitectura debemos tener en cuenta que, según la documentación oficial [14], dentro de OpenStack se distinguen dos tipos de redes; las de proyectos (projects networks) y las de proveedores (provider networks). Sabiendo esto, estableceremos una red de proveedor conectada, por un lado, a nuestra red física mediante el puente de capa 2 *br-vlan,* configurado para este propósito, y a una red de proyecto (dónde se encuentran las instancias de la nube) a partir de un router virtual proporcionado por Neutron.

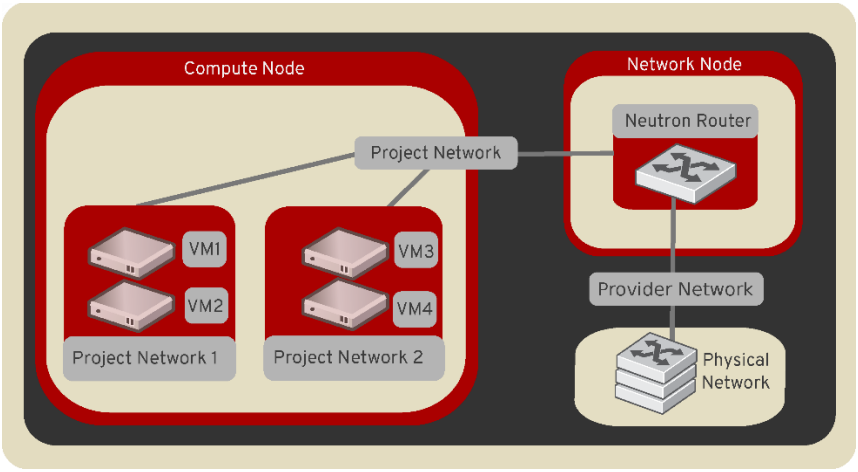

Ilustración 23 Esquema red virtual OpenStack [14]

Para concluir que este apartado funciona correctamente, haremos la prueba de comprobar que las instancias albergadas en OpenStack pueden hacer PING tanto a los ordenadores de la red del laboratorio, y viceversa, como a Internet.

4.2.1. Prerrequisitos

En primer lugar, previo a plantearnos configurar la red donde lanzaremos nuestras instancias, debemos tener en cuenta que antes de crear una máquina virtual, es necesario que el entorno cuente con ciertos parámetros [15]:

- **Fuente de arranque de la instancia:** es básicamente la plantilla que utilizaremos para crear la máquina virtual. Puede ser una imagen, una *snapshot* de una instancia existente o un volumen de almacenamiento en bloque (si está habilitado) que contenga una imagen o una *snapshot*. En nuestro caso utilizaremos una imagen, un archivo que contiene un disco virtual con un sistema operativo de arranque instalado.
- ***Flavor*:** los "sabores" de instancia definen combinaciones variables de CPU, memoria, almacenamiento y capacidad de intercambio. Con dichas combinaciones conformamos las configuraciones de hardware para nuestras máquinas según lo que necesitemos.

- **Un grupo de seguridad:** actúa como un *firewall* virtual el cual controla la conexión hacia/desde las instancias mediante reglas de grupos de seguridad, que son un conjunto de políticas de cortafuegos.
- **Un par de claves:** credenciales SSH utilizadas para acceder a las instancias creadas.

Definidos los parámetros más importantes a establecer, desde este momento realizaremos la mayor parte de la configuración mediante la CLI de OpenStack en el contenedor de "utilidades". Si bien es importante desenvolvernos tanto por línea de comandos como mediante la interfaz web, limitaremos el uso del servicio de Horizon frente a la sencillez de trabajar por terminal.

Añadir una imagen y un *flavor*

Comenzamos creando una imagen [16] Ubuntu Server 16.04 para nuestro entorno. Para ello, primero será necesario descargar la imagen de origen que deseemos de la página de descargas de Ubuntu:

```
$ wget https://cloud-images.ubuntu.com/xenial/current/xenial-server-cloudimg-amd64-disk1.img
```

Ahora, mediante el comando **openstack image create** añadimos la imagen al servicio de imágenes Glance. Indicaremos que el tamaño mínimo del disco y RAM necesario para arrancar la imagen es de 5 GB y 1024 Mb respectivamente, el formato de disco QCOW2, la visibilidad pública (**--public**) para que todos los proyectos puedan acceder a la imagen y el formato de contenedor por defecto será del tipo **bare**:

```
$ openstack image create --public --min-disk 5 --min-ram 1024 --disk-format qcow2 --property architecture=x86_64 --property hw_disk_bus=virtio --property hw_vif_model=virtio --file xenial-server-cloudimg-amd64-disk1.img "ubuntu16"
```

Para confirmar que la imagen se ha subido correctamente al listar las imágenes disponibles utilizando el comando **openstack image list**:

```
root@infra1-utility-container-23297593:/# openstack image list
+--------------------------------------+----------+--------+
| ID                                   | Name     | Status |
+--------------------------------------+----------+--------+
| 19e48025-514a-41f0-b970-a22ec716f77f | ubuntu16 | active |
+--------------------------------------+----------+--------+
```

Ilustración 24 Listado de imágenes de la nube OSA

En función de esta imagen crearemos un sabor adecuado [17]. El tamaño mínimo de disco y RAM deben coincidir con los que introducimos al crear la imagen, además, especificamos un identificador y visibilidad pública:

```
$ openstack flavor create --public ubuntu16.flavor --id auto --ram 1024 --disk 5
```

A las opciones que no hemos especificado, como el número de procesadores virtuales y el factor RXTX (indica la porción de ancho de banda que las instancias pueden emplear), se les asigna un valor por defecto. En este caso, las máquinas lanzadas con este sabor utilizarán 1 vCPU y un factor RXTX de 1.0.

```
root@infra1-utility-container-23297593:/# openstack flavor list
+------+-----------------+------+------+-----------+-------+-----------+
| ID   | Name            |  RAM | Disk | Ephemeral | VCPUs | Is Public |
+------+-----------------+------+------+-----------+-------+-----------+
| auto | ubuntu16.flavor | 1024 |    5 |         0 |     1 | True      |
+------+-----------------+------+------+-----------+-------+-----------+
```

Ilustración 25 Listado de flavors de la nube OSA

Credenciales de acceso y seguridad

Para corroborar que es posible la comunicación entre nuestra nube y el exterior, debemos ser capaces de conectarnos a una instancia albergada en OpenStack y realizar PINGs en ambas direcciones. Por tanto, es necesario que dicha instancia cuente con reglas *firewall* que permitan el acceso a ella por SSH y el tráfico de red ICMP.

Editaremos las reglas del grupo de seguridad predeterminado **default** para este propósito. Tanto el proyecto **admin** como el proyecto **service** cuentan con un grupo de seguridad denominado **default**. Podemos definir las reglas haciendo referencia directamente al nombre **default** y OpenStack identificará que nos referimos al grupo que pertenece al proyecto de administrador. Sin embargo, vamos a hacerlo referenciando su número de identificación. Lo haremos así, no necesariamente por ser estrictamente correctos, sino para ubicarnos mejor en el entorno y, así, evitar confusiones.

En primer lugar, listamos los proyectos de la nube para obtener el identificador del proyecto **admin**:

```
root@infra1-utility-container-23297593:/# openstack project list
+----------------------------------+---------+
| ID                               | Name    |
+----------------------------------+---------+
| 15fe6be059654128a9e5b92dd2f7fc85 | service |
| 406bc2348d83438aaeea1b6671a49e61 | admin   |
+----------------------------------+---------+
```

Ilustración 26 Listado de proyectos de la nube OSA

Ahora listamos los grupos de seguridad presentes en el entorno OpenStack y, atendiendo a la columna **Project**, conseguimos el identificador del grupo **default** buscado:

```
root@infra1-utility-container-23297593:/# openstack security group list
+--------------------------------------+---------+------------------------+----------------------------------+
| ID                                   | Name    | Description            | Project                          |
+--------------------------------------+---------+------------------------+----------------------------------+
| 558b5f22-d627-4159-808c-712388b5f4d2 | default | Default security group | 15fe6be059654128a9e5b92dd2f7fc85 |
| 85e168bf-3fb2-43e1-84b7-0eac3b20641b | default | Default security group | 406bc2348d83438aaeea1b6671a49e61 |
+--------------------------------------+---------+------------------------+----------------------------------+
```

Ilustración 27 Listado de los grupos de seguridad de la nube OSA

Una vez identificado el ID correcto, los comandos a introducir son [18]:

```
$ openstack security group rule create --proto icmp 85e168bf-3fb2-43e1-84b7-0eac3b20641b

$ openstack security group rule create --proto tcp --dst-port 22 85e168bf-3fb2-43e1-84b7-0eac3b20641b

$ openstack security group rule create --ethertype IPv6 --proto tcp --dst-port 22 85e168bf-3fb2-43e1-84b7-0eac3b20641b
```

Para verificar que las reglas se han añadido correctamente podemos dirigirnos a Horizon y seleccionar “Administrar reglas” en el grupo correspondiente, o bien listándolas haciendo uso del identificador mediante la CLI de OpenStack:

```
root@infra1-utility-container-23297593:/# openstack security group rule list 85e168bf-3fb2-43e1-84b7-0eac3b20641b
+--------------------------------------+-------------+-----------+-----------+------------+-----------+--------------------------------------+--------------------+
| ID                                   | IP Protocol | Ethertype | IP Range  | Port Range | Direction | Remote Security Group                | Remote Address Group |
+--------------------------------------+-------------+-----------+-----------+------------+-----------+--------------------------------------+--------------------+
| 066a7d16-3c76-41b9-a5b1-17c55a65e9e1 | None        | IPv6      | ::/0      |            | egress    | None                                 | None               |
| 3dbe0150-182c-43ef-9675-727994c1f882 | None        | IPv6      | ::/0      |            | ingress   | 85e168bf-3fb2-43e1-84b7-0eac3b20641b | None               |
| 5299671b-4162-49e3-8f15-15d20beb1389 | icmp        | IPv4      | 0.0.0.0/0 |            | egress    | None                                 | None               |
| 66661d3f-9dce-456d-b7bf-4c2e2dcefa5e | tcp         | IPv4      | 0.0.0.0/0 | 22:22      | ingress   | None                                 | None               |
| 6b1c0b61-ff39-46ee-b20e-b8c91cfb8746 | icmp        | IPv4      | 0.0.0.0/0 |            | ingress   | None                                 | None               |
| 849c0517-4a22-4dfe-98c2-a13a46f3c5b2 | None        | IPv4      | 0.0.0.0/0 |            | ingress   | 85e168bf-3fb2-43e1-84b7-0eac3b20641b | None               |
| c12be213-0768-42ae-a2ee-fc67cd89cf7d | tcp         | IPv6      | ::/0      | 22:22      | ingress   | None                                 | None               |
| f5ca40bc-1496-468b-b11d-2917c45e3f4f | None        | IPv4      | 0.0.0.0/0 |            | egress    | None                                 | None               |
+--------------------------------------+-------------+-----------+-----------+------------+-----------+--------------------------------------+--------------------+
```

Ilustración 28 Listado de reglas del grupo de seguridad predeterminado

Ahora solo nos queda generar un par de claves para poder acceder de forma segura a las instancias vía SSH. Este paso lo llevaremos a cabo en la interfaz web, tenemos que ubicarnos en la pestaña de **Computación** y pinchar en **Pares de claves**:

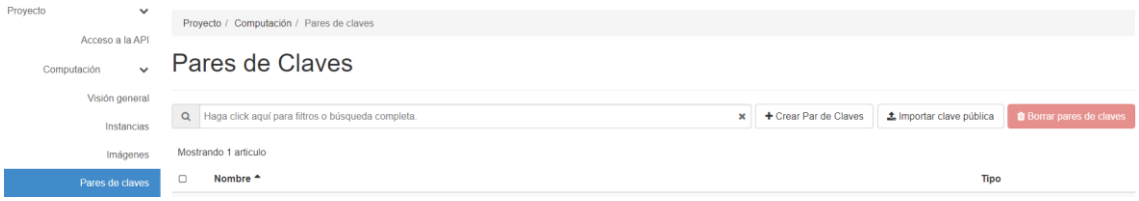

Ilustración 29 Creación de pares de claves

Seleccionamos **Crear Par de Claves**, introducimos un nombre, **Clave SSH** como **Tipo de Clave** y confirmamos la operación:

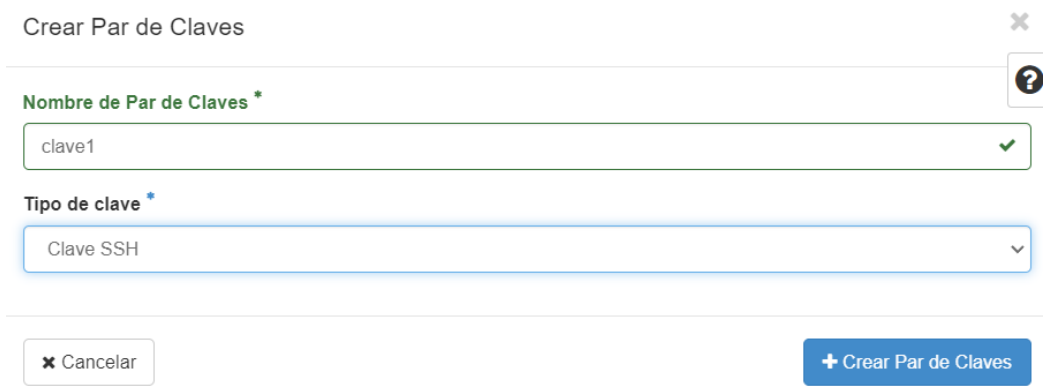

Ilustración 30 Opciones del par de claves

La clave privada se descargará automáticamente en nuestro ordenador, la cual podremos copiar fácilmente a cualquiera de los dos hosts mediante la función de arrastrar ficheros de MobaXterm.

4.2.2. Conformado de las redes de OpenStack

Habiendo añadido a nuestro entorno los componentes necesarios para crear una máquina virtual y conectarnos a ella, lo siguiente será establecer la infraestructura de red virtual sobre la que lanzaremos dicha instancia. Como explicamos anteriormente, estará formada por una red de proyecto (privada) capaz de comunicarse con la red física exterior a partir de una red de proveedor.

Creación de la red de proveedor

Comenzamos conformando la red de proveedor [19] desde el contenedor de "utilidades". Creamos la red de nombre "externa" con el comando **openstack network create**:

```
$ openstack network create externa --share --external --provider-network-type flat --provider-physical-network flat
```

El flag **--share** tiene una función parecida a la opción **--public** que empleábamos al crear la imagen y el *flavor*, que es la de permitir que todos los proyectos puedan utilizar esta red virtual. Mediante la opción **--external** especificamos que la red es externa y **--provider-physical-network** conecta la red física flat, que configuramos en el archivo **openstack_user_config.yml**, con esta red virtual. Por tanto, la red externa deberá ser a su vez de tipo flat (**--provider-network-type flat**).

A continuación, para aclarar un poco este proceso, explicaré los ficheros de configuración que marcan las directrices de la implementación de una red de proveedor.

OSA utiliza el agente Linux Bridge y el plugin ML2 para realizar la conexión de capa 2 entre ambas redes (de proveedor y proyecto). Durante el despliegue se usa la información del fichero **openstack_user_config.yml** para conformar sus archivos de configuración **ml2_conf.ini** y **linuxbridge_agent.ini**. Estos archivos se encuentran en el contenedor del servicio Neutron (en el directorio **/etc/neutron/plugins/ml2**), y se encarga de administrar las redes virtuales de OpenStack, y definir los parámetros necesarios para crear la conexión. Recordemos la configuración correspondiente a la red de proveedor en **openstack_user_config.yml**, ya que será necesario tener presente dicha información a la hora de comprender esta parte:

```
- network:
    container_bridge: "br-vlan"
    container_type: "veth"
    container_interface: "eth12"
    host_bind_override: "eth12"
    type: "flat"
    net_name: "flat"
    group_binds:
      - neutron_linuxbridge_agent
```

Ilustración 31 Configuración del puente br-vlan en el archivo openstack_user_config.yml

El fichero **ml2_conf.ini** contiene la configuración para el plugin ML2, un *framework* que dota al servicio de red Neutron de la capacidad de utilizar las diferentes tecnologías de red de capa 2 (flat, VLAN, VXLAN). De esta forma, aquí se especifican las características a tener en cuenta al conformar la red virtual de proveedor que se implementará sobre nuestra red física. La opción **type_drivers** indica los tipos de tecnología de capa 2 que podemos utilizar a la hora de crear una red externa. Dado que la nuestra es de tipo flat (sin VLAN), debemos dirigirnos a la opción **flat_networks** donde se indica el nombre a utilizar en el flag **--provider-physical-network,** en el comando de crear la red externa. Esta opción se ha rellenado con el valor de la sección **net_name**, referente a la configuración de la red *br-vlan* en el archivo **openstack_user_config.yml**:

```
[ml2]
type_drivers = flat,vlan,vxlan,local
tenant_network_types = vxlan,flat,vlan
mechanism_drivers = linuxbridge
extension_drivers = port_security
# ML2 flat networks

[ml2_type_flat]
flat_networks = flat
```

Ilustración 32 Archivo ml2_conf.ini

Al crear una red plana en la nube, la opción de configuración **physical_interface_mappings** del archivo **linuxbridge_agent.ini**, describe la conexión a realizar, entre la red flat y la interfaz que indicamos en la sección **container_interface** del fichero **openstack_user_config.yml** (container_interface: "eth12"). Recordemos que dicha interfaz estará a su vez conectada la interfaz virtual eth12 (host_bind_override: "eth12") del puente *br-vlan* de nuestra red física, completando así la unión por capa 2 entre la red de proveedor en OpenStack y la del laboratorio:

```
[linux_bridge]
physical_interface_mappings = flat:eth12,vlan:br-vlan
```

Ilustración 33 Archivo linuxbridge_agent.ini

Como resumen, la conexión desde la red de proveedor **externa** de OpenStack hasta nuestra red física se establece de la siguiente forma:

externa (flat) -> eth12 (veth de la red OpenStack) -> eth12 (veth de la red física) -> br-vlan -> eth4

Cabe destacar que en otros despliegues se emplea el puente **br-ex** (puentes OVS en vez de Linuxbrige). Dejando todo esto explicado, continuamos con la creación de una subred para la red **externa**:

```
$ openstack subnet create sub_provider --allocation-pool
start=172.29.21.185,end=172.29.21.189 --subnet-range 172.29.20.0/22 --no-dhcp --
gateway 172.29.21.201 --dns-nameserver 8.8.8.8 --network externa
```

La subred utilizará el rango de direcciones 172.29.20.0/22, correspondiente al laboratorio del departamento de automática, la dirección IP 172.29.21.201 del host de control como *gateway* para la red **externa** y especificamos un servidor DNS válido y DHCP deshabilitado. Por último, en base a las IPs del laboratorio que nos ha asignado la universidad para este proyecto (las cuáles van de la 172.29.21.170 a la 172.29.21.189), indicamos la primera y la última dirección IP del rango de la subred que se emplearán para asignar a las instancias lanzadas en el entorno.

Creación del router

La red de proveedor estará conectada a la red de proyecto privada mediante un router virtual. Para configurar este punto, crearemos el router que conectará ambas redes, que debe contener al menos una puerta de enlace en la red de proveedor y una interfaz en la red de proyecto. De momento, antes de pasar a la configuración de la red interna, podemos crearlo y establecer la red **externa** como su puerta de enlace [20]:

```
$ openstack router create --project admin router

$ openstack router set router --external-gateway externa
```

La dirección que se le asigna al *gateway* resultante de conectar la red externa con el router es la 172.29.21.188, acorde al rango de IPs de la subred **sub_provider**. En horizon, accediendo a "**Proyecto -> Red -> Topología de red**", podemos ver una representación gráfica de la red que llevamos configurada hasta el momento:

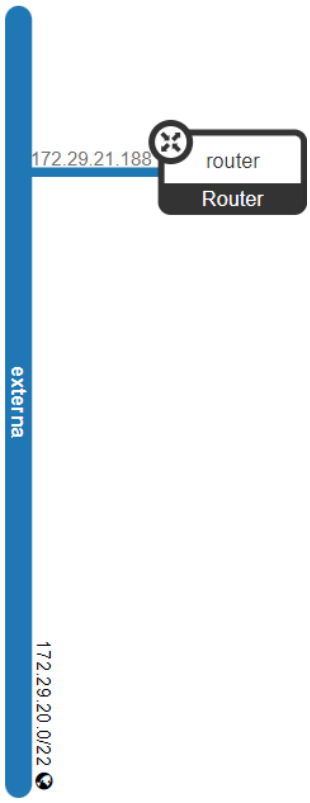

Ilustración 34 Topología de red (1)

Creación de la red de proyecto

Finalizamos la configuración de la red virtual estableciendo la red de proyecto **interna** y su respectiva subred [20]:

```
$ openstack network create interna

$ openstack subnet create sub_project --allocation-pool start=10.0.125.20,end=10.0.125.30 --subnet-range 10.0.124.0/22 --gateway 10.0.124.1 --dns-nameserver 10.0.124.1 --dns-nameserver 8.8.8.8 --network interna
```

El rango de la subred es 10.0.124.0/22, con la dirección .1 como puerta de enlace y DNS (junto a la 8.8.8.8) y 10 IPs a asignar a partir de la 10.0.125.20.

Solo nos queda añadir la subred como una interfaz en el router y habremos terminado con la configuración de la red virtual en OpenStack:

```
$ openstack router add subnet router sub_project
```

Una vez lanzado este comando, la puerta de enlace con dirección 10.0.124.1 será asignada para dicha interfaz. Podemos ver un resumen de todos los detalles del router ejecutando **openstack router show <nombre_del_router>**.

Con esto, la topología de red final resultante es la siguiente:

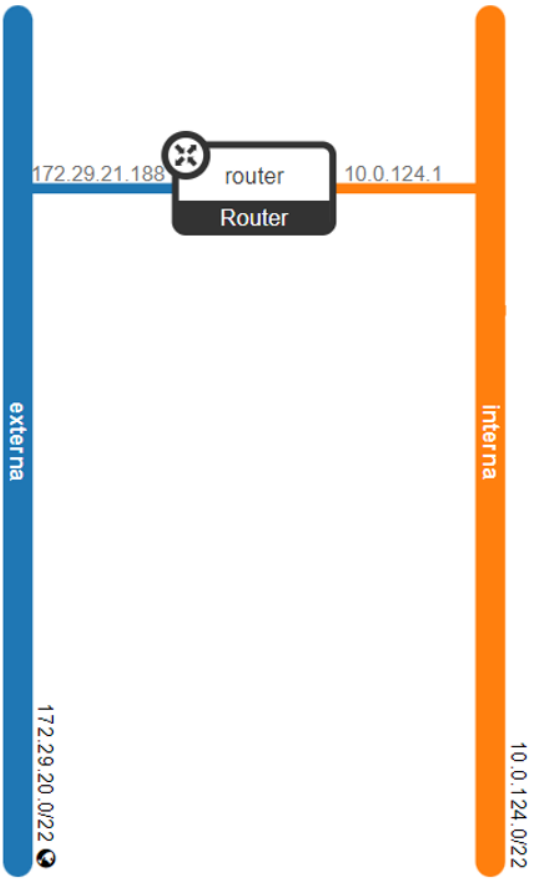

Ilustración 35 Topología de red (2)

Verificación inicial del funcionamiento de la red

Antes de lanzar directamente una instancia para testear si la conexión con el exterior es posible, podemos hacernos una idea de si el resultado será el esperado analizando el entorno generado.

En OpenStack, las redes, routers y muchos servicios de red (DHCP) están virtualizados en espacios de nombres de red (*namespace*). Un espacio de nombres se define como una copia de la pila de red con sus propia tabla de enrutamiento, reglas de *firewall,* dispositivos de interfaz de red y direcciones IP.

Por cada red privada o router que establecemos en nuestro entorno OpenStack, se creará a su vez un *namespace* asociado en el host de control desde el cuál se gestiona el tráfico. De esta forma, se permite que todos los routers virtuales y redes de nuestra nube coesxistan dentro del mismo host, sin que su enrutamiento interfiera entre sí o con el enrutamiento del propio host.

Sabiendo esto, si listamos los espacios de nombres existentes en el host de control deberíamos visualizar dos resultados, referentes a la red **interna** y al router de la nube:

```
silviu@control:~$ ip netns
qdhcp-49ffb60b-d007-4f21-938b-554bb51e4dd3 (id: 12)
qrouter-8f20142d-165a-4203-9017-df4cdecc0722 (id: 11)
```

Ilustración 36 Listado de espacios de nombres en el equipo de control

El identificador de los espacios de nombres de un enrutador está formado por [21] **qrouter-** y el ID del router, mientras que los espacios de nombres de red comienzan por **qdhcp-** seguido del ID de la red. Podemos comprobar que esto es así verificando el identificador del router mediante el comando **openstack router list**:

```
root@infra1-utility-container-23297593:/# openstack router list
+--------------------------------------+--------+--------+-------+----------------------------------+-------+
| ID                                   | Name   | Status | State | Project                          | HA    |
+--------------------------------------+--------+--------+-------+----------------------------------+-------+
| 8f20142d-165a-4203-9017-df4cdecc0722 | router | ACTIVE | UP    | 406bc2348d83438aaeea1b6671a49e61 | False |
+--------------------------------------+--------+--------+-------+----------------------------------+-------+
```

Ilustración 37 Listado de routers en la nube OSA

Y el ID de la red de proyecto al con **openstack network list**:

```
root@infra1-utility-container-23297593:/# openstack network list
+--------------------------------------+---------+--------------------------------------+
| ID                                   | Name    | Subnets                              |
+--------------------------------------+---------+--------------------------------------+
| 49ffb60b-d007-4f21-938b-554bb51e4dd3 | interna | 27e5c257-802a-4906-934a-3b23afc97968 |
| 5b9967f6-5295-4aa7-8080-f336c8465e1d | externa | 177ed6ee-c486-48bb-9523-23b303e5e2e5 |
+--------------------------------------+---------+--------------------------------------+
```

Ilustración 38 Listado de redes en la nube OSA

Habiendo determinado a quién pertenece cada *namespace*, podremos examinar sus configuraciones de red y testear su conectividad. Lo haremos a partir del comando [22] **ip netns exec <namespace_id> <comando>**, el cual nos permite ejecutar comandos dentro de espacios de nombres.

Si examinamos la tabla de rutas de la red **interna** observamos que, aparte de contener una ruta para la comunicación entre instancias de la misma red, tiene como *gateway* la interfaz del router que le hemos asignado, tal como vimos en la figura de la topología de red:

```
silviu@control:~$ sudo ip netns exec qdhcp-49ffb60b-d007-4f21-938b-554bb51e4dd3 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.124.1      0.0.0.0         UG    0      0        0 ns-2df6396c-57
10.0.124.0      0.0.0.0         255.255.252.0   U     0      0        0 ns-2df6396c-57
```

Ilustración 39 Tabla de encaminamiento de la red interna

Por su parte, el router tiene salida a la red física del laboratorio a partir del *gateway* 172.29.21.201 mediante el puente *br-vlan*:

```
silviu@control:~$ sudo ip netns exec qrouter-8f20142d-165a-4203-9017-df4cdecc0722 route -n
[sudo] password for silviu:
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.29.21.201   0.0.0.0         UG    0      0        0 qg-52e2583f-f7
10.0.124.0      0.0.0.0         255.255.252.0   U     0      0        0 qr-79c952b4-8e
172.29.20.0     0.0.0.0         255.255.252.0   U     0      0        0 qg-52e2583f-f7
```

Ilustración 40 Tabla de encaminamiento del router

Y una vez el tráfico llegue al host de control, podremos acceder a Internet o comunicarnos con las máquinas de la red 172.29.20.0/22 siguiendo la primera y la última ruta de su tabla:

```
silviu@control:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.29.20.1     0.0.0.0         UG    0      0        0 eth1
10.0.3.0        0.0.0.0         255.255.255.0   U     0      0        0 lxcbr0
10.0.120.0      0.0.0.0         255.255.252.0   U     0      0        0 br-mgmt
10.0.124.0      0.0.0.0         255.255.252.0   U     0      0        0 br-vxlan
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
172.29.20.0     0.0.0.0         255.255.252.0   U     0      0        0 eth1
```

Ilustración 41 Tabla de encaminamiento del host de control

A priori, según la información que nos muestran las tablas de enrutamiento, podemos deducir que el tráfico de la red **interna** puede comunicarse con el exterior. Es más, somos capaces de comprobarlo desde su *namespace* mediante el comando **ip netns exec**:

```
silviu@control:~$ sudo ip netns exec qdhcp-49ffb60b-d007-4f21-938b-554bb51e4dd3 ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=1.85 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=1.79 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=118 time=1.69 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.693/1.777/1.847/0.063 ms
silviu@control:~$ sudo ip netns exec qdhcp-49ffb60b-d007-4f21-938b-554bb51e4dd3 ping 172.29.21.183
PING 172.29.21.183 (172.29.21.183) 56(84) bytes of data.
64 bytes from 172.29.21.183: icmp_seq=1 ttl=63 time=0.267 ms
64 bytes from 172.29.21.183: icmp_seq=2 ttl=63 time=0.329 ms
^C
--- 172.29.21.183 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1011ms
rtt min/avg/max/mdev = 0.267/0.298/0.329/0.031 ms
```

Ilustración 42 Test ICMP de la red interna

Desde la red de proyecto, obtenemos respuesta al realizar un PING tanto a Internet como a una dirección del laboratorio. Una vez confirmado que la configuración de la red virtual funciona correctamente, estamos preparados para realizar la prueba final desde una instancia albergada en la nube.

4.2.3. Salida a Internet desde una instancia

Despliegue de una instancia

Es el momento de utilizar los datos de toda la configuración hecha previamente para lanzar una máquina virtual [20]. Del apartado de “Prerrequisitos” necesitamos el nombre de la imagen creada (ubuntu16), el “sabor” correspondiente (ubuntu16.flavor), el nombre del par de claves a utilizar (clave1) y el ID correspondiente al grupo de seguridad **default** del proyecto **admin** (**85e168bf-3fb2-43e1-84b7-0eac3b20641b**). Por último, será necesario indicar el ID de la red donde se va a desplegar la instancia, en este caso, en la red **interna**. Recordemos que dicho ID lo obtuvimos a partir del comando **openstack network list**:

```
root@infra1-utility-container-23297593:/# openstack network list
+--------------------------------------+---------+--------------------------------------+
| ID                                   | Name    | Subnets                              |
+--------------------------------------+---------+--------------------------------------+
| 49ffb60b-d007-4f21-938b-554bb51e4dd3 | interna | 27e5c257-802a-4906-934a-3b23afc97968 |
| 5b9967f6-5295-4aa7-8080-f336c8465e1d | externa | 177ed6ee-c486-48bb-9523-23b303e5e2e5 |
+--------------------------------------+---------+--------------------------------------+
```

Ilustración 43 Listado de redes de la nube OSA

Cabe destacar que, en caso de que la nube privada cuente con más de un nodo de cómputo, indicamos donde queremos lanzar la instancia mediante el *flag* **--availability-zone** seguido de la zona y el nombre del nodo. Una forma sencilla de identificar los nombres de los ordenadores

físicos que ejecutan un hipervisor y su zona correspondiente es a partir de la salida del comando **openstack host list**:

```
root@infra1-utility-container-23297593:/# openstack host list
+-----------------------------------+-----------+----------+
| Host Name                         | Service   | Zone     |
+-----------------------------------+-----------+----------+
| infra1-nova-api-container-7fc29f55 | conductor | internal |
| infra1-nova-api-container-7fc29f55 | scheduler | internal |
| ansible                           | compute   | nova     |
```

Ilustración 44 Listado de hipervisores de la nube OSA

Dado que nuestro entorno sólo cuenta con el nodo de cómputo **ansible**, la instancia se creará en dicho host sin necesidad de indicar nada al respecto:

```
$ openstack server create ubuntu --image 'ubuntu16' --flavor ubuntu16.flavor --key-name clave1 --security-group 85e168bf-3fb2-43e1-84b7-0eac3b20641b --nic net-id=49ffb60b-d007-4f21-938b-554bb51e4dd3
```

El resultado es una nueva instancia, a la que hemos llamado “ubuntu”, ubicada en la red interna a la que se le ha asignado la dirección 10.0.125.25:

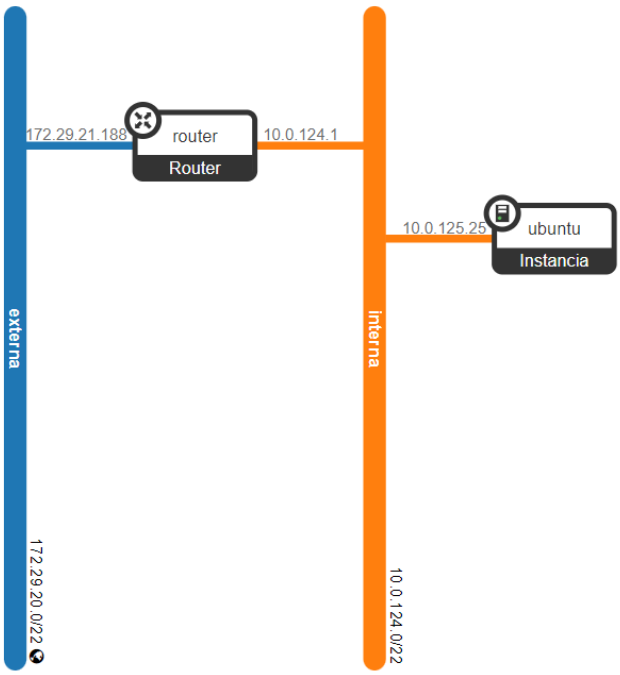

Ilustración 45 Topología de red (3)

Tanto desde Horizon, accediendo a “**Proyecto -> Administrador -> Computación -> Hipervisores**”, como desde la CLI de OpenStack, ejecutando **openstack host show ansible**, podemos visualizar los recursos que se le ha asignado a la instancia del nodo de cómputo:

Hipervisor | Anfitrión de computación

Mostrando 2 artículos

Hostname	Type	VCPUs (used)	VCPUs (total)	RAM (used)	RAM (total)	Local Storage (used)	Local Storage (total)	Instances
ansible	QEMU	1	8	3GB	31,2GB	7GB	113GB	1

Ilustración 46 Resumen de recursos consumidos desde Horizon

```
root@infra1-utility-container-23297593:/# openstack host show ansible
+---------+----------------------------------+-----+-----------+---------+
| Host    | Project                          | CPU | Memory MB | Disk GB |
+---------+----------------------------------+-----+-----------+---------+
| ansible | (total)                          |   8 |     31987 |     113 |
| ansible | (used_now)                       |   1 |      3072 |       7 |
| ansible | (used_max)                       |   1 |      1024 |       5 |
| ansible | 406bc2348d83438aaeea1b6671a49e61 |   1 |      1024 |       5 |
+---------+----------------------------------+-----+-----------+---------+
```

Ilustración 47 Resumen de recursos consumidos desde la CLI de OpenStack

Test ICMP

De momento la instancia "ubuntu" sólo cuenta con una dirección privada dentro del rango de la red de proyecto dónde ha sido lanzada que se utiliza para comunicación interna. Pero, para poder conectarnos a ella vía SSH y realizar las pruebas de conexión pertinentes, debemos dotarla de una IP perteneciente a la red **externa**. OpenStack define este tipo de direcciones asociadas a la red de proveedor que nos permiten comunicarnos hacia fuera de la nube como IPs flotantes.

Como resultado de crear una IP flotante a partir del *pool* de direcciones de la red externa ejecutando el comando **openstack floating ip create externa**, obtenemos la dirección 172.29.21.185. Asignaremos la dirección proporcionada a nuestra instancia "ubuntu":

```
$ openstack server add floating ip ubuntu 172.29.21.185
```

Listando las IPs flotantes de nuestro entorno (openstack floating ip list), o las instancias que hemos lanzado en la nube (openstack server list), nos percatamos de que la asociación se ha realizado correctamente:

```
root@infra1-utility-container-23297593:/# openstack floating ip list
+--------------------------------------+---------------------+------------------+
| ID                                   | Floating IP Address | Fixed IP Address |
+--------------------------------------+---------------------+------------------+
| 4b238056-7326-4fd1-be57-b5f9f039a216 | 172.29.21.185       | 10.0.125.25      |
+--------------------------------------+---------------------+------------------+
```

Ilustración 48 Listado de IPs flotantes de la nube OSA

```
root@infra1-utility-container-23297593:/# openstack server list
+--------------------------------------+--------+--------+----------------------------------+---------+----------------+
| ID                                   | Name   | Status | Networks                         | Image   | Flavor         |
+--------------------------------------+--------+--------+----------------------------------+---------+----------------+
| 488c6377-604e-485c-b2c1-c3c80e3552e3 | ubuntu | ACTIVE | interna=10.0.125.25, 172.29.21.185 | ubuntu16 | ubuntu16.flavor |
+--------------------------------------+--------+--------+----------------------------------+---------+----------------+
```

Ilustración 49 Listado de las VMs desplegadas en la nube OSA

Ahora podremos comunicarnos con la instancia utilizando esta dirección, vía el puente *br-vlan* que conecta nuestra red física con la nube:

```
silviu@control:~$ ping 172.29.21.185
PING 172.29.21.185 (172.29.21.185) 56(84) bytes of data.
64 bytes from 172.29.21.185: icmp_seq=1 ttl=63 time=1.04 ms
64 bytes from 172.29.21.185: icmp_seq=2 ttl=63 time=1.03 ms
64 bytes from 172.29.21.185: icmp_seq=3 ttl=63 time=0.926 ms
^C
--- 172.29.21.185 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.926/0.999/1.042/0.051 ms
```

Ilustración 50 Test ICMP hacía la instancia lanzada en la nube OSA

Por tanto, podremos conectarnos a ella por SSH mediante la clave privada “clave.pem” que creamos en el apartado “Prerrequisitos”, y desde la máquina virtual, testear la conexión a Internet y a nuestros nodos:

Ilustración 51 Test ICMP desde la instancia hacía el exterior (1)

Así como a otros equipos existentes en el laboratorio:

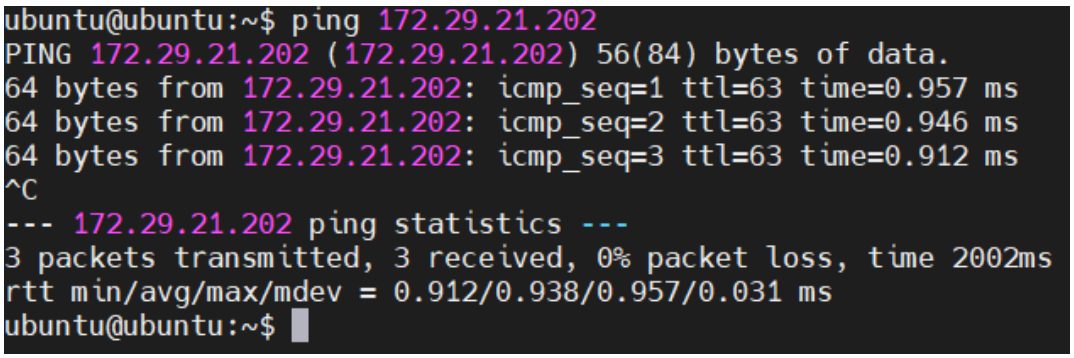

Ilustración 52 Test ICMP desde la instancia hacía el exterior (2)

Cabe destacar, que se puede dar el caso en el que la comunicación solo sea posible desde la nube hacía nuestra red física, fallando en dirección a OpenStack. Una posible solución es indicarle directamente a la tabla de encaminamiento la ruta hacía las instancias de la nube, al proporcionarle al puente **br-vlan** una IP del laboratorio.

5. Nube híbrida OpenStack Omni

Iniciamos la búsqueda de un método que nos permita conformar una nube híbrida a partir de nuestra nube privada, con el proyecto OpenStack Omni [23]. Se trata de un proyecto desarrollado por Platform 9, que tiene el objetivo de integrar un entorno OpenStack con algunos de los principales proveedores de nube pública; Amazon Web Services, Google Cloud y Microsoft Azure.

Omni se encarga de proporcionar una API estándar de OpenStack desde la cual podemos gestionar una nube pública como si fuese propia de OpenStack. Para conseguir esto, cuenta con un conjunto de controladores, o drivers, para el entorno de la nube pública que van conectados a los servicios de OpenStack (principalmente Nova, Neutron, Cinder y Glance), permitiendo la gestión de la nube pública, así como la infraestructura de la privada.

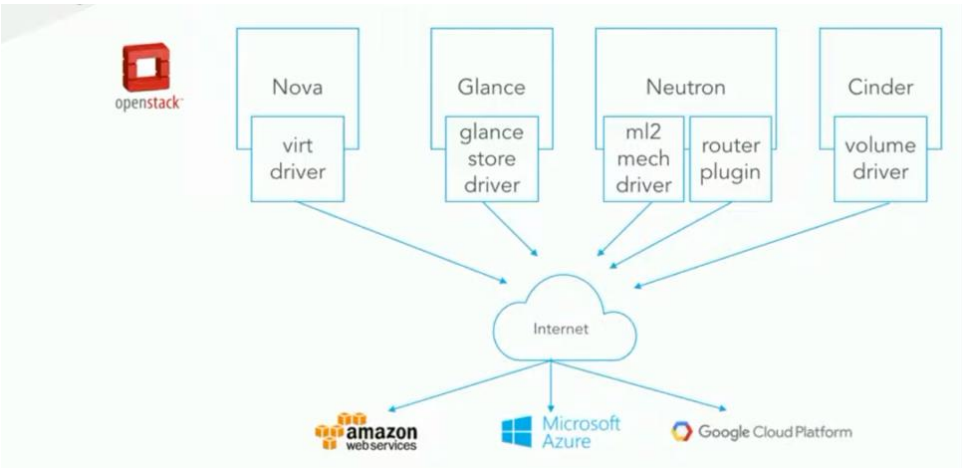

Ilustración 53 Esquema de la arquitectura del sistema Omni

Para configurarlo, se deben añadir al entorno de la nube OpenStack los controladores mencionados, que están disponibles en un repositorio público, junto a las credenciales de la nube pública que vayamos a utilizar. De esta forma, el sistema Omni obtendrá permiso, a partir de dichas credenciales, a la API de la nube pública, y así, podrá efectuar llamadas que permitan a los controladores realizar tareas dentro de ella como; crear instancias, redes, volúmenes o imágenes.

5.1. Instalación de OpenStack Omni en la nube OSA

El proceso de configuración de Omni en una nube privada OpenStack lo encontramos en su repositorio correspondiente (https://opendev.org/x/omni). Dentro, cada servicio que posee un controlador tiene una carpeta con su nombre, es decir, hay cuatro carpetas llamadas Nova, Neutron, Cinder y Glance. En cada carpeta encontramos los drivers de cada uno y un archivo **README.md**, el cuál explica los pasos a realizar para añadirlos a nuestro entorno OpenStack, que son básicamente los mismos para cada servicio:

1. Comprobar que cumplimos con la lista de prerrequisitos necesarios para que el controlador se integre correctamente.
2. Copiar la carpeta del repositorio que contiene los drivers, a la carpeta del entorno OpenStack que almacena los drivers del servicio en cuestión.

3. Editar el fichero de configuración del servicio correspondiente (nova.conf, neutron.conf, cinder.conf o glance-api.conf dependiendo del servicio) con las credenciales de la API correspondientes a la nube pública elegida.
4. Reiniciar el servicio.
5. Por último, hay carpetas que contienen scripts de prueba que podemos ejecutar para testear que el controlador funciona correctamente.

Sin embargo, atendiendo a los prerrequisitos correspondientes al controlador del servicio de Neutron, nos percatamos de que no cumplimos con una de las condiciones expuestas. El archivo **README.md** nos indica que en la nube OpenStack no deben existir agentes Neutron configurados antes de añadir el driver de Omni. En caso de contar con el agente Neutron L3 y el driver ML2 configurados, este servicio no funciona.

OpenStack-Ansible, como vimos en el apartado de la configuración de la red de proveedor, configura dichos agentes en el proceso de despliegue, como cualquier otro método de despliegue en general. Pareciera ser que este proyecto está más enfocado hacía un entorno OpenStack instalado manualmente, en el que sea posible modelar Neutron de cero.

Debido a que sin el controlador de Neutron no podemos utilizar el de Nova (porque en OpenStack primero es necesario configurar una red sobre la que lanzar las VMs), y a que existe otra alternativa más sencilla para utilizar sistema Omni con una nube desplegada mediante DevStack, de momento dejaremos apartada la nube privada OSA. Teniendo una nube OpenStack completamente funcional, y sabiendo que le podemos dar uso en el futuro si la necesitamos para utilizar otro método que nos permita conformar una nube híbrida, se ha tomado la decisión de no modificarla.

5.2. Instalación de OpenStack Omni en una nube DevStack

A parte de la opción de inyectar los controladores a un entorno OpenStack existente, tenemos la posibilidad de utilizar el método DevStack para desplegar una nube con el sistema Omni ya instalado. El equipo de Platform 9 pone a nuestra disposición un plugin para DevStack que se encarga de configurar los drivers de Omni durante el proceso de instalación. De esta forma, sólo será necesario añadir dicho plugin al archivo de configuración del despliegue de DevStack y ejecutarlo.

Para ello, nos apoyaremos en el ejemplo de despliegue expuesto en el libro “Hybrid Cloud for Architects” [4]. En el capítulo “Using PreBuilt Hybrid Cloud Solutions”, el autor conforma una nube híbrida con OpenStack y Amazon Web Services mediante este proyecto.

5.2.1 Prueba de concepto DevStack

Antes de pasar a la sección del despliegue de DevStack con el sistema Omni integrado, primero vamos a familiarizarnos con este método. Procederemos de esta forma ya que, en un primer momento, me dispuse a replicar directamente el ejemplo realizado por Alok Shrivastwa en su libro, obteniendo como resultado varios intentos de despliegue fallidos. Por tanto, ejecuté una instalación sencilla que me ayudase a comprender mejor cómo funciona DevStack, y así estar más preparado a la hora de buscar soluciones a los errores acontecidos durante mis primeros intentos.

Simplemente conformaremos una nube con un único nodo que, aparte de desplegar la nube y ocuparse de su control, contendrá todos los servicios de OpenStack, los cuales testearemos realizando la prueba de lanzar una instancia y verificar su conexión a Internet. Utilizaremos otro ordenador del laboratorio que, aunque esté más limitado en recursos, nos servirá para este

despliegue. Dicho host cuenta con un Ubuntu Desktop 20.04 y la siguiente configuración de red en el archivo **/etc/netplan/01-network-manager-all.yaml**:

```
# Let NetworkManager manage all devices on this system
network:
  version: 2
  renderer: NetworkManager
  ethernets:
          enp2s0:
                  dhcp4: no
                  addresses: [172.29.21.181/22]
                  gateway4: 172.29.20.1
                  nameservers:
                          addresses: [8.8.8.8, 1.1.1.1]
```

Ilustración 54 Archivo de configuración de red para despliegue DevStack

Le asignamos una IP del laboratorio y, al igual que en configuraciones anteriores, su correspondiente gateway al router de la Universidad.

Instalación de DevStack en el equipo

La documentación oficial de DevStack [24] nos indica que la instalación debe ser ejecutada con un usuario no-root con privilegios *sudo*. Crearemos el usuario *stack* para este propósito:

```
$ sudo useradd -s /bin/bash -d /opt/stack -m stack
```

Proporcionamos privilegios sudo al nuevo usuario:

```
$ echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack
```

Actualizamos el sistema y accedemos al usuario *stack*, desde donde realizaremos el despliegue:

```
$ sudo apt update

$ sudo apt -y upgrade && sudo apt -y dist-upgrade

$ sudo -u stack -i
```

Y elegimos una versión estable, en nuestro caso Victoria, de DevStack a descargar del repositorio público https://opendev.org/openstack/devstack:

```
$ git clone https://opendev.org/openstack/devstack -b stable/victoria
```

Configuración del despliegue

Al clonar el repositorio se habrá creado el directorio **devstack** en nuestro equipo, donde podremos encontrar los scripts necesarios para la instalación de OpenStack. Nos movemos a dicho directorio y copiamos el archivo de configuración **local.conf** de ejemplo que nos proporciona el despliegue en la carpeta principal:

```
$ cd devstack

$ cp samples/local.conf .
```

El fichero **local.conf** [25] nos permite especificar las características del entorno OpenStack a desplegar. Aquí es dónde indicaremos los servicios que vendrán instalados en nuestra nube y demás opciones de configuración. Editaremos el archivo de configuración de ejemplo que recién hemos copiado, para conformar una nube privada OpenStack basada en nuestras mínimas necesidades. Ejecutamos "**cat local.conf | grep -v "#" | grep -v "^$"**" para visualizar el contenido del fichero resultante sin comentarios:

```
[[local|localrc]]
ADMIN_PASSWORD=admin
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
HOST_IP=172.29.21.181
FLOATING_RANGE=172.29.20.0/22
Q_FLOATING_ALLOCATION_POOL=start=172.29.21.183,end=172.29.21.184
PUBLIC_NETWORK_GATEWAY=172.29.21.181
LOGFILE=$DEST/logs/stack.sh.log
LOGDAYS=2
SWIFT_HASH=66a3d6b56c1f479c8b4e70ab5c2000f5
SWIFT_REPLICAS=1
SWIFT_DATA_DIR=$DEST/data
```

Ilustración 55 Configuración del archivo local.conf

Las opciones que hemos modificado o añadido indican:

- **ADMIN_PASSWORD**: contraseña del administrador de la nube. A su vez, indicamos esta variable como la contraseña de la base de datos, Rabbit y la interfaz web Horizon.
- **HOST_IP**: indica la dirección IP del host que se encargará del control de la nube.
- **FLOATING_RANGE**: rango de IPs flotantes asociado a la red proveedor que nos conceda salida a Internet.
- **Q_FLOATING_ALLOCATION_POOL**: especifica las IPs flotantes dentro del rango **FLOATING_RANGE** que se asignarán a los dispositivos de la nube OpenStack.
- **PUBLIC_NETWORK_GATEWAY**: dirección *gateway* de la red de proveedor.

Despliegue y verificación del entorno

Con el despliegue configurado, simplemente tenemos que ejecutar el script **stack.sh** del directorio **devstack**:

```
$ . /stack.sh
```

Cuando la instalación se haya completado, se nos mostrará por terminal la información necesaria para iniciar sesión en el entorno de OpenStack:

```
=========================
DevStack Component Timing
 (times are in seconds)
=========================
wait_for_service      44
pip_install          113
apt-get              180
run_process           29
dbsync               911
apt-get-update        20
test_with_retry       10
osc                  190
-------------------------
Unaccounted time     1326
=========================
Total runtime        2823


This is your host IP address: 172.29.21.181
This is your host IPv6 address: ::1
Horizon is now available at http://172.29.21.181/dashboard
Keystone is serving at http://172.29.21.181/identity/
The default users are: admin and demo
The password: admin

Services are running under systemd unit files.
For more information see:
https://docs.openstack.org/devstack/latest/systemd.html

DevStack Version: victoria
Change: 06c5cb0adc53949799e36e6222c7d57b5be93ad1 Write safe.directory items to system git config 2022-04-18 21:47:09 -0500
OS Version: Ubuntu 20.04 focal
```

Ilustración 56 Fin del proceso de instalación DevStack

Según esto, podemos acceder al cuadro de mandos Horizon mediante el usuario y contraseña **admin** en la dirección http://172.29.21.181/dashboard. Una vez dentro, dirigiéndonos a la sección "**Proyecto -> Red -> Topología de red**" del proyecto **demo**, observamos la red que DevStack ha implementado en la nube durante el despliegue, teniendo en cuenta los parámetros indicados en el fichero **local.conf**:

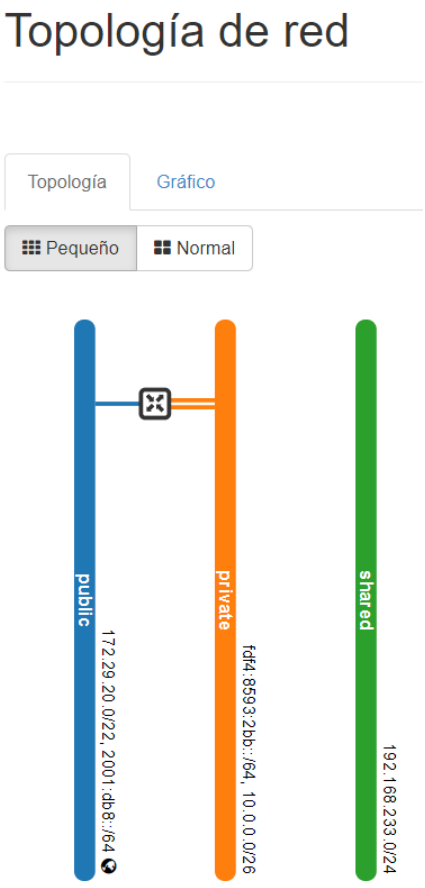

Ilustración 57 Topología de red de la nube DevStack

La red de proveedor **public**, que nos permite el acceso a Internet, se ha configurado con el rango de la red física del laboratorio que especificamos en **FLOATING_RANGE**. DevStack conecta esta red con la del laboratorio implementando un puente denominado **br-ex**, al cual se le asigna la dirección IP indicada en **PUBLIC_NETWORK_GATEWAY**. Este puente se configura utilizando el agente OVS, en vez de Linux Bridge como era el caso de OSA.

Si listamos la tabla de enrutamiento del host de control, podemos observar que tenemos acceso a la red de proveedor OpenStack a partir del puente **br-ex**:

```
silviu@silviu-devstack:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.29.20.1     0.0.0.0         UG    100    0        0 enp2s0
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 virbr0
172.29.20.0     0.0.0.0         255.255.252.0   U     0      0        0 br-ex
172.29.20.0     0.0.0.0         255.255.252.0   U     100    0        0 enp2s0
192.168.122.0   0.0.0.0         255.255.255.0   U     0      0        0 virbr0
silviu@silviu-devstack:~$
```

Ilustración 58 Tabla de rutas del host de control de la nube DevStack

Comprobamos también que se han generado los espacios de nombre equivalentes a las redes de proyecto **private** y **shared**, y al **router** de la red OpenStack:

```
silviu@silviu-devstack:~$ ip netns list
qdhcp-034eb818-b4c6-45f9-913f-f817958d1941 (id: 2)
qrouter-764043d4-efb3-4f31-b392-46382ffccd45 (id: 1)
qdhcp-80670379-d473-4295-9ff6-6f77567fafcd (id: 0)
```

Ilustración 59 Listado de espacios de nombres del host de control DevStack

Y que desde la nube tenemos acceso a Internet a partir de la puerta de enlace 172.29.21.181, que es la dirección del equipo asignada al puente **br-ex**:

```
silviu@silviu-devstack:~$ sudo ip netns exec qrouter-764043d4-efb3-4f31-b392-46382ffccd45 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.29.21.181   0.0.0.0         UG    0      0        0 qg-c1ef31de-d1
10.0.0.0        0.0.0.0         255.255.255.192 U     0      0        0 qr-fcecefe0-e1
172.29.20.0     0.0.0.0         255.255.252.0   U     0      0        0 qg-c1ef31de-d1
silviu@silviu-devstack:~$
```

Ilustración 60 Tabla de rutas del router de la nube DevStack

```
silviu@silviu-devstack:~$ sudo ip netns exec qrouter-764043d4-efb3-4f31-b392-46382ffccd45 ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=2.18 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=1.63 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=118 time=1.60 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 1.598/1.804/2.183/0.267 ms
```

Ilustración 61 Salida a Internet desde el router de la nube DevStack

Prueba de salida a Internet desde una instancia

El siguiente paso será crear una instancia, tal como lo hicimos en el caso del despliegue con OpenStack-Ansible, para verificar que todo está en orden. En primer lugar, deberemos configurar las variables de entorno OpenStack CLI (variables **openrc**) para el proyecto. Se pueden descargar desde la Web GUI, en "**Proyecto -> Acceso a la API -> Descargar fichero RC de**

OpenStack". Como resultado de la descarga obtenemos el fichero **demo-openrc.sh**, inyectamos sus credenciales en el equipo al ejecutarlo con el comando **source** y proporcionar la contraseña del administrador.

Siguiendo los pasos que tomamos en la nube OSA, será necesario editar el grupo de seguridad que se asignará a la VM para poder conectarnos a ella vía SSH. Para obtener el identificador del grupo correcto (recordemos que hay un grupo con el nombre **default** por cada proyecto), primero necesitamos el ID del proyecto **demo**. Otra forma de encontrarlo fácilmente es a partir del comando "**env | grep OS_PROJECT_ID**", que busca dicho identificador entre las variables de entorno del equipo. Una vez adquirido (**19058a1205274e1da3be49d6d909c049**), listamos los grupos de seguridad presentes en nuestra nube e identificamos el grupo que nos interesa a partir de la columna **Project**:

```
stack@silviu-devstack:~$ openstack security group list
+--------------------------------------+---------+------------------------+----------------------------------+------+
| ID                                   | Name    | Description            | Project                          | Tags |
+--------------------------------------+---------+------------------------+----------------------------------+------+
| 0689cc00-5a7a-433e-bf1c-e2dcdba99aaf | default | Default security group | 19058a1205274e1da3be49d6d909c049 | []   |
| e70f48fd-14e9-40a0-9c27-6792dd79d17c | default | Default security group | f56f78b5f741434d91239d27ba980de4 | []   |
+--------------------------------------+---------+------------------------+----------------------------------+------+
```

Ilustración 62 Listado de grupos de seguridad en la nube DevStack

Añadimos las reglas que permiten la conexión remota por SSH a partir del ID de grupo:

```
$ openstack security group rule create --proto tcp --dst-port 22 0689cc00-5a7a-433e-bf1c-e2dcdba99aaf

$ openstack security group rule create --ethertype IPv6 --proto tcp --dst-port 22 0689cc00-5a7a-433e-bf1c-e2dcdba99aaf
```

Y comprobamos que nuestras reglas han sido añadidas correctamente:

```
stack@silviu-devstack:~$ openstack security group rule list 0689cc00-5a7a-433e-bf1c-e2dcdba99aaf
+--------------------------------------+-------------+-----------+-----------+------------+--------------------------------------+
| ID                                   | IP Protocol | Ethertype | IP Range  | Port Range | Remote Security Group                |
+--------------------------------------+-------------+-----------+-----------+------------+--------------------------------------+
| 08349419-7cbd-4d67-96f1-cdf781caf60f | tcp         | IPv4      | 0.0.0.0/0 | 22:22      | None                                 |
| afee9a55-6cfb-4bf8-b50f-a85dbf24ee36 | None        | IPv4      | 0.0.0.0/0 |            | 0689cc00-5a7a-433e-bf1c-e2dcdba99aaf |
| b8a22134-2085-48ed-8712-f7573f4e6671 | None        | IPv4      | 0.0.0.0/0 |            | None                                 |
| ed530498-22ea-43f4-b205-afdf36b19f99 | None        | IPv6      | ::/0      |            | 0689cc00-5a7a-433e-bf1c-e2dcdba99aaf |
| fc0013ce-8aef-4068-a0fa-d7a9d00749a1 | tcp         | IPv6      | ::/0      | 22:22      | None                                 |
| fea92aaf-07ef-4964-a6ba-b3f1feba1f0c | None        | IPv6      | ::/0      |            | None                                 |
+--------------------------------------+-------------+-----------+-----------+------------+--------------------------------------+
```

Ilustración 63 Listado de reglas del grupo de seguridad predeterminado de DevStack

Ahora, dado que durante el despliegue DevStack ya se ha ocupado de añadir a nuestro entorno una lista de "sabores" y una imagen por defecto para la creación de VMs, nos será más sencillo lanzar una instancia mediante la interfaz web. Accedemos a "**Computación -> Instancias**" y pinchamos en "**Lanzar instancia**". Le proporcionamos un nombre e indicamos la zona en la que se ejecutará y el número de instancias a crear con estas características:

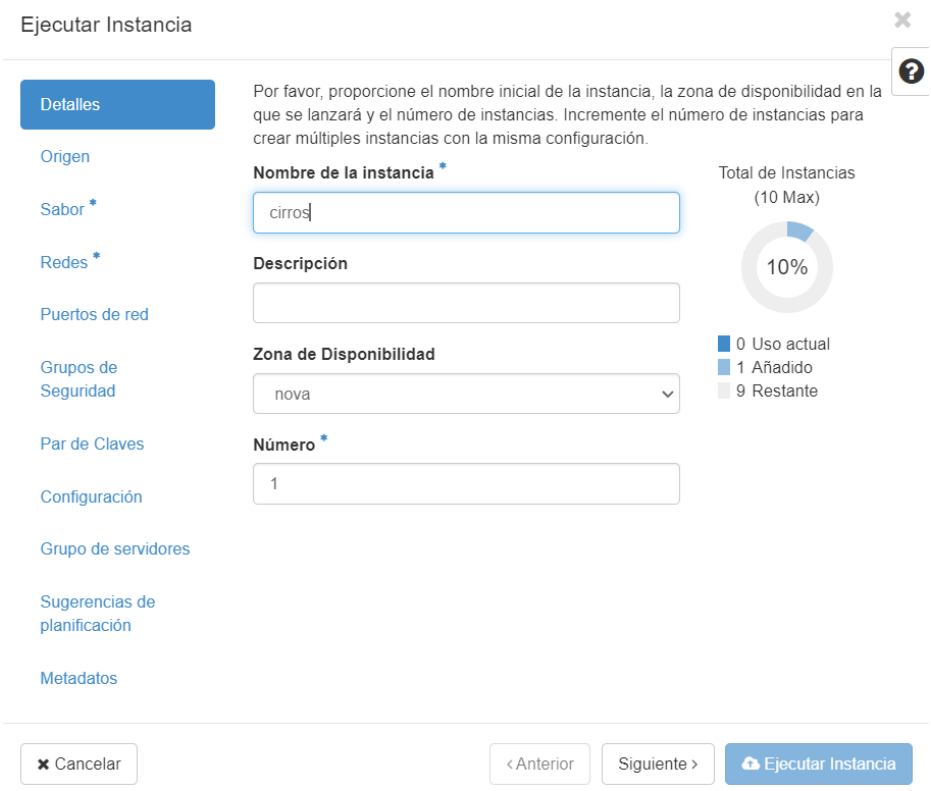

Ilustración 64 Despliegue de una instancia desde Horizon (1)

En la pestaña "**Origen**", seleccionamos una imagen **cirros** que tenemos disponible como fuente de arranque de la máquina virtual:

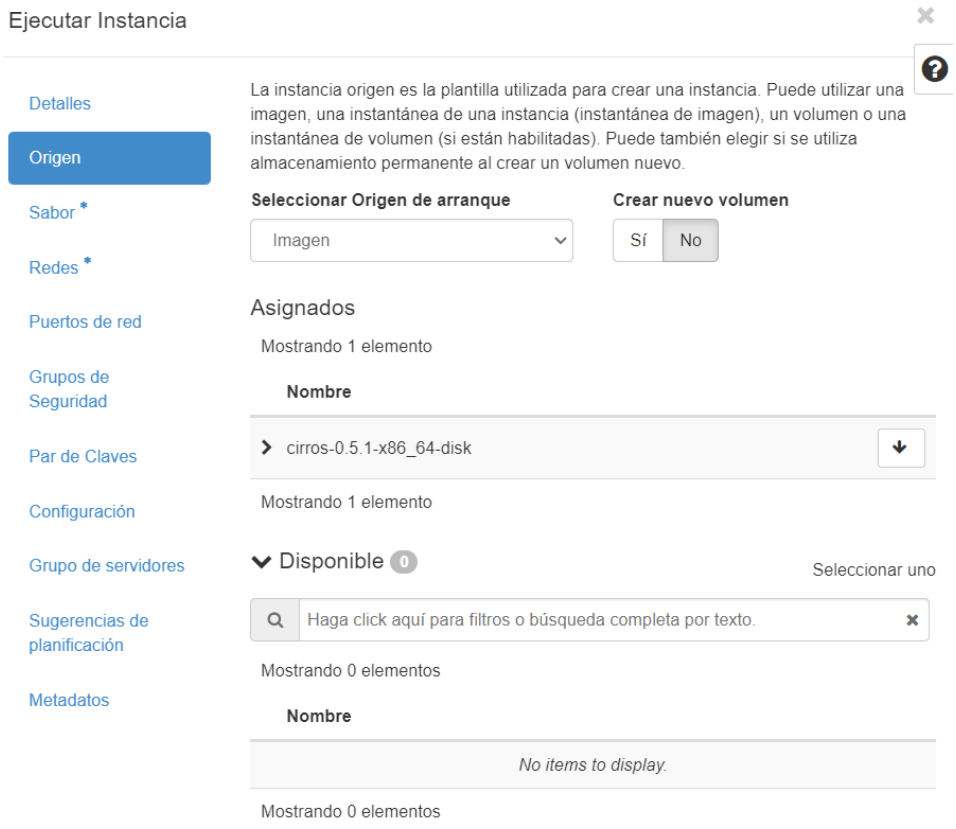

Ilustración 65 Despliegue de una instancia desde Horizon (2)

Seleccionamos un *flavor* de pequeño tamaño:

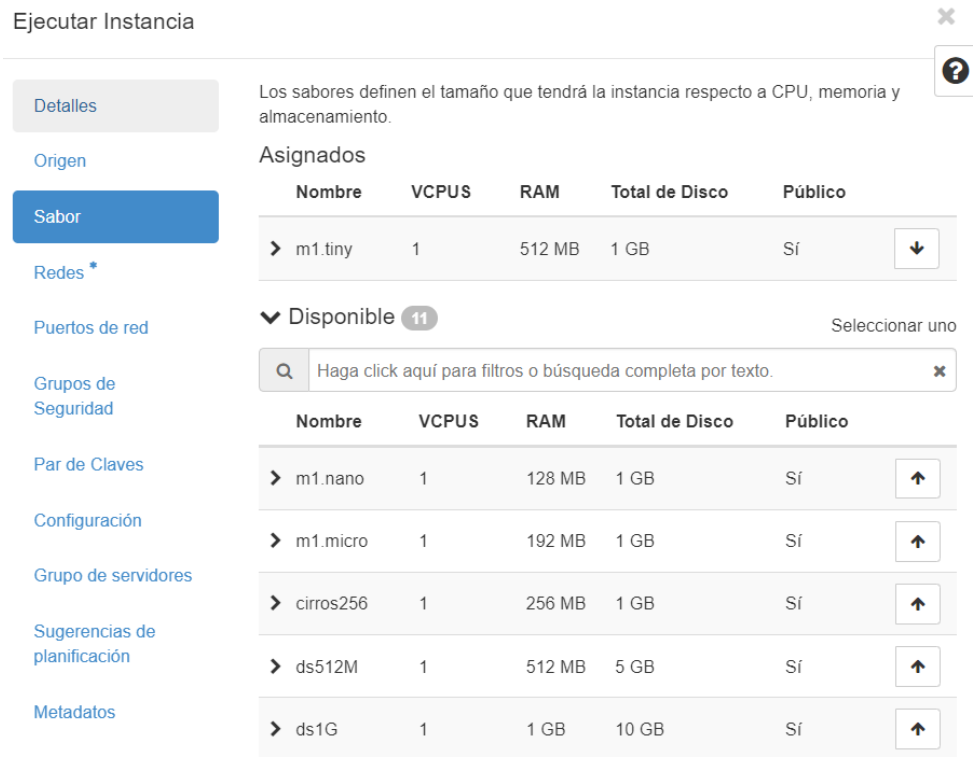

Ilustración 66 Despliegue de una instancia desde Horizon (3)

Indicamos la red donde se ejecutará la instancia, en este caso en la red de proyecto **private**:

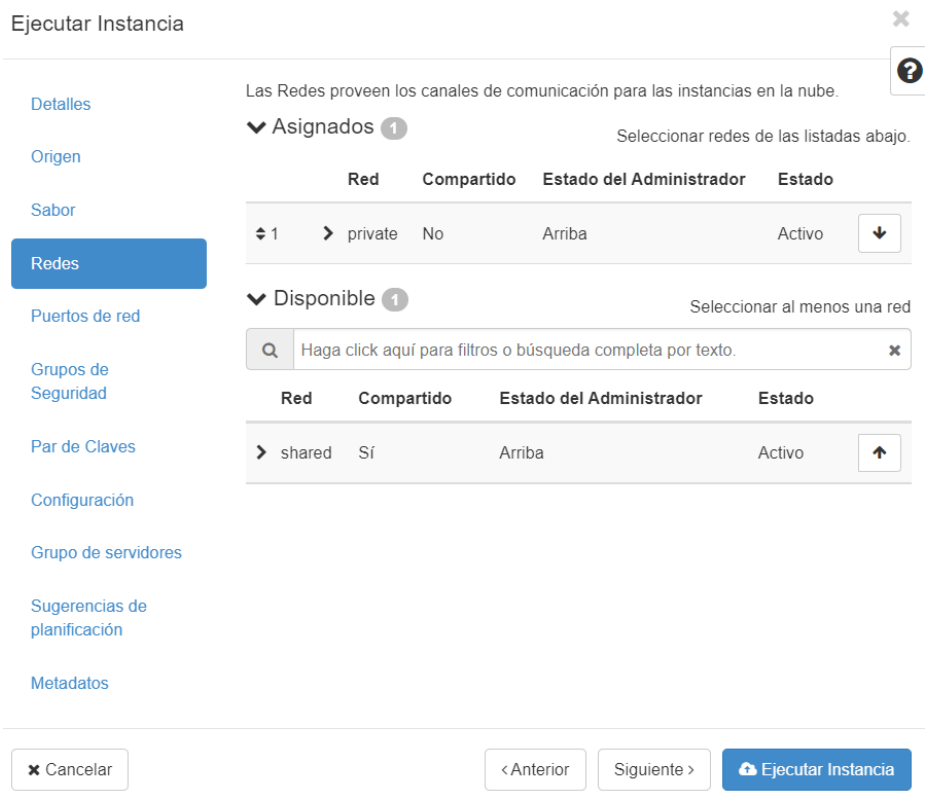

Ilustración 67 Despliegue de una instancia desde Horizon (4)

Por último, en la pestaña "**Par de Claves**" seleccionamos "**Crear Par de Claves**", para poder acceder a la VM por SSH:

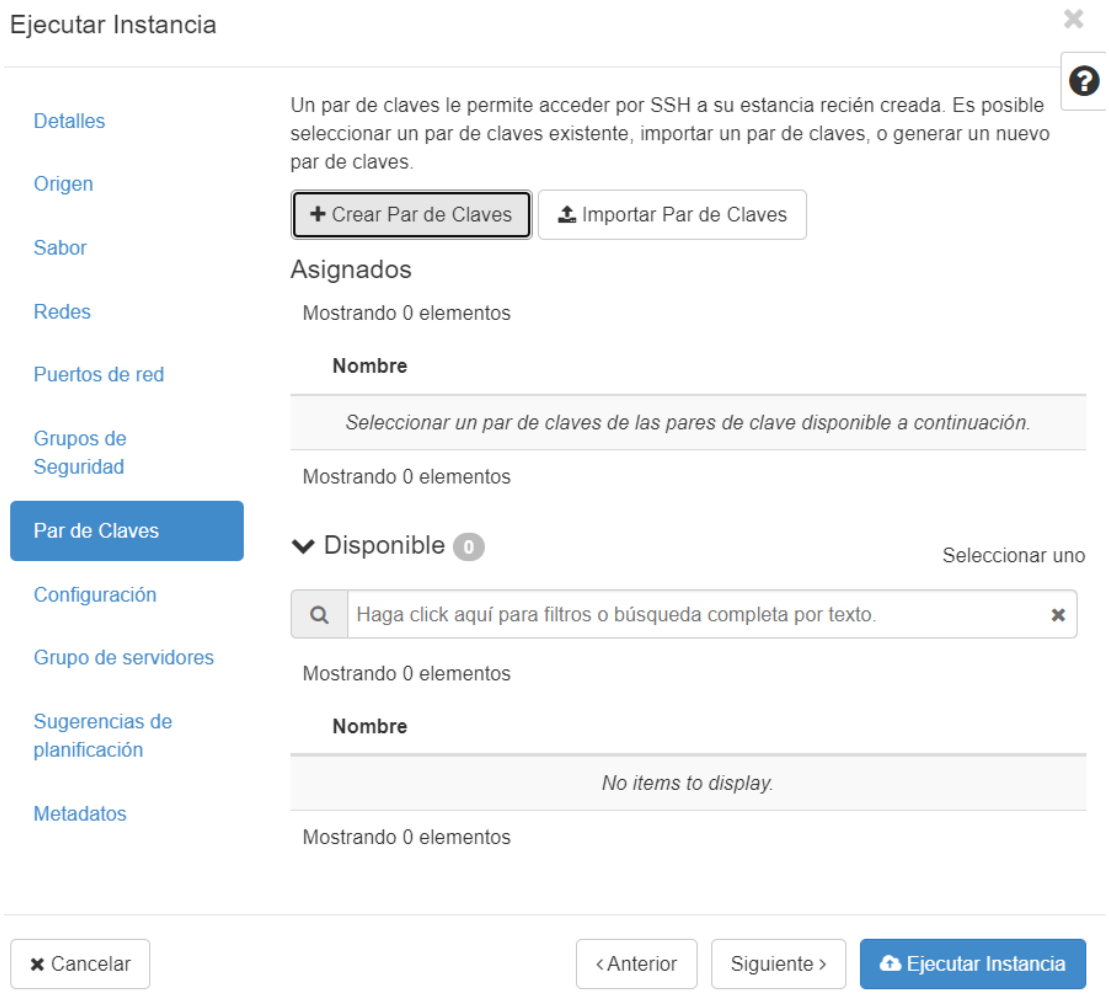

Ilustración 68 Despliegue de una instancia desde Horizon (5)

Introducimos un nombre e indicamos "**Clave SSH**" como "**Tipo de Clave**" en la pestaña emergente:

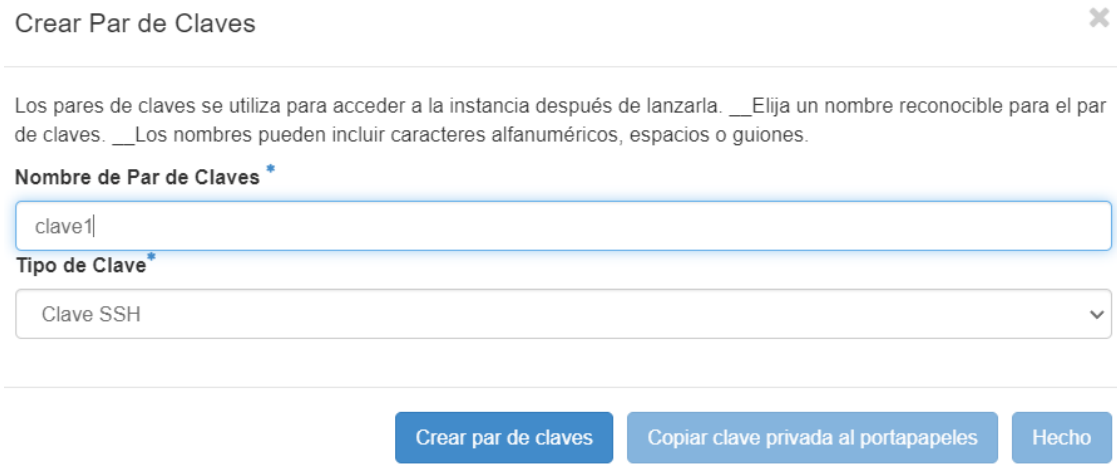

Ilustración 69 Despliegue de una instancia desde Horizon (6)

Al pinchar la opción "**Crear par de claves**", se nos mostrará un nuevo recuadro con la clave privada. Copiamos su contenido a un fichero en el equipo de control y pinchamos en "**Hecho**":

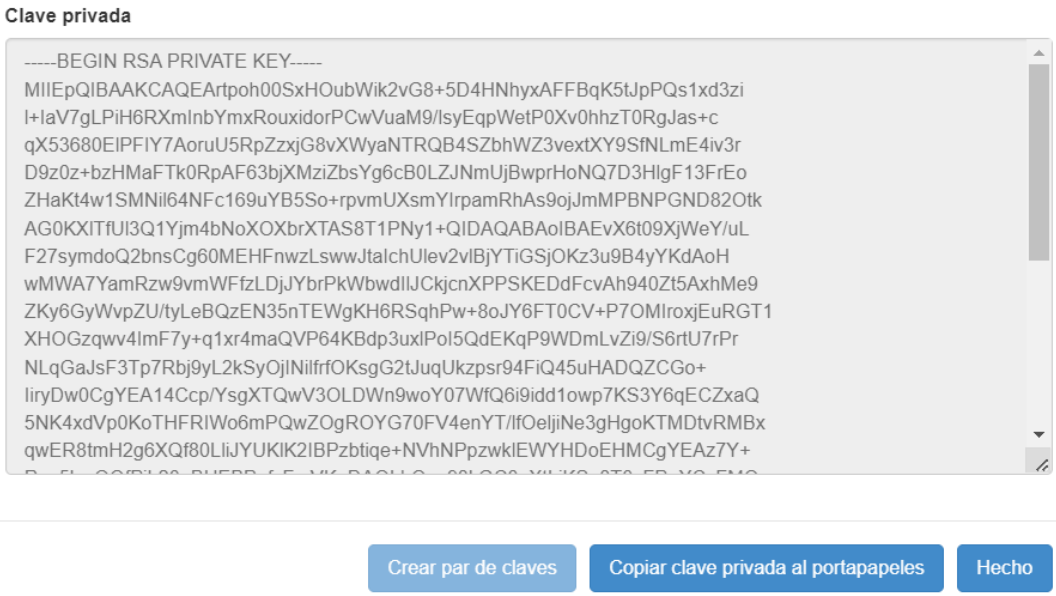

Ilustración 70 Despliegue de una instancia desde Horizon (7)

Para terminar, lanzamos la VM al seleccionar "**Ejecutar Instancia**", ahora, en la sección "**Proyecto -> Computación -> Instancias**" veremos nuestra nueva instancia ejecutándose. Desde aquí le asignaremos una dirección IP flotante a partir de la opción "**Asociar IP flotante**" en el desplegable de la columna "**Actions"**:

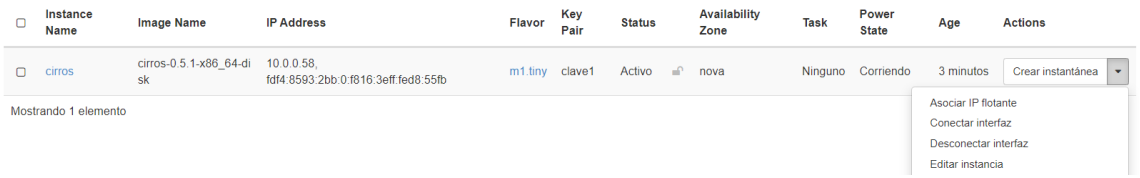

Ilustración 71 Asignación de IP flotante desde Horizon (1)

Se nos abrirá el siguiente recuadro:

Ilustración 72 Asignación de IP flotante desde Horizon (2)

Pinchamos en la opción "**+**", a la derecha del desplegable de "**Dirección IP**". Obtendremos como resultado otro recuadro desde el que seleccionar el pool IPs flotantes que queremos utilizar, en este caso, el de la red **public**:

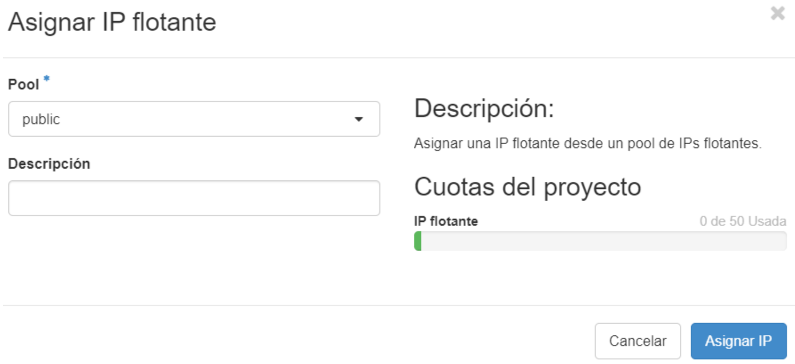

Ilustración 73 Asignación de IP flotante desde Horizon (3)

Una vez seleccionado el pool correspondiente, volvemos al recuadro anterior para elegir una dirección a asignar del desplegable de "**Dirección IP**":

Ilustración 74 Asignación de IP flotante desde Horizon (4)

Habiendo aportado a la instancia una dirección IP de la red del laboratorio para que pueda comunicarse con el exterior de la nube OpenStack, procedemos a conectarnos a la VM vía SSH. Accederemos mediante el fichero, que hemos denominado **clave1.pem**, con la clave privada y la contraseña **gocubsgo**, y verificaremos que somos capaces de realizar un PING a Internet a través del puente **br-ex** que conecta la nube con nuestra red física:

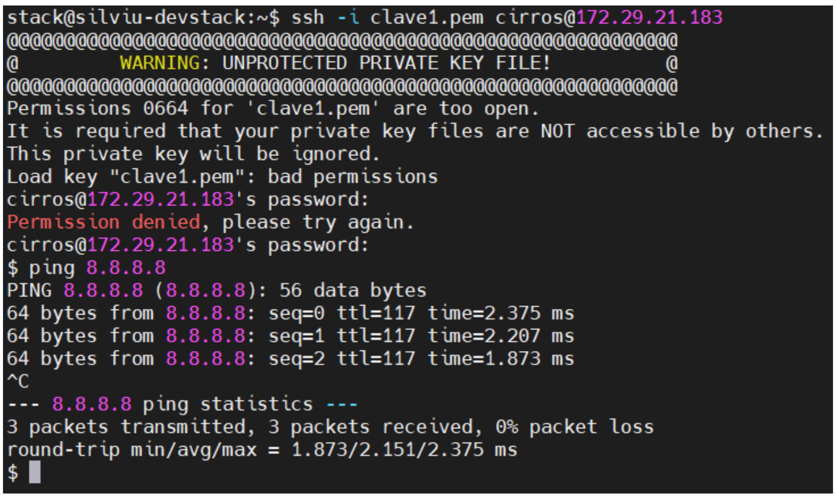

Ilustración 75 Acceso a la instancia y test de salida a Internet

5.2.2. Despliegue DevStack Omni

El despliegue del apartado anterior me ha permitido comprender mejor la configuración y el proceso de instalación de DevStack y así, por ejemplo, saber ubicar con mayor eficacia de donde proviene un problema para resolverlo. Con todo lo aprendido, me he dispuesto a instalar DevStack-Omni utilizando AWS como proveedor de nube pública, sin embargo, ninguno de los intentos realizados a resultado en un despliegue satisfactorio. Aun así, a continuación, expondré los pasos previos necesarios para prepararlo, así como las diferentes soluciones que me han permitido saltarme los errores acontecidos durante una instalación previa.

Creación de las credenciales de acceso a la API de AWS

Para la realización de este despliegue contamos con una cuenta en AWS que nos permita utilizar sus servicios. Será necesario poseer un usuario con acceso ilimitado a dicha cuenta ya que, en caso contrario, no tendremos permiso para crear las credenciales de acceso a la API. En esta ocasión, soy el propietario de la cuenta y no tendremos problema en ese aspecto, pero, de no ser así, sería necesario solicitar permisos para nuestro usuario al administrador de la cuenta o proyecto que estemos utilizando.

En vistas a que los drivers sean capaces de trabajar sobre AWS, crearemos un nuevo usuario IAM (servicio que administra el acceso a los recursos de Amazon Web Services de forma segura) con permiso para acceder a su API. Como resultado de crear dicho usuario, obtendremos una clave de acceso (access key) y una clave secreta (secret key) que, una vez otorgadas al sistema Omni durante el proceso de instalación, le darán la capacidad de realizar llamadas a la API de AWS.

Buscamos el servicio IAM desde la barra de búsqueda en la esquina izquierda superior, o a partir del desplegable de servicios:

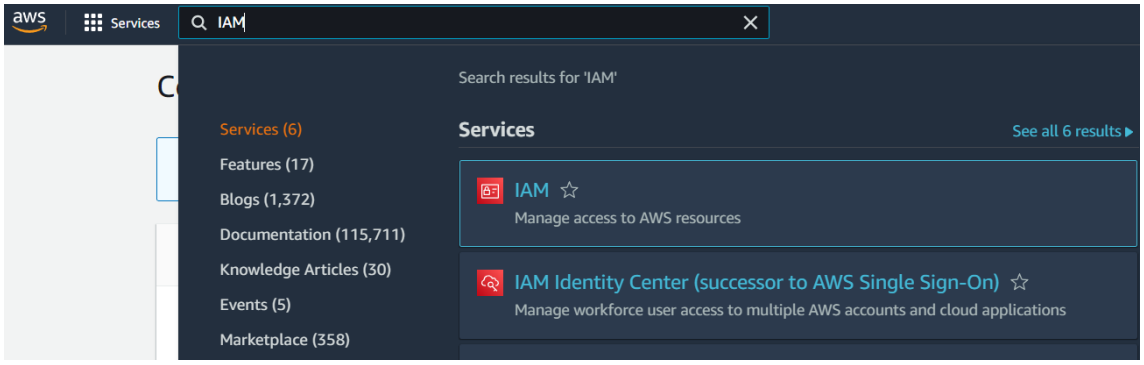

Ilustración 76 Creación de las credenciales de AWS (1)

En el tablero de la izquierda seleccionamos "**Access management -> Users**" y pinchamos en "**Add users**":

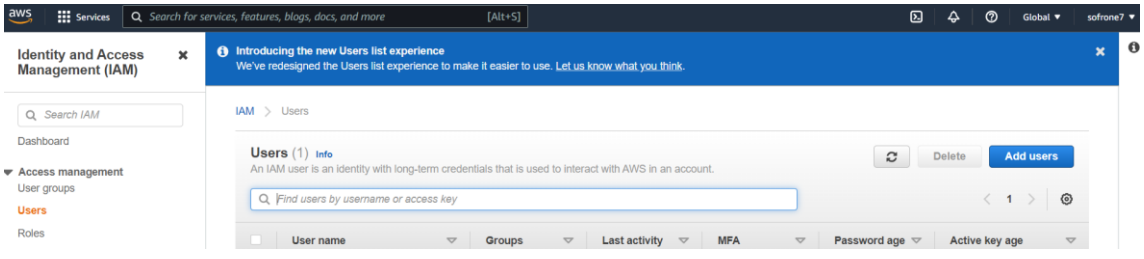

Ilustración 77 Creación de las credenciales de AWS (2)

Indicamos un nombre de usuario (apiuser) y elegimos "**Access key - Programatic access**" como el tipo de credenciales de acceso:

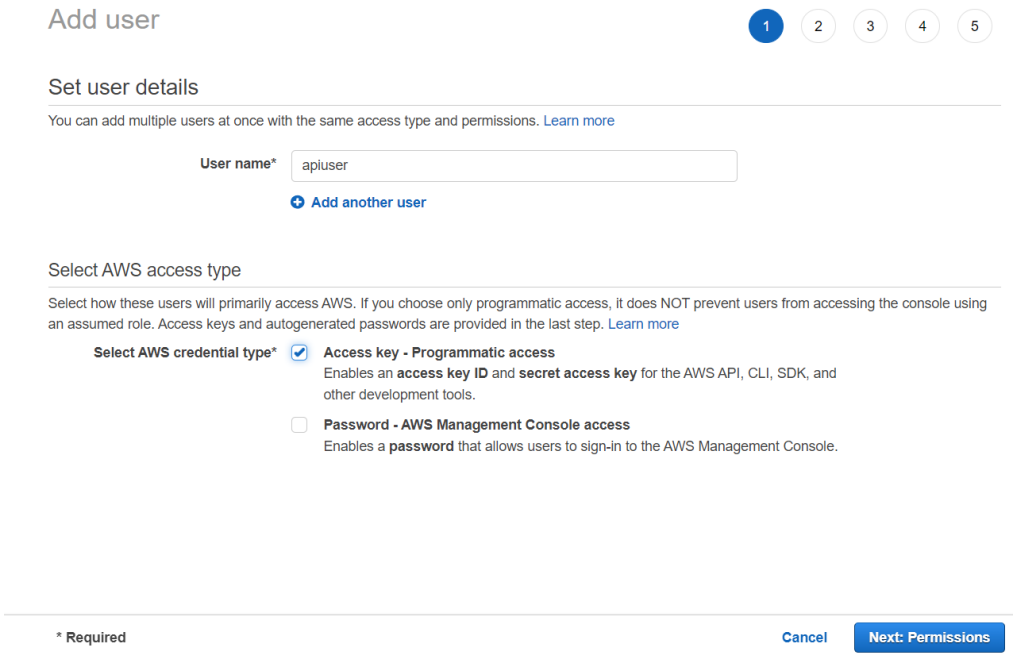

Ilustración 78 Creación de las credenciales de AWS (3)

Pinchamos en "**Next: Permissions**" para acceder a la configuración de los permisos de usuario. Le otorgaremos permisos de administrador seleccionando "**AdministratorAccess**":

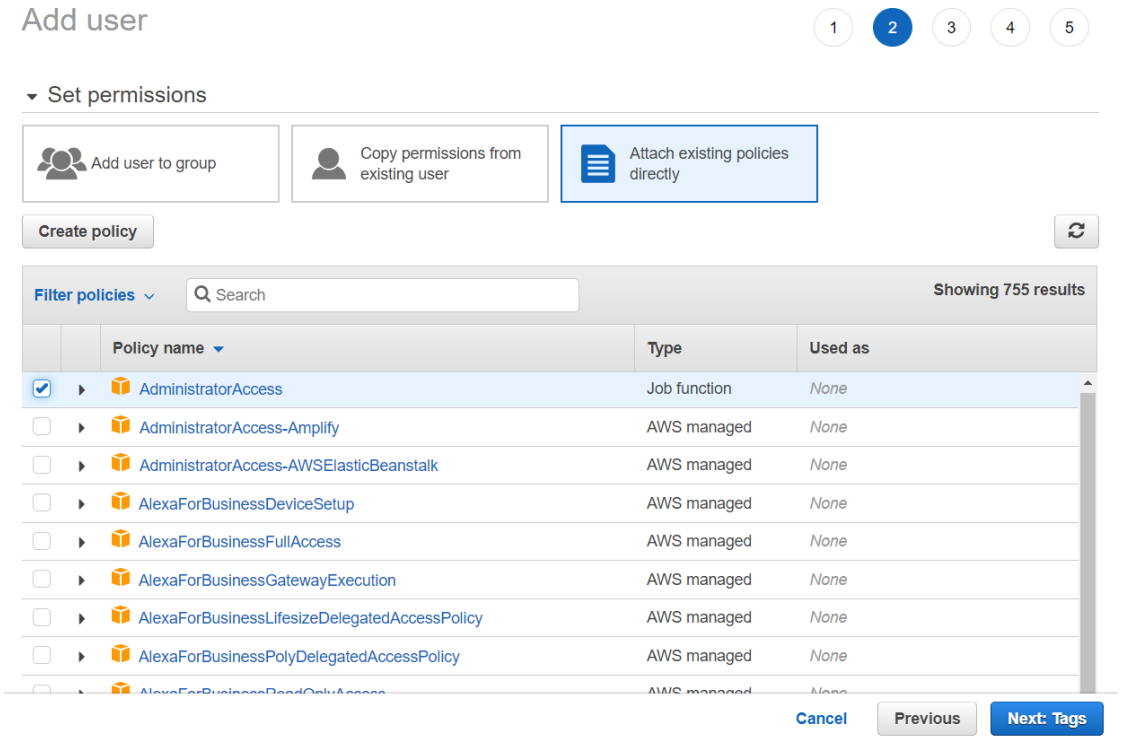

Ilustración 79 Creación de las credenciales de AWS (4)

Podemos saltar directamente al cuarto paso, dónde se nos muestra un resumen con las características del usuario a crear. Comprobamos que todo es correcto y pinchamos en "**Create user**":

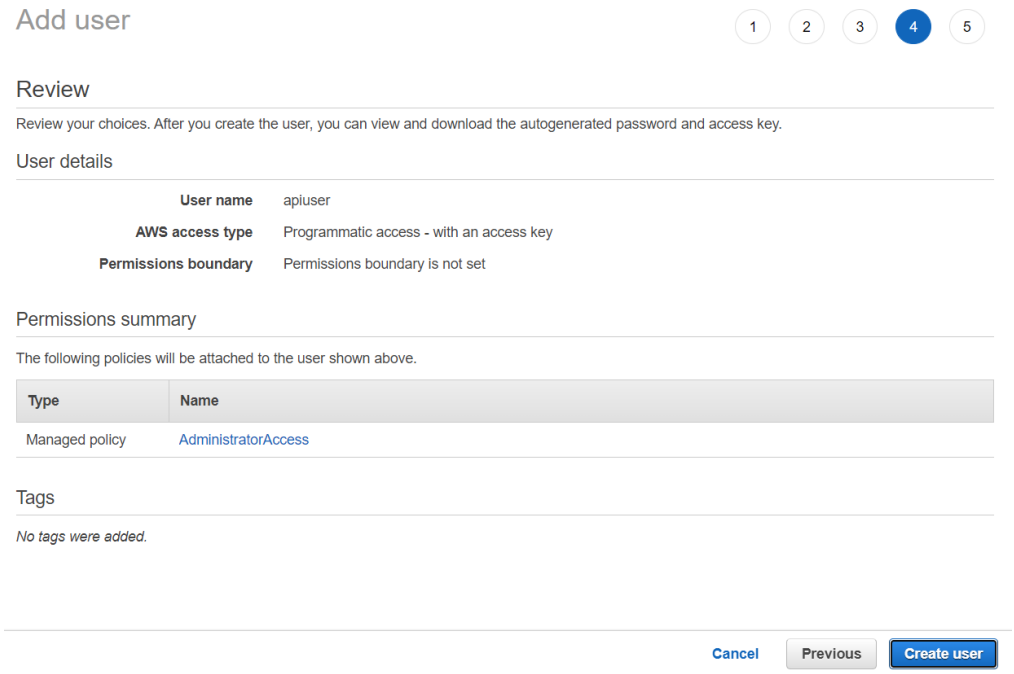

Ilustración 80 Creación de las credenciales de AWS (5)

Una vez creado el usuario, se nos proporciona la clave secreta y de acceso recién generadas:

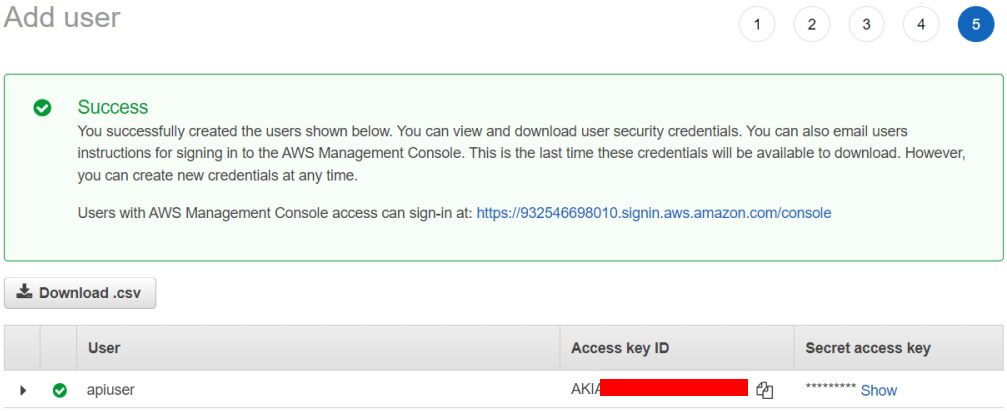

Ilustración 81 Creación de las credenciales de AWS (6)

Las copiamos y almacenamos en un lugar seguro, ya que, si caen en manos de otra persona, podrá acceder a nuestra cuenta asimilando el rol de administrador.

Archivo local.conf

Tras varios intentos, esta es la configuración del fichero **local.conf** que me ha permitido llegar más lejos en el proceso de instalación:

```
[[local|localrc]]
ADMIN_PASSWORD=admin
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
ENABLED_SERVICES+=,g-api,g-reg
enable_plugin omni https://opendev.org/x/omni.git
OMNI_PROVIDER=aws
AWS_SECRET_KEY=[redacted]
AWS_ACCESS_KEY=[redacted]
NEUTRON_CREATE_INITIAL_NETWORKS=False
LOGFILE=$DEST/logs/stack.sh.log
LOGDAYS=2
SWIFT_HASH=66a3d6b56c1f479c8b4e70ab5c2000f5
SWIFT_REPLICAS=1
SWIFT_DATA_DIR=$DEST/data
```

Ilustración 82 Archivo local.conf del despliegue DevStack-Omni

De momento sólo explicaré las opciones que hay que añadir para integrar el plugin de Omni [4] en DevStack, dejando el resto de los detalles de la configuración para el apartado donde hablaremos de la solución de errores:

- **enable_plugin omni https://opendev.org/x/omni.git**: habilitamos el plugin de Omni indicando su nombre y el enlace a su repositorio. Podemos verificar todos los plugins disponibles para DevStack en https://docs.openstack.org/devstack/latest/plugin-registry.html.
- **OMNI_PROVIDER=aws**: establecemos Amazon Web Services como proveedor de nube pública soportado por Omni.
- Credenciales de la API de AWS:
  - **AWS_SECRET_KEY**: clave secreta del usuario **apiuser** configurado en el paso previo.
  - **AWS_ACCESS_KEY**: clave de acceso del usuario **apiuser**.

Solución de problemas

En primer lugar, el error más obvio es no haber limpiado correctamente el entorno OpenStack de la instalación anterior antes de proceder con un nuevo despliegue. Para ello, DevStack pone a nuestra disposición los scripts **unstack.sh** y **clean.sh**, que en conjunto se encargan de apagar las instancias ejecutadas, detener los servicios de OpenStack junto a mysql y rabbitmq, eliminar los archivos de log y limpiar la base de datos, entre otras tareas. La documentación de DevStack especifica que es importante ejecutarlos en el orden correcto, primero **unstack.sh** seguido de **clean.sh**.

Aclarado este punto básico a tener en cuenta, así como el proceso de limpieza que hemos seguido para iniciar cada nuevo despliegue, expondré un conjunto de prerrequisitos para el despliegue, los fallos más relevantes con los que me he topado a través de todo este proceso y las soluciones propuestas para omitirlos al ejecutar la siguiente instalación:

- Asegurarse que contamos con **pip**, **virtualenv** y, mínimo, la versión 3.7 de Python en nuestro equipo, ya que son herramientas necesarias para el despliegue.
- También será necesario tener el módulo Python **config-parser**. Para instalarlo ejecutamos "**pip install config-parser**".
- "**ebtables v1.8.4 (nf_tables): table `broute' is incompatible, use 'nft' tool.**":

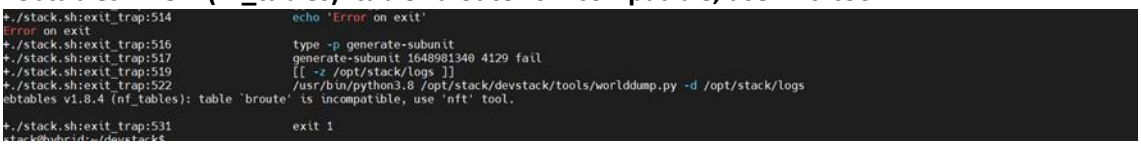

Ilustración 83 Error del despliegue DevStack-Omni (1)

Este error se da debido a que la herramienta **iptables** ha quedado obsoleta, y en su lugar, ha sido sustituida por **nftables**. Atendiendo a las soluciones que se proponen en los foros de DevStack, seguiremos los pasos para cambiar a **iptable-legacy**:

```
apt-get update
apt-get upgrade
apt-get install iptables
apt-get install arptables
apt-get install ebtables

update-alternatives --set iptables /usr/sbin/iptables-legacy || true
update-alternatives --set ip6tables /usr/sbin/ip6tables-legacy || true
update-alternatives --set arptables /usr/sbin/arptables-legacy || true
update-alternatives --set ebtables /usr/sbin/ebtables-legacy || true
```

- "**[ERROR] /home/devstack/devstack/functions-common:216 Failure creating NET_ID for private**": Sucede cuando el despliegue crea la red de proveedor **private** que se configura por defecto en el proyecto **demo**. Si bien este error puede ocurrir por no haber limpiado la nube OpenStack de la instalación anterior, dado que el problema persiste tras borrar el entorno correctamente y que el error está ligado al driver que actúa sobre Neutron (el servicio que se encarga de crear esta red), quitaremos las variables de **local.conf** que configuran la red **private** y hacen referencia a nuestra red física. Es decir, el rango de direcciones IP flotantes que le otorgamos (**FLOATING_RANGE**), la puerta de enlace hacía la red del laboratorio (**PUBLIC_NETWORK_GATEWAY**), etc. Con este paso deberíamos ser capaces de sobrepasar este error, pero, si aun así persiste, en última instancia podemos directamente indicarle al despliegue que Neutron no cree redes iniciales con la opción **NEUTRON_CREATE_INITIAL_NETWORKS=False**.
- "**Failed to restart devstack@g-reg.service: Unit devstack@g-reg.service not found.**":

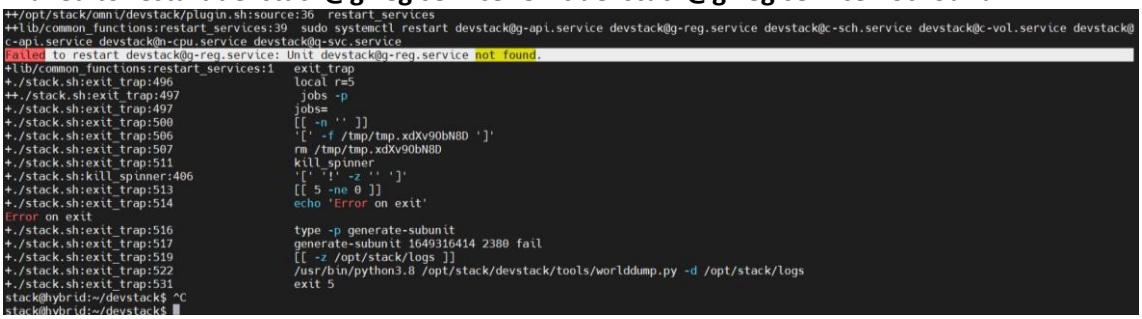

Ilustración 84 Error del despliegue DevStack-Omni (2)

Por alguna razón, los servicios **glance-registry** y, posteriormente, **glance-api** fallan al desplegarse. He sido capaz de corregir esta situación habilitándolos específicamente en el archivo **local.conf**. Para ello, añadimos la opción **ENABLED_SERVICES+=,g-api,g-reg**.

- "**ERROR: Command errores out with exit status 1: python setup.py egg_info Check the logs for full command output.**":

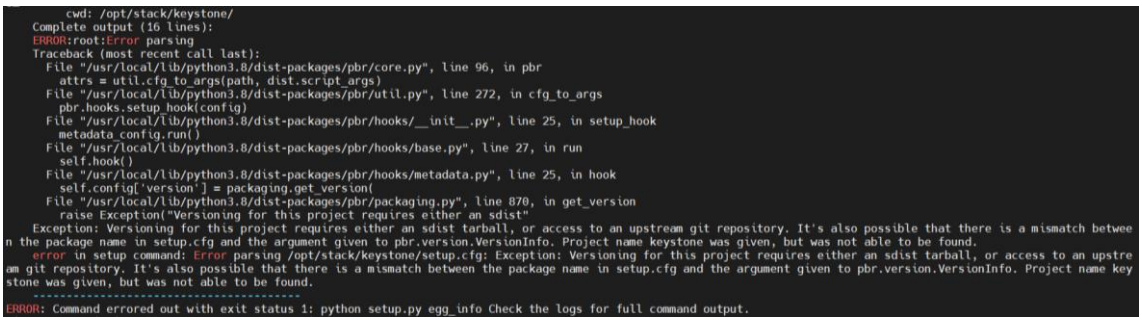

Ilustración 85 Error del despliegue DevStack-Omni (3)

Este es un error meramente ligado a Python y resolverlo nos abrirá paso a través de otros fallos relacionados. Lo encontramos en la biblioteca de proceso de desarrollo de paquetes **setuptools**, y para solucionarlo, simplemente será necesario actualizarla junto a **pip**:

```
pip3 install --upgrade pip setuptools wheel
```

Tras numerosos intentos fallidos en los que nuevos errores han ido aconteciendo, he decidido abandonar esta opción y buscar otras soluciones que me permitan conformar una nube híbrida.

6. Nube híbrida con Juju

Siguiendo con nuestro objetivo de implementar una nube híbrida, nos centraremos en la herramienta Juju [26] de Canonical. Se trata de una plataforma de modelado de aplicaciones de código abierto que nos da la posibilidad de desplegar y orquestar aplicaciones en diferentes entornos sin necesidad de preocuparnos por la configuración del sistema operativo que hay por debajo de ellas. También permite controlar las aplicaciones en la fase de operación. El software necesario se despliega a partir de los denominados *charms* (encantos), que no son más que scripts en formato *yaml*.

La razón por la que nos es útil para nuestro caso es porque, entre los entornos que Juju identifica como válidos para desplegar unidades de aplicación, se encuentran los principales proveedores de nube pública, así como nubes privadas basadas en OpenStack. Esto nos abre la posibilidad de conectar a Juju tanto OpenStack como las nubes públicas que deseemos, y utilizarlo como único punto de gestión de sus recursos y aplicaciones. Rescataremos la nube privada OpenStack que desplegamos con OSA para unirla a Juju junto con AWS y Google Cloud, de esta forma, al menos desde el punto de vista de control, habremos conformado una nube híbrida.

Para entender cómo funcionará la arquitectura de Juju en nuestro modelo, debemos diferenciar entre dos conceptos: cliente Juju y controlador. Se denomina controlador [27] a la instancia inicial que se despliega, mediante el proceso llamado *Bootstrapping*, en cada nube a gestionar por el sistema Juju. Por su parte, el host cliente se encarga, aparte de lanzar dichos controladores, de encomendarles las órdenes de gestión que le indiquemos. Una vez que una tarea le llega a un controlador, realizará los cambios definidos por el cliente Juju a partir de las credenciales de acceso a la API de la nube en la que se ubica que le fueron otorgadas durante el proceso de *Bootstrapping*.

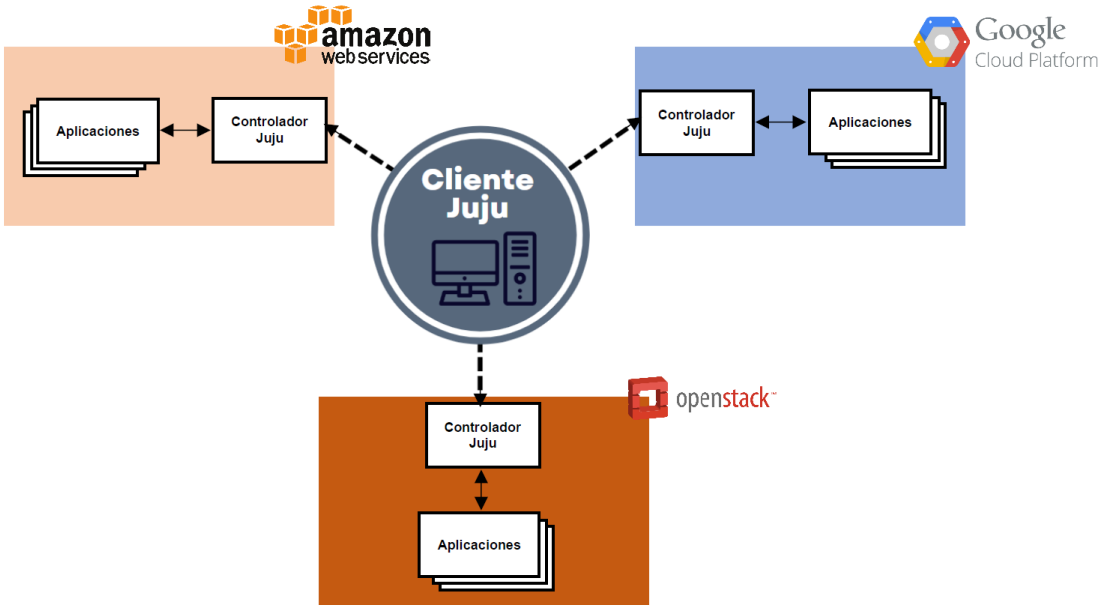

Ilustración 86 Esquema de la nube híbrida con Juju

Para desarrollar este entorno utilizaremos como referencia la información mostrada en el capítulo “Installing the Juju client and bootstrapping clouds” [4] del libro “Hybrid Cloud for Architects”, donde se configura una nube híbrida con OpenStack y AWS, el ejemplo “Hybrid

Cloud with MAAS and JuJu" [28] del usuario Sean Shuping, conformado por las principales nubes públicas disponibles en el mercado, así como la documentación oficial de Juju.

6.1. Instalación de Juju

Trabajaremos desde el nodo de despliegue (172.29.21.201) que utilizamos con OpenStack-Ansible como host cliente Juju para gestionar la nube híbrida. Accedemos a él para añadir el repositorio de Juju al sistema [4], y procedemos con la instalación:

```
$ sudo add-apt-repository ppa:juju/stable

$ sudo apt update

$ sudo apt install juju
```

Una vez se haya instalado Juju en el equipo, podemos verificar las nubes con las que puede trabajar Juju mediante el comando "**juju list-clouds --all**":

```
silviu@control:~$ juju list-clouds --all
You can bootstrap a new controller using one of these clouds...

Clouds available on the client:
Cloud        Regions  Default        Type        Credentials  Source    Description
aws          22       us-east-1      ec2         0            public    Amazon Web Services
aws-china    2        cn-north-1     ec2         0            public    Amazon China
aws-gov      2        us-gov-west-1  ec2         0            public    Amazon (USA Government)
azure        42       centralus      azure       0            public    Microsoft Azure
azure-china  4        chinaeast      azure       0            public    Microsoft Azure China
cloudsigma   12       dub            cloudsigma  0            public    CloudSigma Cloud
equinix      25       px             equinix     0            public
google       25       us-east1       gce         0            public    Google Cloud Platform
localhost    1        localhost      lxd         0            built-in  LXD Container Hypervisor
oracle       4        us-phoenix-1   oci         0            public    Oracle Compute Cloud Service
rackspace    6        dfw            rackspace   0            public    Rackspace Cloud
```

Ilustración 87 Listado de nubes con las que puede trabajar Juju

Como podemos observar, tanto Amazon Web Services como Google Gloud aparecen como nubes a las que Juju sabe conectarse.

6.2. Conexión con OpenStack

Añadir la nube OpenStack al cliente Juju

Para unir nuestra nube privada OpenStack a Juju, primero deberemos definirla con la información referente a los *endpoints* de su API, así como las credenciales de acceso de esta. Estos datos son los que Juju empleará para desplegar su controlador en OpenStack y para poder gestionarla desde dicha instancia. Este es el motivo por el que AWS y GCP se identifican como nubes a las que Juju puede unirse en la salida del comando "**juju list-clouds --all**", la información de sus puntos finales es pública y el equipo de Canonical ya se ha encargado de definirlas, siendo solamente necesario que el usuario proporcione las credenciales privadas de la API de su proyecto AWS o GCP en cuestión. En cambio, en una nube privada sólo el propietario tiene acceso a todos estos datos por lo que, aparte de indicar los requisitos de autentificación, será necesario registrar la nube al sistema Juju.

Atendiendo a la documentación de Juju [29] realizaremos esta tarea a partir del comando "**juju add-cloud**" seguido del nombre con el que vamos a identificar esta nube. Podemos obtener fácilmente los datos de OpenStack que se nos solicitará para esta configuración, así como para la de añadir sus credenciales, a partir del archivo **openrc** de la nube. Recordemos que lo descargamos en Horizon desde "**Proyecto -> Acceso a la API -> Descargar fichero RC de OpenStack**", o al pinchar en el nombre de usuario (**admin**) en la esquina superior derecha y

seleccionar "**Fichero OpenStack RC**". Obtendremos el fichero **admin-openrc.sh** que, al ejecutarlo con el comando **source** e introducir la contraseña de Keystone, nos dará la capacidad de trabajar con la CLI de OpenStack directamente desde el equipo sin necesidad de acceder al contenedor de "utilidades". Podemos listar las variables de entorno referentes a OpenStack que se ha inyectado al sistema con "**env | grep OS_**":

```
silviu@control:~$ env | grep OS_
OS_REGION_NAME=RegionOne
OS_PROJECT_DOMAIN_ID=default
OS_INTERFACE=public
OS_AUTH_URL=https://10.0.123.10:5000
OS_USERNAME=admin
OS_PROJECT_ID=406bc2348d83438aaeea1b6671a49e61
OS_USER_DOMAIN_NAME=Default
OS_PROJECT_NAME=admin
OS_PASSWORD=118c211ab404105ef42466bd3ac34
OS_IDENTITY_API_VERSION=3
```

Ilustración 88 Listado de variables de entorno de la nube OSA

Teniendo en cuenta dichas variables, lanzamos la orden "**juju add-cloud**":

```
silviu@control:~$ juju add-cloud openstack2
This operation can be applied to both a copy on this client and to the one on a controller.
No current controller was detected and there are no registered controllers on this client: either bootstrap one or register one.
Cloud Types
  lxd
  maas
  manual
  openstack
  vsphere

Select cloud type: openstack

Enter the API endpoint url for the cloud [https://10.0.123.10:5000]: https://10.0.123.10:5000/v3

Enter a path to the CA certificate for your cloud if one is required to access it. (optional) [none]: /etc/openstack_deploy/pki/certs/certs/haproxy_control-10
.0.123.10.crt

Auth Types
  access-key
  userpass

Select one or more auth types separated by commas: userpass

Enter region [RegionOne]: RegionOne

Enter the API endpoint url for the region [use cloud api url]: https://10.0.123.10:5000/v3

Enter another region? (y/N): n

Successfully read CA Certificate from /etc/openstack_deploy/pki/certs/certs/haproxy_control-10.0.123.10.crt
Cloud "openstack2" successfully added to your local client.
You will need to add a credential for this cloud (`juju add-credential openstack2`)
before you can use it to bootstrap a controller (`juju bootstrap openstack2`) or
to create a model (`juju add-model <your model name> openstack2`).
```

Ilustración 89 Agregación de la nube OSA

Los datos solicitados son:

- **Cloud type**: el tipo de nube que vamos a añadir a Juju.
- **API endpoint**: la URL del *endpoint* de la API de la nube, en el caso de OpenStack corresponde al punto final de Keystone, que se encarga de gestionar el acceso a los servicios de OpenStack. Entre corchetes se nos muestra un valor por defecto obtenido de la variable de entorno **OS_AUTH_URL**. Si pulsamos el botón **Enter** para continuar con la configuración, se seleccionará dicha opción sin necesidad de introducirla a mano. Sin embargo, en este caso obtendremos un error ya que la dirección completa lleva un "**/v3**" al final, que hace referencia a la versión Keystone utilizada.
- **CA certificate path**: el enlace a Keystone está protegido por SSL/TLS, por tanto, Juju necesitará el certificado de autenticación correspondiente. En OpenStack-Ansible podemos encontrarlo en el directorio **/etc/openstack_deploy/pki/certs/certs/** con el nombre **haproxy_control-*<external_load_balancer_address>*.crt**.

- **Auth type**: define el modo de autentificación. Elegimos la autentificación por usuario, ya que utilizaremos las credenciales del archivo **openrc**.
- **Region**: el nombre de la región donde está ubicada la nube privada.

Una vez añadida la nube OpenStack, saldrá en el listado de "**juju list-clouds --all**". Ahora sólo nos queda la tarea de agregar sus credenciales [30] al cliente Juju. Dado que las variables de entorno cuentan con los datos necesarios para este paso, la forma más rápida y eficiente de hacerlo es indicándole a Juju que escanee dichas variables y cargue de forma automática las credenciales que necesita. Para ello, usamos el comando "**juju autoload-credentials**":

```
silviu@control:~$ juju autoload-credentials
This operation can be applied to both a copy on this client and to the one on a controller.
No current controller was detected but there are other controllers registered: use -c or --controller to specify a controller if needed.
Do you ONLY want to add a credential to this client? (Y/n): y


Looking for cloud and credential information on local client...
ERROR unable to detect local LXC credentials: failed to connect to local LXD: ensuring default bridge config: Not Found

1. openstack region "RegionOne" project "admin" user "admin" (existing, will overwrite)
2. rackspace credential for user "admin" (new)
Select a credential to save by number, or type Q to quit: 1

Select the cloud it belongs to, or type Q to quit [openstack2]:

Saved openstack region "RegionOne" project "admin" user "admin" to cloud openstack2 locally

1. openstack region "RegionOne" project "admin" user "admin" (existing, will overwrite)
2. rackspace credential for user "admin" (new)
Select a credential to save by number, or type Q to quit: q

silviu@control:~$
```

Ilustración 90 Agregación de las credenciales de la nube OSA

Esta acción ha detectado, a partir de las variables de entorno, las credenciales pertenecientes a nuestro proyecto **admin** ubicado en la región **RegionOne**. Las seleccionamos y a la hora de indicar la nube a la que relacionarlas simplemente pulsamos **Enter** para elegir la opción por defecto, la nube **openstack2** recién creada. Verificamos las credenciales asignadas con "**juju credentials --client --format yaml --show-secrets**":

```
openstack2:
  default-region: RegionOne
  admin:
    auth-type: userpass
    domain-name: ""
    password: 118c211ab404105ef42466bd3ac34
    project-domain-name: ""
    tenant-id: 406bc2348d83438aaeea1b6671a49e61
    tenant-name: admin
    user-domain-name: Default
    username: admin
    version: "3"
```

Ilustración 91 Verificación de las credenciales añadidas

De las variables de entorno empleadas podemos identificar el ID (**OS_PROJECT_ID** -> **tenant-id**) y el usuario (**OS_PROJECT_NAME** -> **tenant-name**) del proyecto, la contraseña del entorno (**OS_PASSWORD** -> **password**), entre otras.

Generar los metadatos de la imagen del controlador

Ya hemos dotado al cliente con la información que necesita de la nube OpenStack para crear un controlador Juju, sin embargo, aún nos queda un paso más por hacer. Siguiendo con la

documentación oficial respecto a cómo unir Juju y OpenStack [29], a la hora de desplegar el controlador, se especifica que el cliente Juju necesita acceso a las imágenes de OpenStack para crear la instancia. Esto se debe a que, cuando Juju genera un controlador, emplea los siguientes datos: el identificador de la imagen que se va a utilizar para crear la VM y la URL con la información para configurar el controlador. La documentación [31] nos indica que estos "metadatos" se guardan en un formato JSON llamado *Simplestreams*.

Juju ya cuenta con esta información para la mayoría de las nubes que tiene configuradas por defecto. Pero, para el caso de una nube privada, seremos nosotros los encargados de proporcionarle dichos metadatos en base a nuestra nube. Hay dos formas de hacerlo [31]: almacenando los datos de la imagen en un directorio local o en un sistema de almacenamiento de objetos. Dado que nuestra nube OSA no cuenta con el servicio Swift, nos decantaremos por la opción en local.

El primer paso será crear el directorio destinado a guardar los metadatos que vamos a generar:

```
$ mkdir -p ~/simplestreams/images
```

Elegimos una imagen que tengamos definida en Glance:

```
root@infra1-utility-container-23297593:/# openstack image list
+--------------------------------------+----------+--------+
| ID                                   | Name     | Status |
+--------------------------------------+----------+--------+
| 19e48025-514a-41f0-b970-a22ec716f77f | ubuntu16 | active |
+--------------------------------------+----------+--------+
```

Ilustración 92 Listado de las imágenes de la nube OSA

En nuestro caso sólo tenemos una, así que copiamos su identificador y a continuación generamos los datos para dicha imagen con el siguiente comando:

```
$ juju metadata generate-image -d ~/simplestreams -i 19e48025-514a-41f0-b970-a22ec716f77f
-s xenial -r RegionOne -u https://10.0.123.10:5000/v3
```

Los *flags* empleado indican:

- **-d**: directorio donde se van a almacenar los metadatos de la imagen.
- **-i**: ID de la imagen.
- **-s**: la serie a la que se refiere la imagen, en este caso **xenial**.
- **-r**: el nombre de la región de la nube. Sabemos que es **RegionOne** de apartados anteriores.
- **-u**: la dirección *endpoint* del servicio Keystone.

Por último, verificamos que en el directorio **simplestreams** se han generado los archivos JSON (Ver anexo) con los metadatos:

```
silviu@control:~$ ls simplestreams/images/streams/v1/
com.ubuntu.cloud-released-imagemetadata.json  index.json
silviu@control:~$
```

Ilustración 93 Verificación del directorio simplestreams

Bootstrap del controlador OpenStack

Ahora sí contamos con todos los requisitos para desplegar un controlador Juju en nuestra nube OpenStack. La forma de hacerlo es mediante el comando "juju bootstrap", seguido del nombre que le hemos puesto a la nube al añadirla (openstack2) y opciones adicionales [29]:

```
$ juju bootstrap openstack2 --verbose --constraints "mem=1024M" --metadata-source
/home/silviu/simplestreams/images --bootstrap-series=xenial --config network=49ffb60b-
d007-4f21-938b-554bb51e4dd3 --config use-floating-ip=true --debug
```

A continuación, explicaré las opciones del despliegue:

- **--constraints**: establece restricciones al modelo del controlador. En este caso, hemos especificado que cuente con 1024 Mb de RAM.
- **--metadata-source**: ruta en la que se encuentran almacenados los metadatos de la imagen a utilizar.
- **--bootstrap-series**: elegimos **xenial** como serie de la máquina.
- **--config network**: indica una o más opciones de configuración de la instancia:
  - **network=49ffb60b-d007-4f21-938b-554bb51e4dd3**: especificamos el ID de la red donde queremos que la VM del controlador se lance. Elegimos la red de proyecto **interna**:

```
root@infra1-utility-container-23297593:/# openstack network list
+--------------------------------------+---------+--------------------------------------+
| ID                                   | Name    | Subnets                              |
+--------------------------------------+---------+--------------------------------------+
| 49ffb60b-d007-4f21-938b-554bb51e4dd3 | interna | 27e5c257-802a-4906-934a-3b23afc97968 |
| 5b9967f6-5295-4aa7-8080-f336c8465e1d | externa | 177ed6ee-c486-48bb-9523-23b303e5e2e5 |
+--------------------------------------+---------+--------------------------------------+
```

Ilustración 94 Listado de redes de la nube OSA

  - **use-floating-ip=true**: le asignaremos una dirección IP flotante del pool de direcciones de la red **externa** para poder acceder a la interfaz web Juju del controlador y comunicarnos con él desde el exterior.

Cabe destacar que Juju ya se encarga de generar un grupo nuevo con las reglas que considera pertinentes para el controlador. Aunque podemos asociarle el grupo de seguridad por defecto del proyecto con el flag "**--config use-default-secgroup=true**".

Una vez el despliegue se ha completado satisfactoriamente la salida del comando nos indica el controlador con nombre **openstack2-regionone** está habilitado:

```
09:24:17 INFO  cmd bootstrap.go:613 Bootstrap agent now started
09:24:17 DEBUG juju.provider.openstack provider.go:453 instance 3c7ef780-252f-4640-aa70-ee64436f4148 has floating IP address: 172.29.21.187
09:24:17 INFO  juju.juju api.go:302 API endpoints changed from [] to [172.29.21.187:17070 10.0.125.29:17070]
09:24:17 INFO  cmd controller.go:89 Contacting Juju controller at 172.29.21.187 to verify accessibility...
09:24:17 INFO  juju.juju api.go:67 connecting to API addresses: [172.29.21.187:17070 10.0.125.29:17070]
09:24:23 DEBUG juju.api apiclient.go:1092 successfully dialed "wss://172.29.21.187:17070/model/b18dc04b-e38c-4165-824e-4cb4e9e59864/api"
09:24:23 INFO  juju.api apiclient.go:624 connection established to "wss://172.29.21.187:17070/model/b18dc04b-e38c-4165-824e-4cb4e9e59864/api"
09:24:24 INFO  juju.juju api.go:302 API endpoints changed from [10.0.125.29:17070 172.29.21.187:17070] to [172.29.21.187:17070 10.0.125.29:17070]
09:24:24 DEBUG juju.api monitor.go:35 RPC connection died
09:24:24 INFO  cmd controller.go:100
Bootstrap complete, controller "openstack2-regionone" is now available
Controller machines are in the "controller" model
09:24:24 INFO  cmd bootstrap.go:535 Initial model "default" added
09:24:24 INFO  cmd supercommand.go:525 command finished
```

Ilustración 95 Fin del proceso de bootstrapping

Podemos comprobarlo con el comando "**juju controllers**", que nos lista todos los controladores configurados en el cliente Juju y su estado:

```
silviu@control:~$ juju controllers
Use --refresh option with this command to see the latest information.

Controller            Model    User   Access     Cloud/Region          Models  Nodes    HA  Version
openstack2-regionone*  default  admin  superuser  openstack2/RegionOne       2      1  none  2.7.6
```

Ilustración 96 Listado de controladores del cliente Juju

Y también en OpenStack, dónde identificamos una nueva instancia correspondiente al controlador Juju:

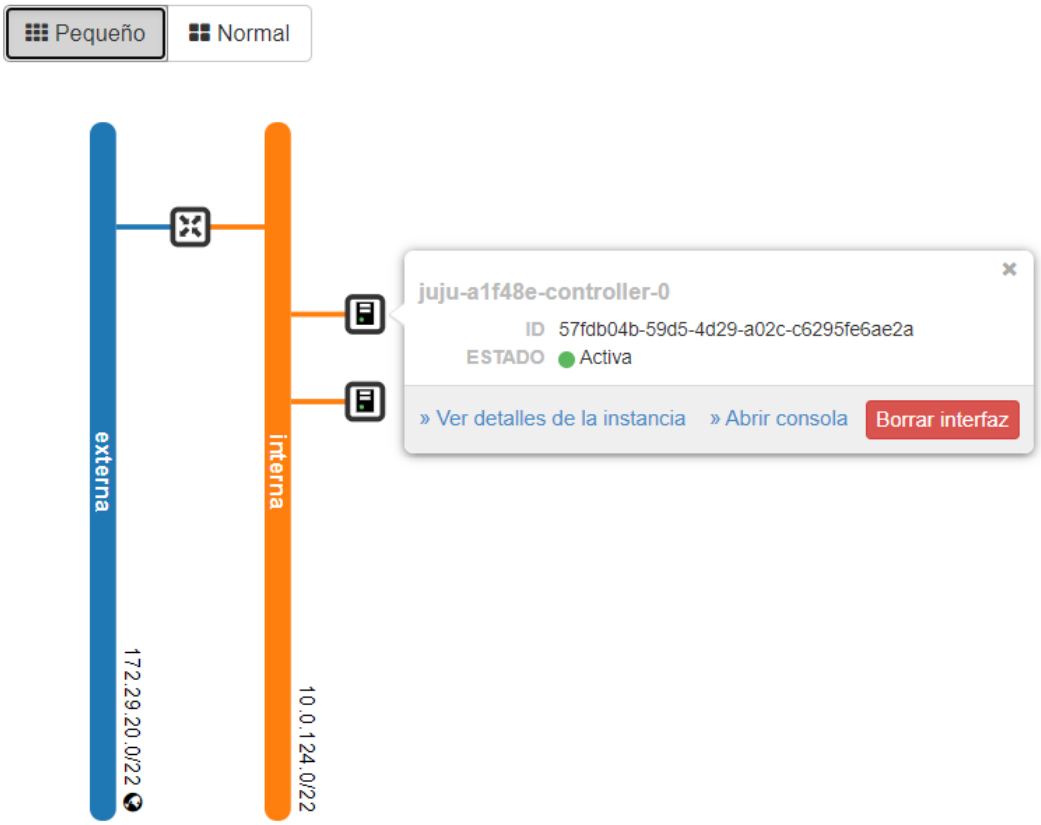

Ilustración 97 Controlador Juju en la nube OSA

Llegados a este punto, desplegar una aplicación dentro de la nube es tan sencillo como lanzar el comando "**juju deploy**" seguido del nombre de su *charm*, esto desplegará por defecto una nueva instancia y el "encanto" se encargará de configurar directamente la aplicación dentro de él. En https://charmhub.io/ podemos encontrar todos los *charms* para ejecutar aplicaciones que tiene a su disposición Juju. Por ejemplo, con el siguiente comando le encomendamos al controlador de OpenStack la tarea de ejecutar una instancia en la nube con el sistema de gestión de base de datos PostgreSQL:

```
silviu@control:~$ juju deploy postgresql
Located charm "cs:postgresql-246".
Deploying charm "cs:postgresql-246".
silviu@control:~$
```

Ilustración 98 Comando para el despliegue de PostgreSQL

En la interfaz web de Juju se puede verificar el icono de la aplicación desplegada:

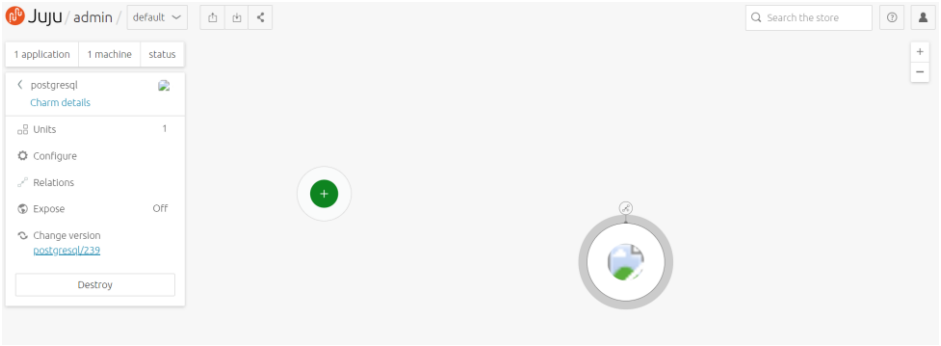

Ilustración 99 Interfaz web de Juju

Hemos obtenido el enlace y las credenciales para autentificarnos en la interfaz web mediante el comando "**juju gui**":

```
silviu@control:~$ juju gui
GUI 2.15.3 for model "admin/default" is enabled at:
  https://172.29.21.187:17070/gui/u/admin/default
Your login credential is:
  username: admin
  password: bc987a2ff15eb6edd3b0eea3683cea4f
```

Ilustración 100 Salida del comando "juju gui"

Observamos que la GUI reside en una dirección formada por la IP flotante del controlador y el puerto 17070. Con estos datos, conformamos un túnel SSH mediante MobaXterm aplicando lo aprendido en el capítulo "Pruebas iniciales del funcionamiento de OpenStack", para poder acceder a la interfaz web desde nuestro ordenador personal:

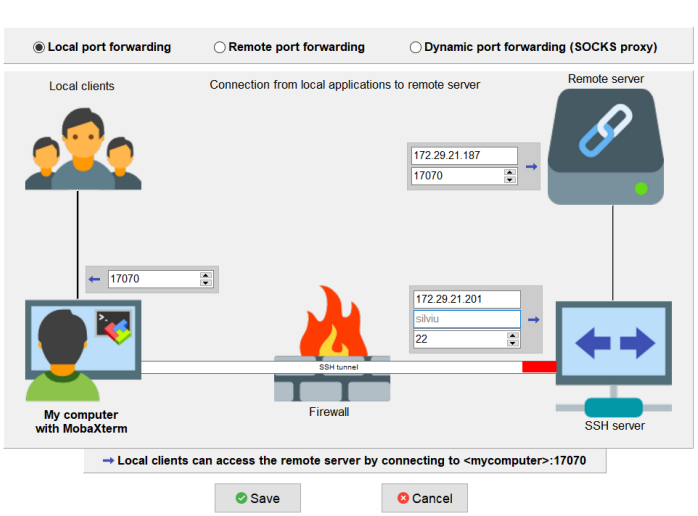

Ilustración 101 Túnel SSH con el panel de control de Juju

6.3. Conexión con Amazon Web Services

Agregar las credenciales al cliente Juju

Como vimos en el capítulo anterior, Amazon Web Services es una de las nubes previamente configuradas para el acceso mediante Juju. Por tanto, solo tendremos que realizar la tarea de añadir las credenciales correspondientes a la API de Amazon y ya seremos capaces de desplegar

un controlador. Recordemos que ya contamos con las credenciales del usuario **apiuser**, que creamos para el despliegue de DevStack con Omni, así que las utilizaremos directamente al ejecutar "**juju add-credential aws**" [32], donde **aws** especifica el nombre que le damos a la nube a configurar:

```
Enter credential name: juju-aws

Regions
  us-east-1
  us-east-2
  us-west-1
  us-west-2
  ca-central-1
  eu-west-1
  eu-west-2
  eu-west-3
  eu-central-1
  eu-north-1
  eu-south-1
  af-south-1
  ap-east-1
  ap-south-1
  ap-southeast-1
  ap-southeast-2
  ap-southeast-3
  ap-northeast-1
  ap-northeast-2
  ap-northeast-3
  me-south-1
  sa-east-1

Select region [any region, credential is not region specific]: eu-west-1

Using auth-type "access-key".

Enter access-key: AK[redacted]

Enter secret-key:

Credential "juju-aws" added locally for cloud "aws".
```

Ilustración 102 Agregación de las credenciales de AWS al cliente Juju

Introducimos un nombre con el que identificar las credenciales (**juju-aws**), una región perteneciente a AWS (**eu-west-1**) y agregamos la clave de acceso y la clave secreta del usuario **apiuser**.

Bootstrap del controlador AWS

Para iniciar la instancia simplemente ejecutamos "**juju bootstrap**" seguido del nombre con el que identificamos nuestra nube de Amazon (**aws**) y el que queremos darle al controlador (**aws-controller**):

```
silviu@control:~$ juju bootstrap aws aws-controller
Creating Juju controller "aws-controller" on aws/us-east-1
Looking for packaged Juju agent version 2.7.6 for amd64
Launching controller instance(s) on aws/us-east-1...
 - i-094049f9c3807efc6 (arch=amd64 mem=4G cores=2)us-east-1a"
Installing Juju agent on bootstrap instance
Fetching Juju GUI 2.15.3
Waiting for address
Attempting to connect to 54.87.169.4:22
Attempting to connect to 172.31.87.248:22
Connected to 54.87.169.4
Running machine configuration script...
Bootstrap agent now started
Contacting Juju controller at 54.87.169.4 to verify accessibility...

Bootstrap complete, controller "aws-controller" is now available
Controller machines are in the "controller" model
Initial model "default" added
```

Ilustración 103 Salida del comando de despliegue de un controlador en AWS

En el panel de control de Amazon Web Services podemos verificar que la VM correspondiente al controlador se ha creado correctamente:

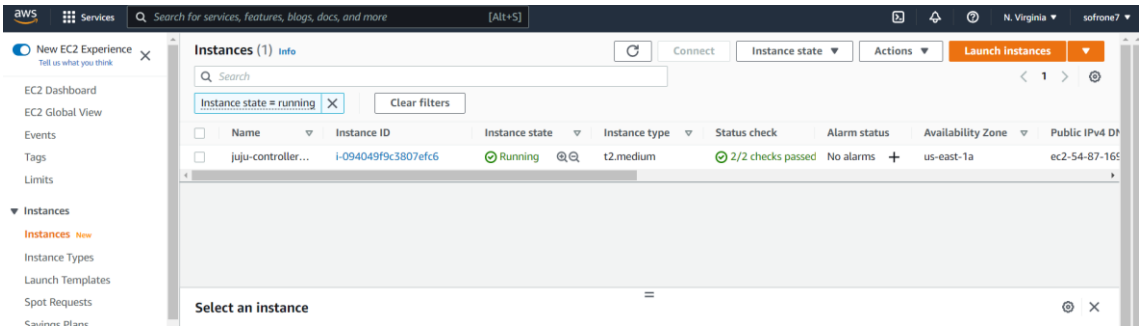

Ilustración 104 Instancia del controlador AWS

Y podemos acceder a su interfaz web a partir de la dirección pública que AWS le ha otorgado:

```
silviu@control:~$ juju gui
GUI 2.15.3 for model "admin/default" is enabled at:
  https://54.87.169.4:17070/gui/u/admin/default
Your login credential is:
  username: admin
  password: d3e9c5179d622c4be47659c1efa8e6ab
```

Ilustración 105 Obtención de las credenciales de acceso al Dashboard de Juju en AWS

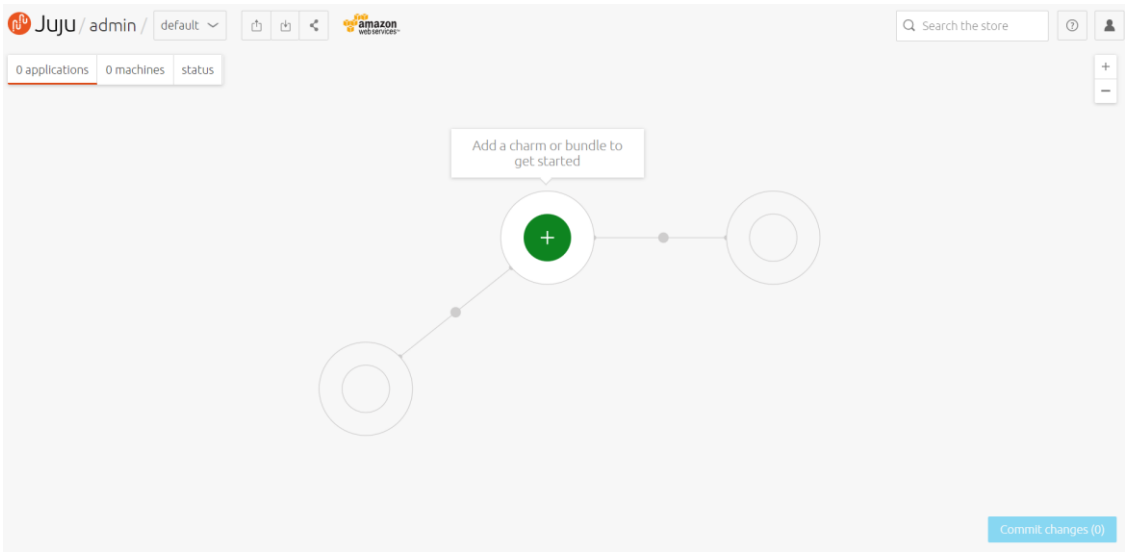

Ilustración 106 Interfaz web de Juju en AWS

Ahora que contamos con dos controladores, es importante conocer el comando "**juju switch**", para cambiar de uno a otro según la nube que queramos configurar. Simplemente lo utilizamos junto al nombre del controlador Juju deseado y verificamos el cambio analizando la salida de "**juju controllers**", que nos indica en cuál estamos ubicado mostrándonos el nombre en color verde con un asterisco:

```
silviu@control:~$ juju controllers
Use --refresh option with this command to see the latest information.

Controller            Model    User   Access     Cloud/Region
aws-controller        default  admin  superuser  aws/us-east-1
openstack2-regionone*  default  admin  superuser  openstack2/RegionOne
-debug                default  admin  superuser  openstack2/RegionOne

silviu@control:~$ juju switch aws-controller
openstack2-regionone:admin/default -> aws-controller:admin/default
silviu@control:~$ juju controllers
Use --refresh option with this command to see the latest information.

Controller           Model    User   Access     Cloud/Region
aws-controller*      default  admin  superuser  aws/us-east-1
openstack2-regionone  default  admin  superuser  openstack2/RegionOne
-debug               default  admin  superuser  openstack2/RegionOne
```

Ilustración 107 Comando para cambiar entre nubes

6.4. Conexión con Google Cloud

Agregar las credenciales al cliente Juju

Para la unión de Juju con GCP, al igual que en el caso de Amazon, sólo será necesario indicarle al cliente las credenciales correspondientes. Primero deberemos crearlas y, para ello, Juju pone a nuestra disposición la documentación necesaria [33] con los pasos a seguir desde la CLI de Google Cloud. Para realizar esta configuración y consumir los recursos de GCP a través del controlador, poseo una cuenta otorgada por la Universidad con créditos suficientes. De no contar con una cuenta, Google también ofrece opciones para acceder a sus servicios de computación gratuita a los usuarios que así lo deseen.

Comenzamos instalando las herramientas del SDK de GCP:

```
$ echo "deb http://packages.cloud.google.com/apt cloud-sdk main" \
  | sudo tee -a /etc/apt/sources.list.d/google-cloud-sdk.list
$ curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
$ sudo apt update && sudo apt install google-cloud-sdk
```

Y las inicializamos mediante:

```
$ gcloud init --console-only
```

La salida del comando nos pedirá elegir un proyecto y una región de GCP predeterminada desde la que trabajar. En mi caso elegiré el proyecto por defecto de mi usuario (**analog-bot-325915**) y la región **europe-west2**:

```
silviu@control:~$ gcloud init --console-only
WARNING: The `--console-only/--no-launch-browser` are deprecated and will be removed in future updates. Use `--no-browser` as a replacement.
Welcome! This command will take you through the configuration of gcloud.

Settings from your current configuration [default] are:
compute:
  region: europe-west2
  zone: europe-west2-c
core:
  account: [redacted]
  disable_usage_reporting: 'True'
  project: analog-bot-325915

Pick configuration to use:
 [1] Re-initialize this configuration [default] with new settings
 [2] Create a new configuration
```

Ilustración 108 Configuración de la herramienta SDK de GCP

Ahora que podemos configurar nuestra cuenta con la CLI de Google desde el nodo de control, la utilizaremos para crear una cuenta de servicio desde la que poder realizar llamadas a la API. Google Cloud define a una cuenta de servicio [34] simplemente como un tipo especial de cuenta utilizada por una aplicación (en nuestro caso Juju) o carga de trabajo, para acceder de forma autorizada a los servicios de los servicios que tenemos a nuestra disposición en la nube. Para poder crearla, en primer lugar, asignaremos el rol “Service Account Key Admin” a nuestro usuario GCP mediante el servicio IAM. Indicaremos el nombre del proyecto y la dirección de correo electrónico del usuario:

```
$ gcloud projects add-iam-policy-binding analog-bot-325915 \
  --member user:s*****7@gmail.com \
  --role roles/iam.serviceAccountKeyAdmin
```

A continuación, crearemos la cuenta de servicio **juju-gce**:

```
$ gcloud iam service-accounts create juju-gce \
  --display-name "Compute Engine Juju service account"
```

Ahora podremos identificarlo en la lista de cuentas de servicio pertenecientes a nuestro proyecto **analog-bot-325915**:

```
silviu@control:~$ gcloud iam service-accounts list --project analog-bot-325915
DISPLAY NAME                            EMAIL                                            DISABLED
Compute Engine default service account  712862630822-compute@developer.gserviceaccount.com  False
App Engine default service account      analog-bot-325915@appspot.gserviceaccount.com     False
Compute Engine Juju service account     juju-gce@analog-bot-325915.iam.gserviceaccount.com  False
gcp                                     gcp-802@analog-bot-325915.iam.gserviceaccount.com  False
silviu@control:~$
```

Ilustración 109 Listado de cuentas de servicio en GCP

Juju gestionará los servicios de Google Cloud a partir de esta cuenta, por tanto, debe contar con los permisos necesarios para hacer esto posible. Dotaremos a la cuenta juju-gce con los roles de 'Compute Instance Admin (v1)' y 'Compute Security Admin', que serán más que suficientes:

```
$ gcloud projects add-iam-policy-binding analog-bot-325915 \
  --member serviceAccount:juju-gce@analog-bot-325915.iam.gserviceaccount.com \
  --role roles/compute.instanceAdmin.v1
$ gcloud projects add-iam-policy-binding analog-bot-325915 \
  --member serviceAccount:juju-gce@analog-bot-325915.iam.gserviceaccount.com \
  --role roles/compute.securityAdmin
```

El siguiente paso será descargar el archivo, que hemos llamado **juju-gce.json**, JSON con las credenciales correspondientes a esta cuenta de servicio:

```
$ gcloud iam service-accounts keys create juju-gce.json \
    --iam-account=juju-gce@analog-bot-325915.iam.gserviceaccount.com
```

Lo almacenaremos en el directorio **~/.local/share/juju/** del cliente Juju:

```
$ mv juju-gce.json ~/.local/share/juju/
```

Por último, será necesario habilitar la API de Compute Engine, encargada de la creación y ejecución de máquinas virtuales en GCP. Pero para ello, primero debemos vincular nuestra cuenta de facturación al proyecto. Necesitaremos el ID de la cuenta de facturación, que encontraremos al listar las cuentas de facturación asociadas con "**gcloud alpha billing accounts list**":

```
silviu@control:~$ gcloud alpha billing accounts list
ACCOUNT_ID            NAME                          OPEN  MASTER_ACCOUNT_ID
01AEFE-59C3D8-AD6024  Billing Account for Education  True
silviu@control:~$
```

Ilustración 110 Listado de cuentas de facturación de GCP

Utilizamos el identificador para vincular la cuenta de facturación a **analog-bot-325915**:

```
$ gcloud alpha billing projects link analog-bot-325915 --billing-account 01AEFE-59C3D8-AD6024
```

Ahora ya seremos capaces de habilitar la API de Compute Engine, así como la API de gestión de IAM, dado que la documentación lo recomienda también:

```
$ gcloud services enable compute.googleapis.com --project analog-bot-325915

$ gcloud services enable iam.googleapis.com --project analog-bot-325915
```

Con todo esto, el paso de agregar las credenciales al sistema Juju es muy parecido al que realizamos para AWS. Ejecutamos "**juju add-credential google**" (siendo **google** el nombre con el que vamos a distinguir esta nube) e introducimos un nombre para las credenciales, una región asociada a GCP y la ruta del archivo JSON que contiene las credenciales (**~/.local/share/juju/juju-gce.json**):

```
Enter credential name: juju-gce

Regions
  us-east1
  us-east4
  us-central1
  us-west1
  us-west2
  us-west3
  us-west4
  asia-east1
  asia-east2
  asia-northeast1
  asia-northeast2
  asia-northeast3
  asia-south1
  asia-southeast1
  asia-southeast2
  australia-southeast1
  europe-central2
  europe-north1
  europe-west1
  europe-west2
  europe-west3
  europe-west4
  europe-west6
  northamerica-northeast1
  southamerica-east1

Select region [any region, credential is not region specific]: europe-west2

Auth Types
  jsonfile
  oauth2

Select auth type [jsonfile]: jsonfile

Enter path to the .json file containing a service account key for your project
(detailed instructions available at https://discourse.jujucharms.com/t/1508).
Path: ~/.local/share/juju/juju-gce.json

Credential "juju-gce" added locally for cloud "google".
```

Ilustración 111 Agregación de las credenciales de GCP

Bootstrap del controlador GCP

Desplegamos el controlador de Google Cloud con "**juju bootstrap**" seguido del nombre que le pusimos a la nube al agregar las credenciales (**google**) y el que queremos para la VM (**google-controller**):

```
silviu@control:~$ juju bootstrap google google-controller
Creating Juju controller "google-controller" on google/us-east1
Looking for packaged Juju agent version 2.7.6 for amd64
Launching controller instance(s) on google/us-east1...
 - juju-525eb5-0 (arch=amd64 mem=3.5G cores=4)
Installing Juju agent on bootstrap instance
Fetching Juju GUI 2.15.3
Waiting for address
Attempting to connect to 34.148.224.162:22
Attempting to connect to 10.142.0.2:22
Connected to 34.148.224.162
Running machine configuration script...
Bootstrap agent now started
Contacting Juju controller at 34.148.224.162 to verify accessibility...

Bootstrap complete, controller "google-controller" is now available
Controller machines are in the "controller" model
Initial model "default" added
silviu@control:~$
```

Ilustración 112 Salida del comando de despliegue de un controlador en GCP

Desde la interfaz web de GCP podemos observar la instancia correspondiente al controlador desplegada:

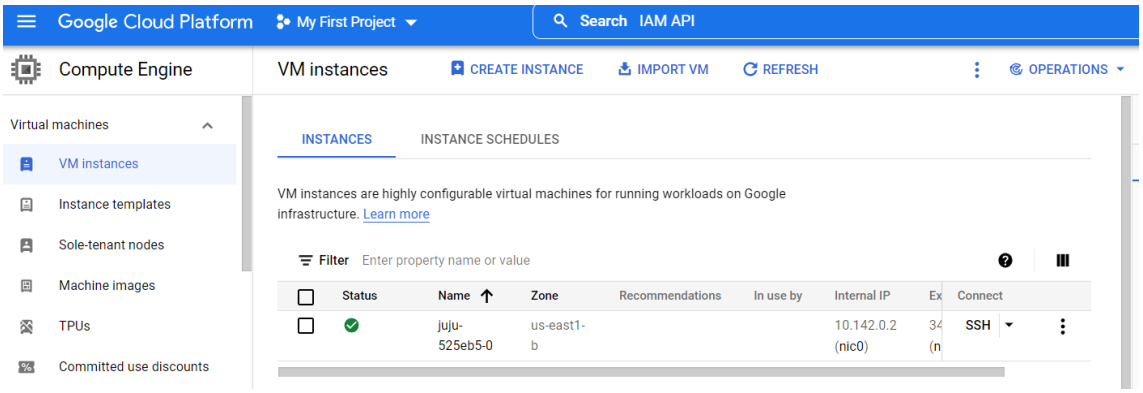

Ilustración 113 Instancia del controlador en GCP

Al igual que en ejemplos anteriores, podemos acceder a la GUI de Juju a partir de la dirección IP pública:

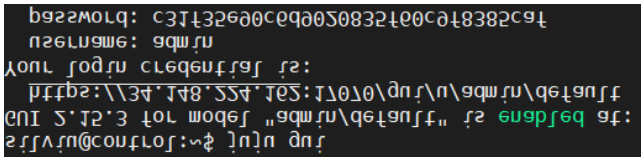

Ilustración 114 Obtención de las credenciales de acceso a la interfaz web Juju de GCP

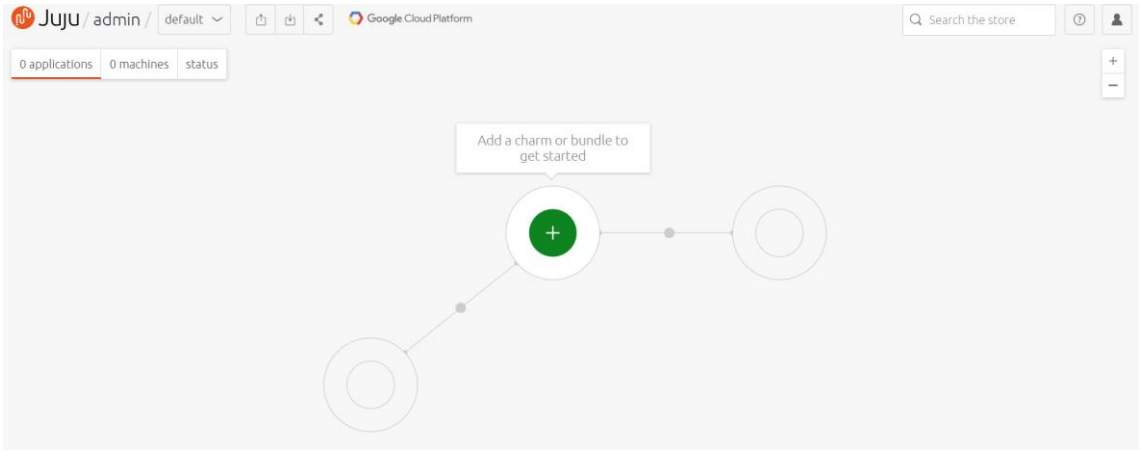

Ilustración 115 Interfaz web de Juju en GCP

Ejemplo final de despliegue de una aplicación en una nube

A lo largo de todo este capítulo, se ha intentado realizar un ejemplo simple con el que desplegar, por ejemplo, un sitio web utilizando la CLI y la GUI de Juju en conjunto. El objetivo es demostrar la facilidad con la que Juju nos permite ejecutar dicha tarea en pocos comandos y sin la necesidad de poseer un conocimiento profundo del software empleado o de su configuración. Sin embargo, los resultados por parte de la GUI no han sido los esperados, debido a una gran cantidad de errores y bugs visuales (como hemos podido ver en el apartado “Conexión con OpenStack”) con los que nos hemos topado. La razón de estos problemas se debe a que, siguiendo los pasos del libro “Hybrid Cloud for Architects”, hemos instalado una versión antigua de Juju (versión 2.7.6), la cual utiliza una interfaz gráfica obsoleta. Las versiones más recientes han abandonado Juju GUI para dar paso a una nueva, llamada JAAS *Dashboard* [35]. Este nuevo

panel de control presenta un mayor enfoque en el uso de la CLI, frente a las opciones interactivas de Juju GUI que permitían, por ejemplo, colocar y relacionar aplicaciones arrastrándolas mediante el uso del ratón.

Para completar este capítulo, hemos vuelto a instalar Juju en su versión más reciente (2.9.34), atendiendo a la documentación oficial [36]:

```
$ sudo snap install juju --classic
```

Hemos configurado de nuevo la nube de Google Cloud y lanzado su controlador correspondiente. A continuación, le indicamos que despliegue el servidor web de código abierto Apache mediante el comando "**juju deploy apache2**". Si accedemos al JAAS *Dashboard* (utilizando "**juju gui**", igual que antes), podemos observar una interfaz renovada y los modelos de las dos instancias que hemos creado; la que contiene la aplicación Apache (**default**) y el controlador (**controller**):

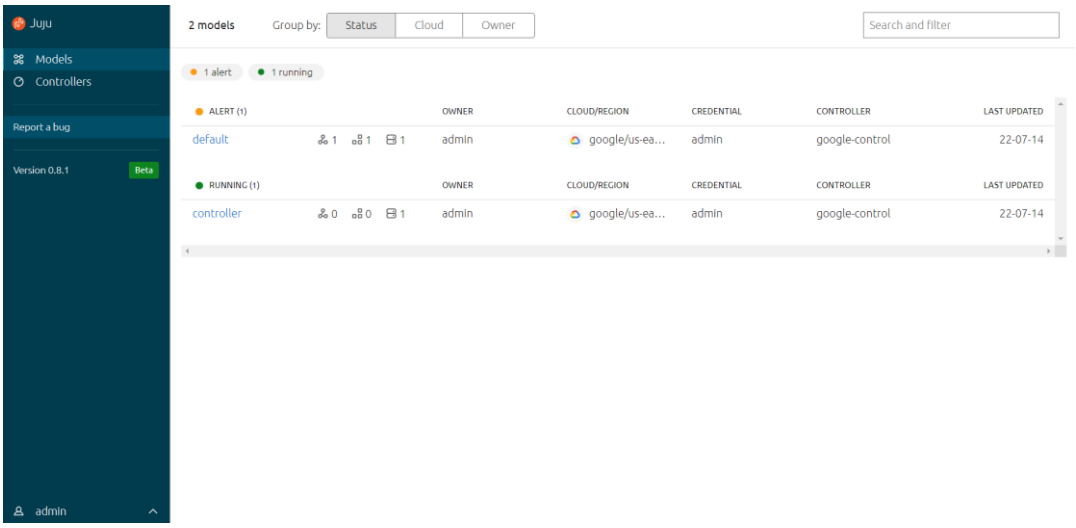

Ilustración 116 Dentro de JAAS Dashboard

Si pinchamos en el modelo **default**, obtendremos información sobre su estado, su dirección IP pública (35.231.241.67) y una CLI web en la parte inferior de la página, desde la que podremos ejecutar directamente los comandos de Juju dentro del modelo:

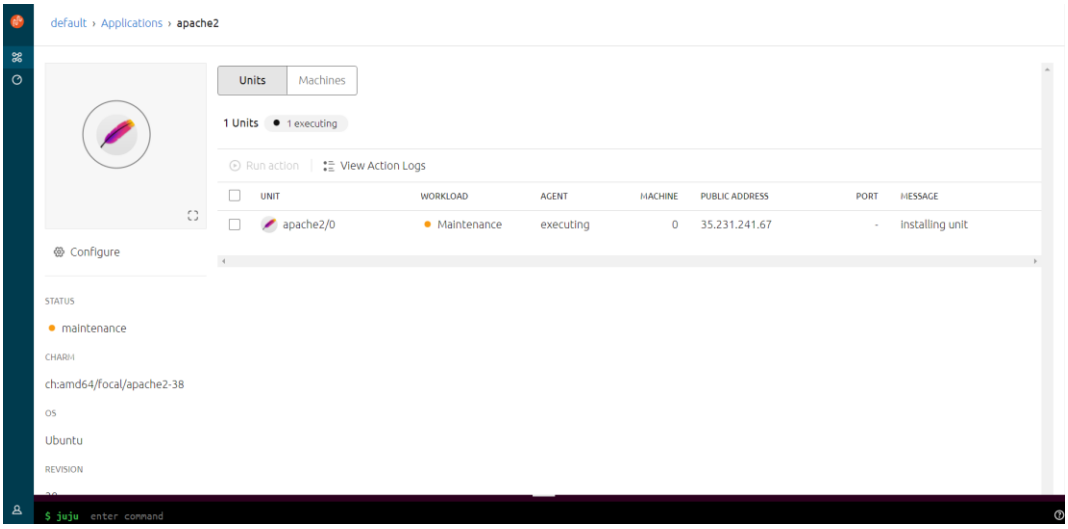

Ilustración 117 Resumen sobre la aplicación Apache desplegada

Una vez que la columna **WORKLOAD** pase a estado activo, indicando que la unidad ya ha terminado con su configuración, ejecutamos "**juju expose apache2**". Este comando simplemente marca la aplicación como "expuesta", un paso necesario antes de configurar las reglas de *firewall* de la instancia que alberga Apache para que sea accesible desde Internet. Ahora, al correr "**juju status**", la columna "**Exposed**" de dicha aplicación debe estar activa:

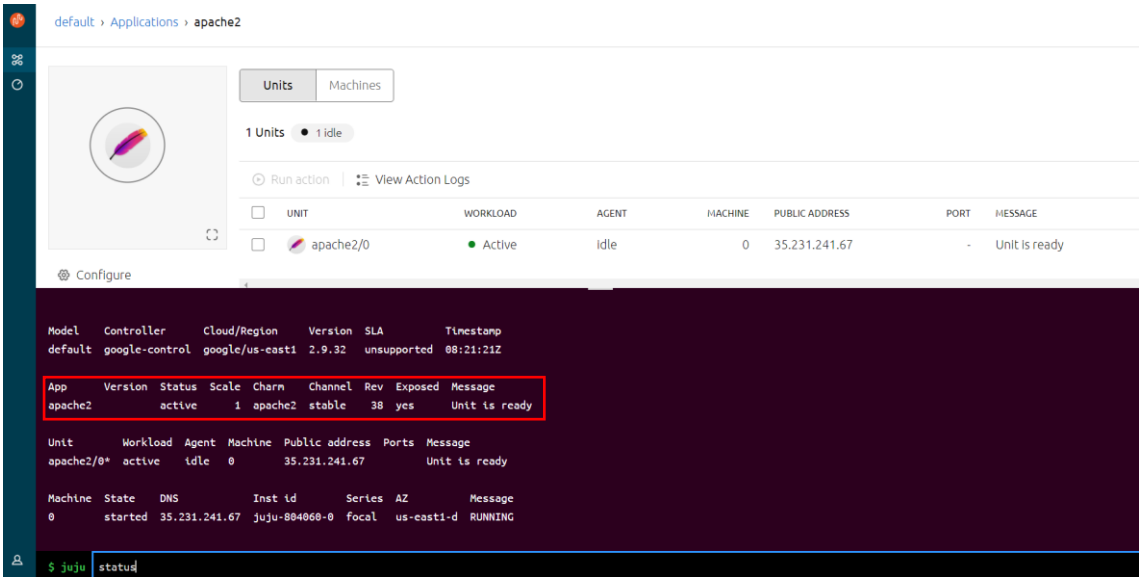

Ilustración 118 Salida del comando "juju status" en JAAS Dashboard

Ahora indicaremos en GCP que se permita el tráfico HTTP en dicha máquina virtual. Para ello nos dirigimos a "**Compute Engine -> VM instances**", buscamos la VM en cuestión mediante la IP pública (35.231.241.67) y pinchamos en su nombre:

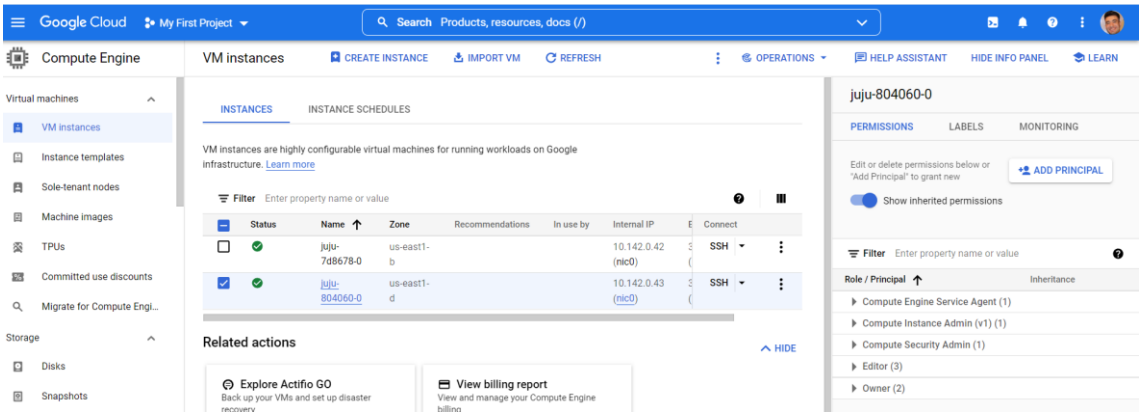

Ilustración 119 Instancias del controlador y la aplicación Apache en GCP

Se nos abrirá una nueva página en la que seleccionaremos "**Edit**" de entre las opciones que se nos muestra en la parte superior. Una vez seleccionado, bajamos hasta la opción "**Firewalls**" y pinchamos en "**Allow HTTP traffic**" y "**Allow HTTPS traffic**":

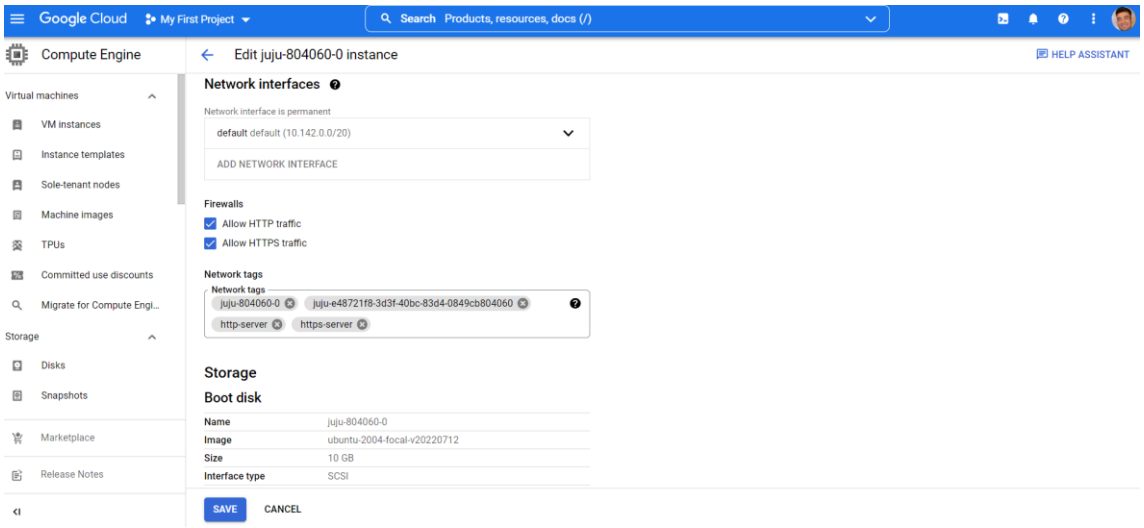

Ilustración 120 Configuración del firewall en GCP

Pinchando sobre la dirección pública de la instancia desde GCP, se nos abrirá una nueva pestaña con la página web por defecto de Apache:

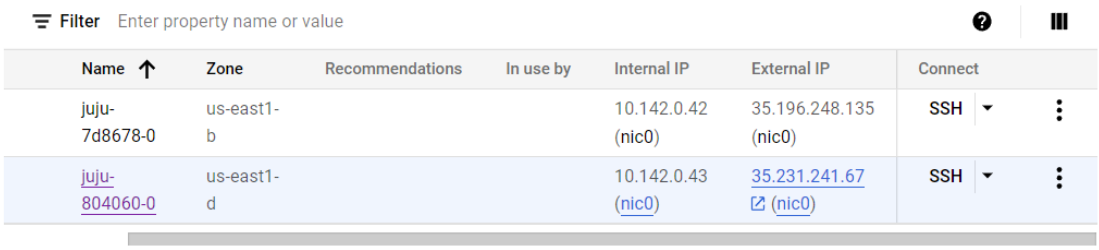

Ilustración 121 Dirección pública de la instancia que alberga Apache

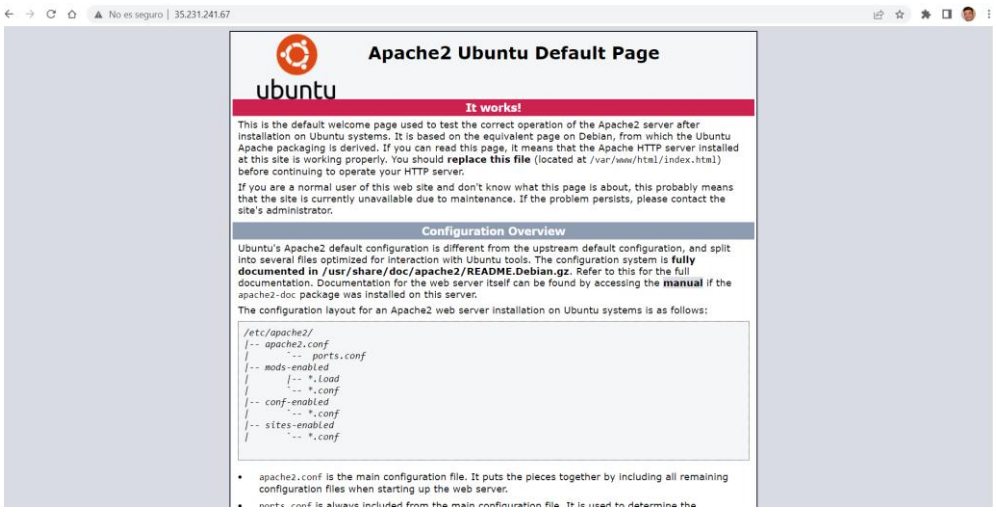

Ilustración 122 Acceso a la página web por defecto de Apache

Por último, para eliminar el controlador y las instancias con las aplicaciones desplegadas a partir de Juju y dejar de consumir recursos y crédito de nuestras nubes, podemos emplear el comando "**juju destroy-controller**" seguido del nombre del controlador y el flag **--destroy-all-models**. En este caso en el que estamos utilizando Google Cloud, quedaría así:

```
$ juju destroy-controller google-controller --destroy-all-models
```

7. Multi-cloud AWS-GCP

Si bien nuestra nube híbrida mediante Juju nos permite gestionar los recursos de cada nube desde un punto de control, no poseemos la capacidad de comunicación entre la nube privada OpenStack y las públicas. Aunque en la teoría existen formas de conectar OpenStack con cualquier nube pública (como muestra el libro "Hybrid Cloud for Architects" [4], que contiene un ejemplo sobre como conectar nuestra red física con AWS mediante conexiones VPN Site-to-site), esto implicaría, en nuestro caso, modificar la configuración del router de la Universidad, al cual no tenemos acceso. Por tanto, aunque no hayamos encontrado una solución factible con respecto a OpenStack, para no dejar la cuestión de la conectividad sin resolver se ha tomado la decisión de enlazar Google Cloud con Amazon Web Services, aprovechando que podemos realizar todo el trabajo de forma virtual. Estableceremos conexiones VPN entre ellas, de esta forma, ambas nubes estarán unidas siendo posible el intercambio de tráfico, formando así, una "multinube" de nubes públicas. Con esto, dentro del entorno con Juju, OpenStack será la única nube sin capacidad para comunicarse con el resto, quedándose sólo con la parte de control:

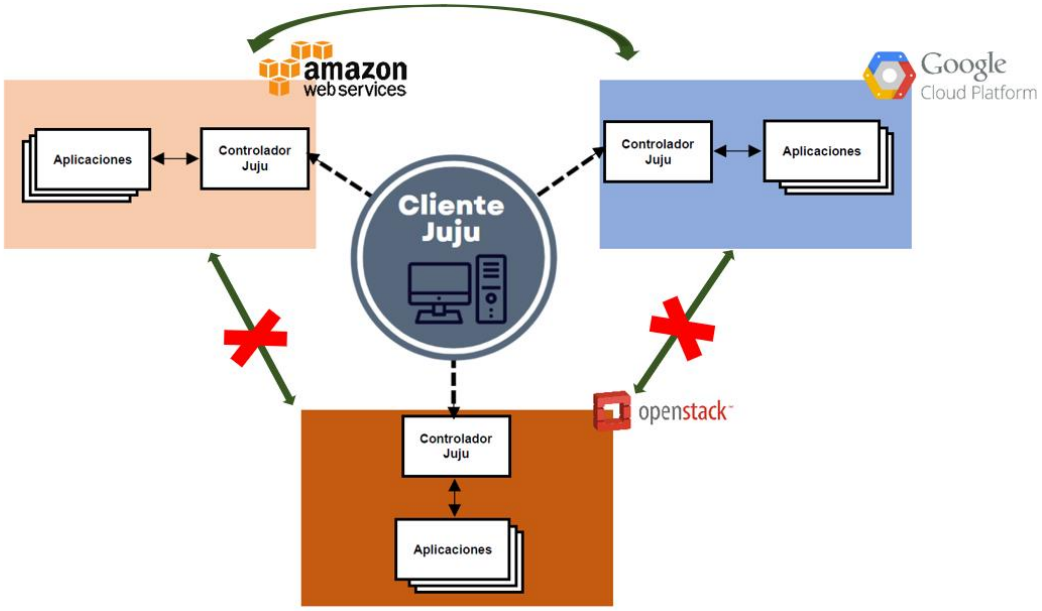

Ilustración 123 Esquema de la comunicación entre las nubes que conforman la nube híbrida con Juju

La multi-cloud básicamente estará formado por dos nubes privadas virtuales (VPCs), una en cada proveedor de nube, con sus respectivas subredes. En el lado de Google Cloud configuraremos una VPN de nube de alta disponibilidad (solución VPN IPSec para la conexión de las VPC de GCP con redes exteriores), mientras que, en AWS, estableceremos dos conexiones VPN Site-to-Site. Para ello, tendremos que asociar la VPC de Google a una puerta de enlace VPN de alta disponibilidad (Cloud HA VPN Gateway) que, con el fin de obtener una disponibilidad del 99.99%, estará compuesta por dos interfaces, cada una con una dirección IP pública asociada. Por tanto, serán necesarias dos conexiones VPN Site-to-Site en AWS, lo que resulta en cuatro direcciones IP externas a conectar con las dos interfaces del Cloud HA Gateway. Esto dará lugar a cuatro túneles VPN con los que, mediante enrutamiento BGP dinámico, seremos capaces de determinar los rangos CIDR de cada VPC y comunicarnos entre ellas. El entorno final resultante tendrá el siguiente aspecto:

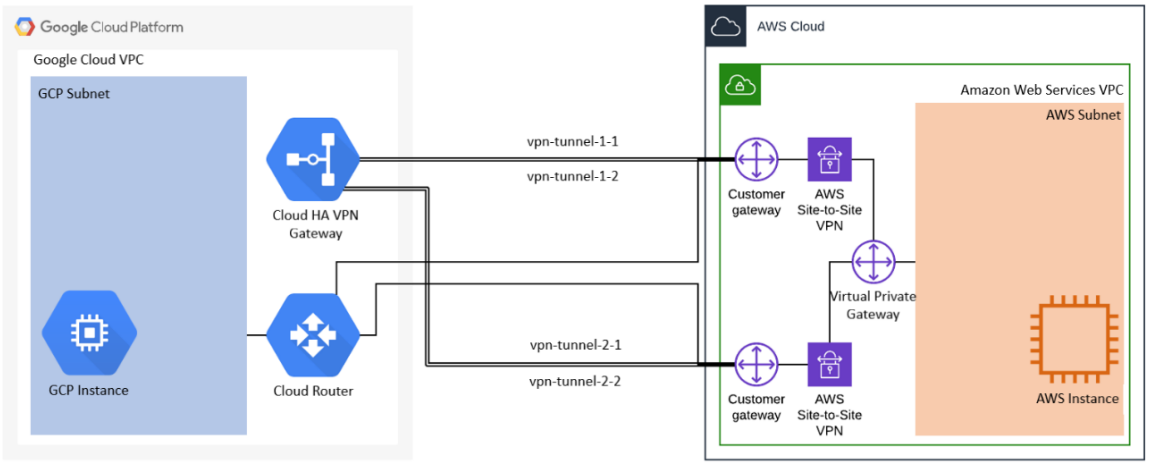

Ilustración 124 Esquema del entorno multi-cloud entre GCP y AWS [39]

Para llevarlo a cabo, seguiremos el tutorial "Build HA VPN connections between Google Cloud and AWS" [37], que tenemos a nuestra disposición en la página oficial de Google Cloud. Dicho tutorial se realiza a partir de instrucciones por línea de comandos, sin embargo, lo vamos a completar por medio de la interfaz web.

7.1. Creación y configuración de una nube privada virtual (VPC) en Google Cloud

En primer lugar, crearemos un nuevo proyecto en GCP. Seleccionamos el nombre del proyecto actual, en la esquina superior izquierda, y picamos en "NEW PROJECT":

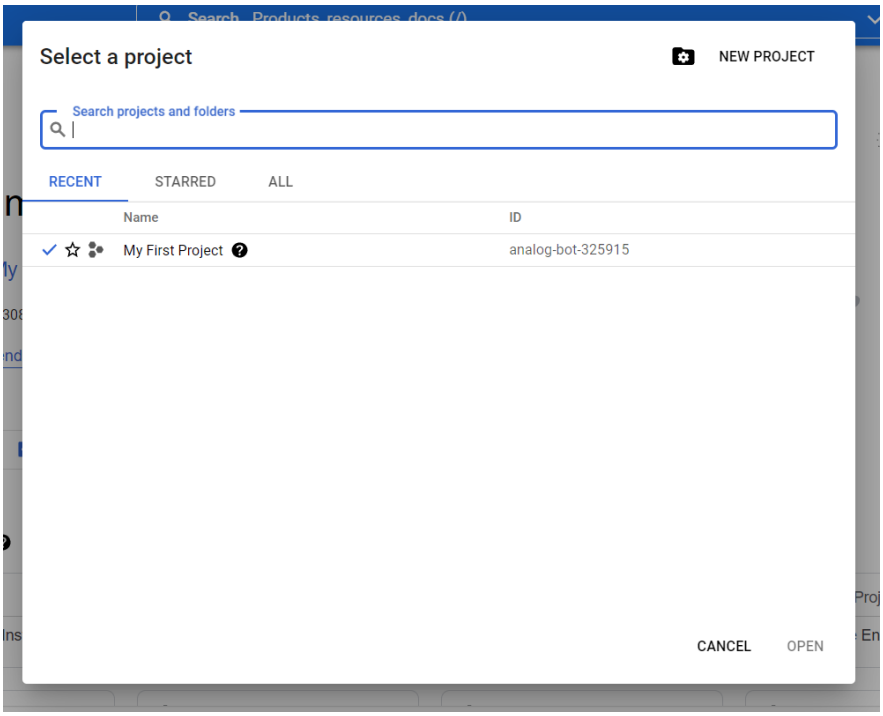

Ilustración 125 Creación de un nuevo proyecto en GCP (1)

En nuestro caso el nombre del proyecto será “aws-gcp-vpn”, el cual seleccionaremos ya que toda la configuración necesaria en GCP la realizaremos él.

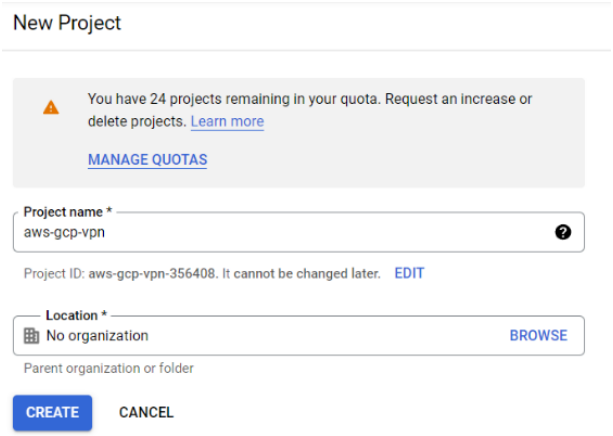

Ilustración 126 Creación de un nuevo proyecto en GCP (2)

Ya dentro del nuevo proyecto crearemos la red VPC correspondiente a Google Cloud, a la que más adelante le asignaremos una puerta de enlace de VPN con alta disponibilidad y un Cloud Router.

Entre los servicios proporcionados en el menú desplegable de la esquina superior izquierda seleccionamos “VPC network”. Nada más entrar nos solicitará que activemos la API encargada de la creación y ejecución de máquinas virtuales, Compute Engine:

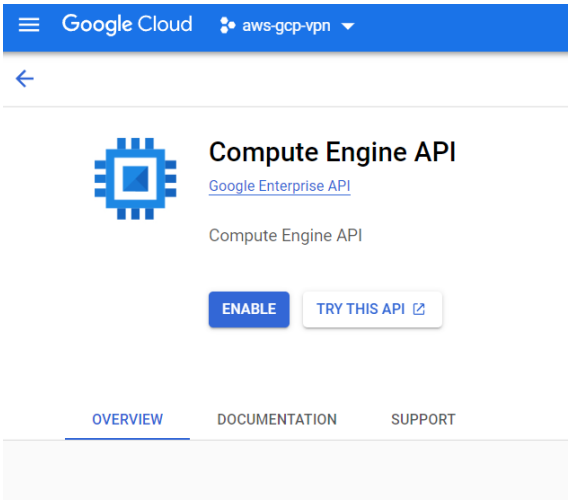

Ilustración 127 Habilitación de la API Compute Engine

Una vez activada observaremos que ya contamos con una VPC predeterminada. Sin embargo, no utilizaremos dicha red, sino que crearemos una nueva con una única subred. Trabajaremos con una nueva buscando establecer un entorno pequeño y sencillo el cual nos permita entender mejor que es lo que estamos haciendo realmente.

Dicho esto, procedemos a crear la red haciendo click en “Create VPC network”. Le damos un nombre y personalizamos la subred que se desplegará a su vez. En nuestro caso se ha seleccionado:

- Region: us-east1

- IPv4 range: 10.0.0.0/24
- Dynamic routing mode: Global (permite aprender dinámicamente las rutas hacia y desde todas las regiones)

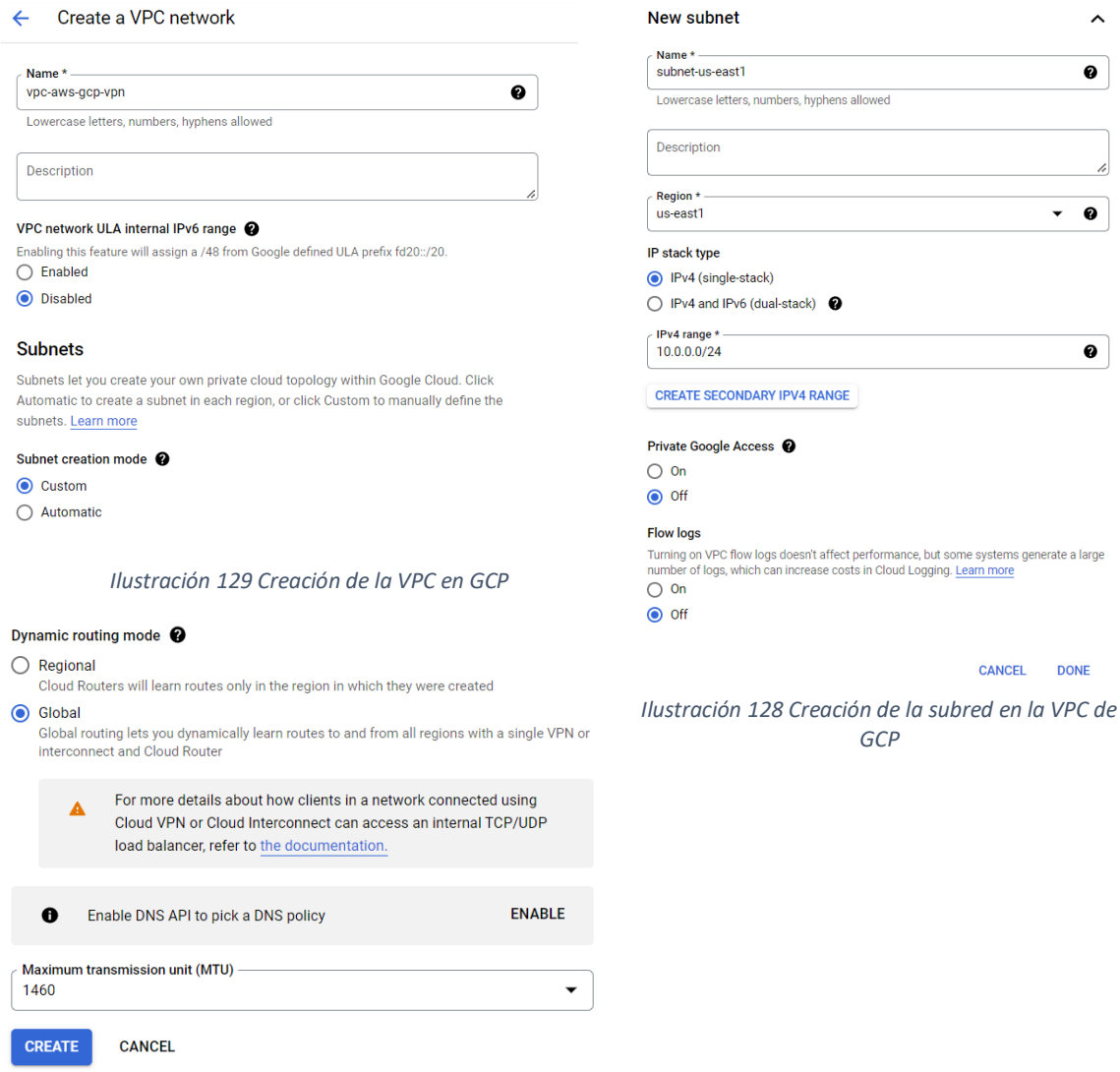

Ilustración 129 Creación de la VPC en GCP

Ilustración 128 Creación de la subred en la VPC de GCP

Ilustración 130 Habilitación del routing dinámico

Una vez terminado, podemos observar que se ha añadido nuestra nueva VPC a parte de la predeterminada:

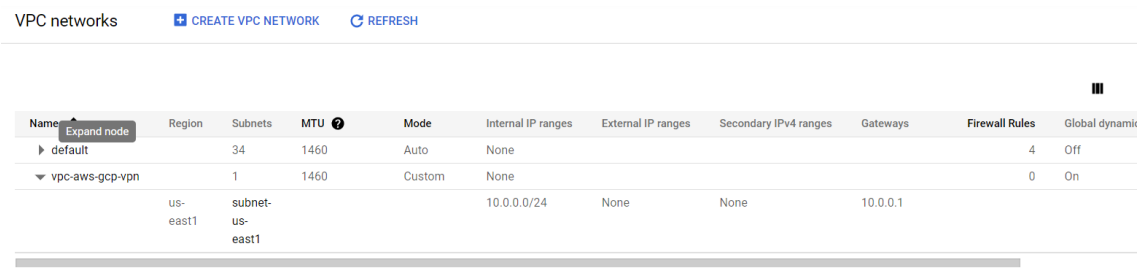

Ilustración 131 Listado de VPCs en GCP

A continuación, crearemos reglas de firewall que nos permitan hacer ping a las instancias albergadas en la VPC recién creada, así como conectarnos a ellas. Para ello, cada vez que

queramos crear una nueva regla, nos dirigiremos a “VPC Network -> Firewall -> Create Firewall Rule”:

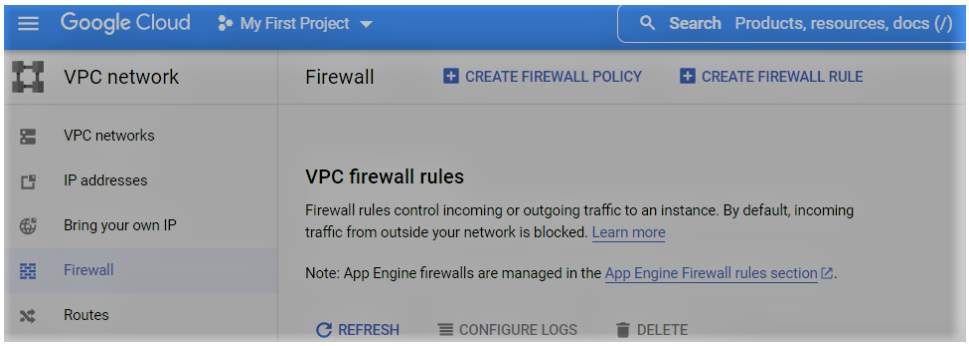

Ilustración 132 Creación de una nueva regla de firewall

Mediante la regla denominada “allow-icmp” permitiremos a cualquier usuario hacer ping a nuestras instancias. Los pasos a seguir y las opciones que elegiremos para dicha regla son:

- Asociamos esta regla a todas las instancias que vayamos a crear en nuestra red VPC:
  - Network: vpc-aws-gcp-vpn
  - Targets: All instances in the network
- Aceptamos los mensajes ping desde cualquier lugar:
  - Direction of traffic: Ingress
  - Action on match: Allow
  - Source filter: IPv4 ranges
  - Source IPv4 ranges: 0.0.0.0/0
- Por último, indicamos el protocolo sujeto a la regla, en este caso ICMP:
  - Specified protocols and ports -> Other -> Protocols: icmp

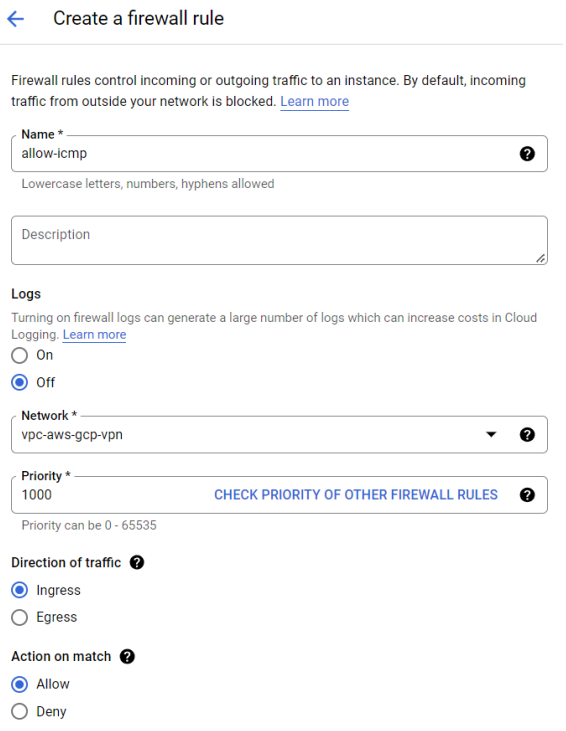

Ilustración 134 Opciones de la regla allow-icmp (1)

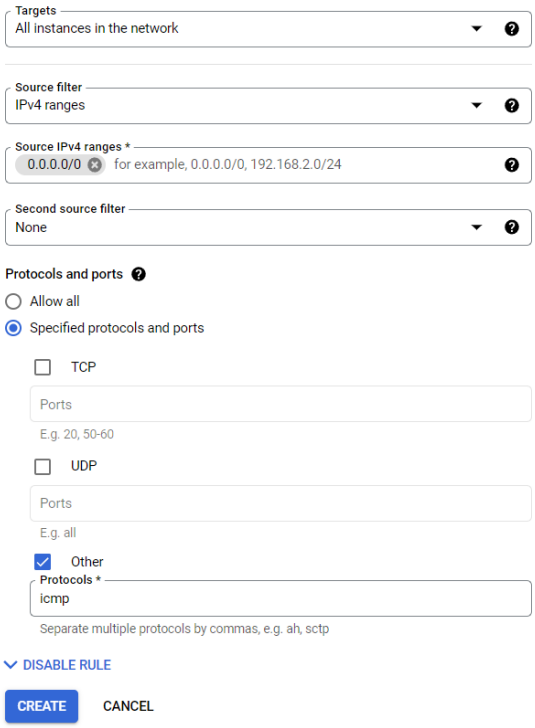

Ilustración 133 Opciones de la regla allow-icmp (2)

Por otra parte, la regla “allow-ssh-from-console”, nos servirá para realizar conexiones SSH a las instancias desde la consola de GCP:

- Asociamos esta regla a todas las instancias creadas en la red vpc-aws-gcp-vpn, tal como lo hicimos en el caso anterior.
- No queremos que se permita realizar conexiones SSH desde cualquier lugar, por lo tanto, sólo daremos acceso desde las direcciones del servicio IAP (Identity-Aware Proxy), que nos permite acceder desde la consola de GCP vía SSH:
    - Source IPv4 ranges: 35.235.240.0/20
- Indicamos el puerto TCP correspondiente al protocolo SSH:
    - Specified protocols and ports -> TCP -> Ports: 22

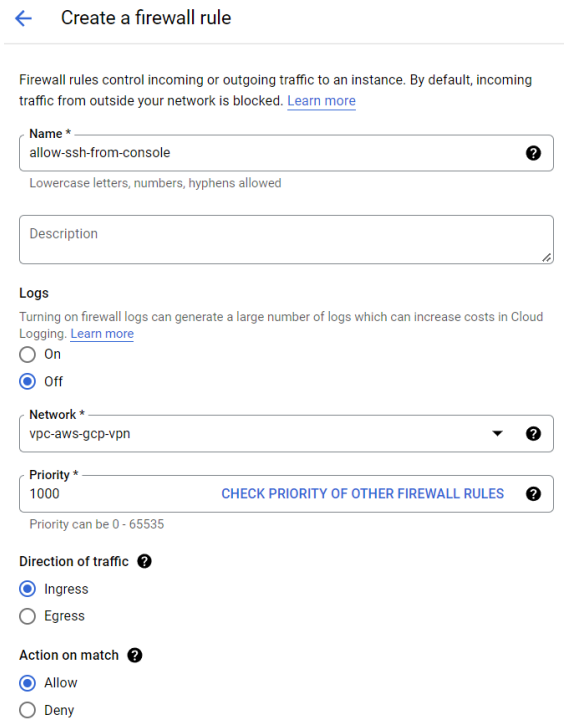

Ilustración 136 Opciones de la regla allow-ssh-from-console (1)

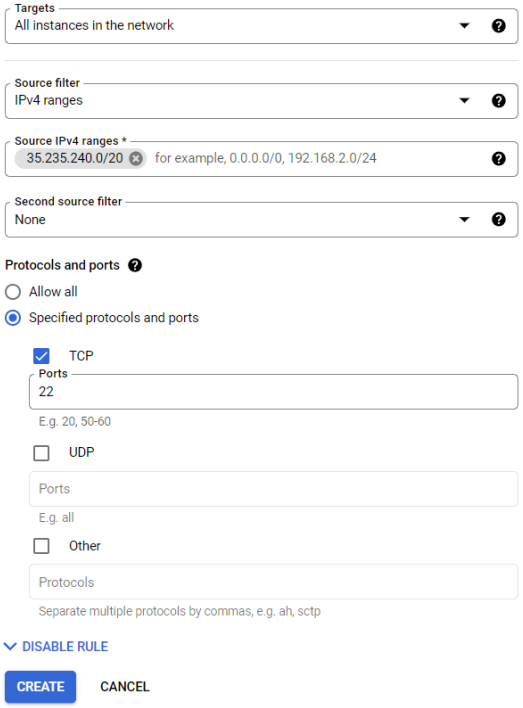

Ilustración 135 pciones de la regla allow-ssh-from-console (2)

7.2. Creación y configuración de una nube privada virtual en AWS

Nos movemos a AWS para establecer su VPC correspondiente. Entre los servicios seleccionamos, o buscamos, “VPC “, dentro en “Your VPCs” observamos, al igual que en Google Cloud, que ya existe una por defecto. De nuevo, para nuestro caso crearemos otra desde cero (seleccionamos “Create VPC”) con el mismo nombre que la de Google Cloud, pero con un rango de direccionamiento diferente, ya que ambas VPC estarán conectadas y en el mismo espacio de direccionamiento privado:

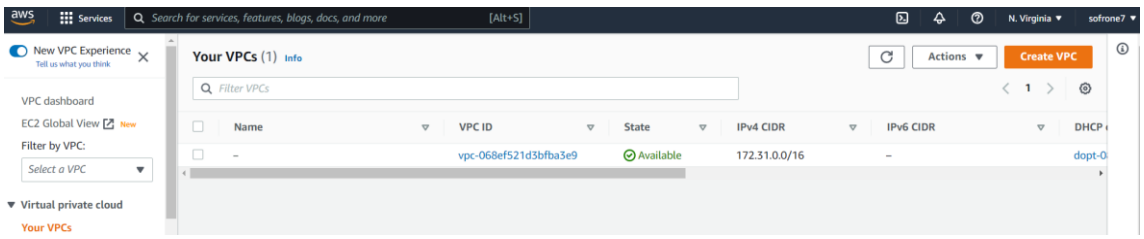

Ilustración 137 VPC predeterminada en AWS

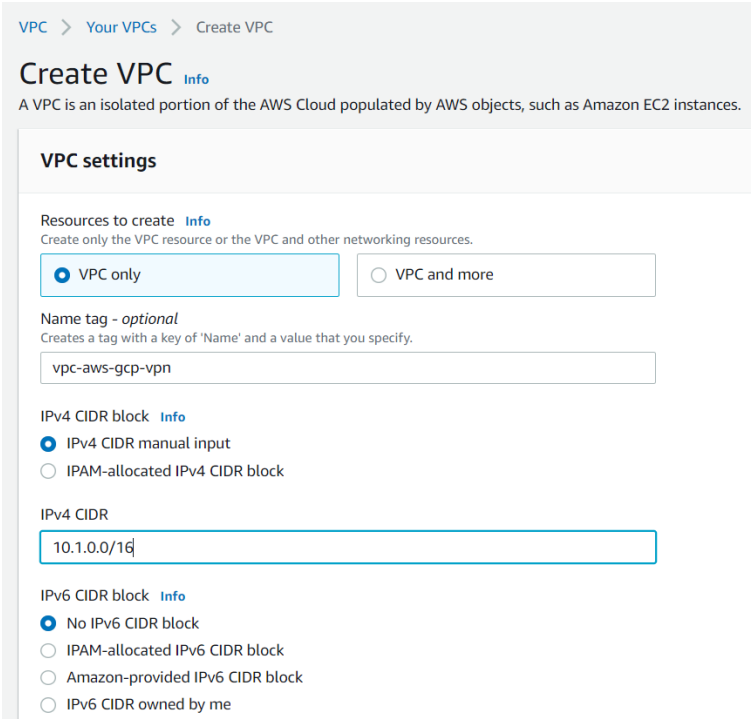

Ilustración 138 Creación de una VPC en AWS

A continuación, clickamos en “VPC -> Subnets -> Create subnet” para crear una subred asociada a la VPC recién creada. Para ello, primero debemos seleccionar el identificador de dicha VPC:

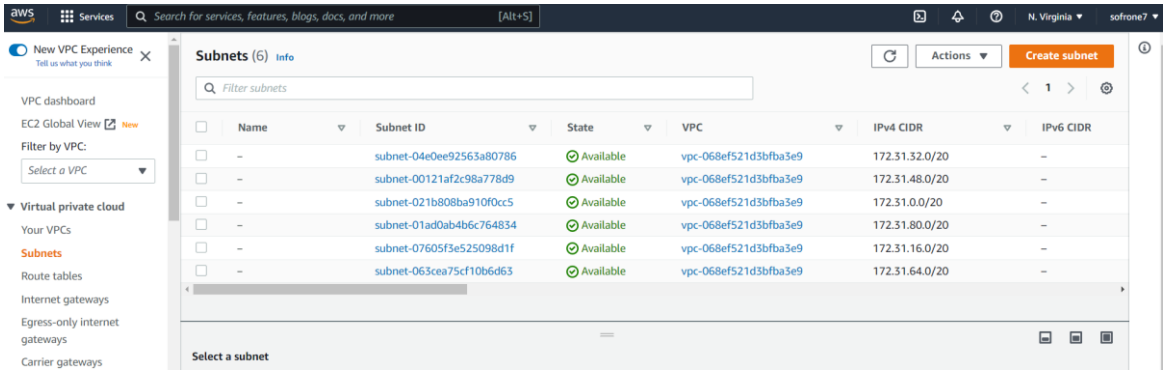

Ilustración 139 Creación de una subred en la VPC (1)

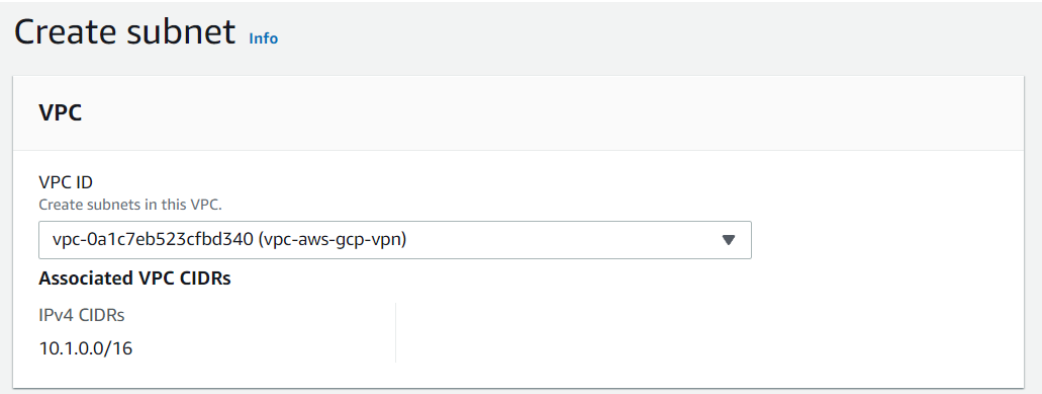

Ilustración 140 Creación de una subred en la VPC (2)

Una vez hayamos indicado en qué VPC queremos crear la nueva subred, se nos desplegarán las opciones de su configuración donde seleccionaremos la zona en la que residirá la subred y su rango de direcciones correspondiente:

- Subnet name: subnet-us-east1
- Availability Zone: US East (N. Virginia) / us-east-1a
- IPV4 CIDR block: 10.1.0.0/24

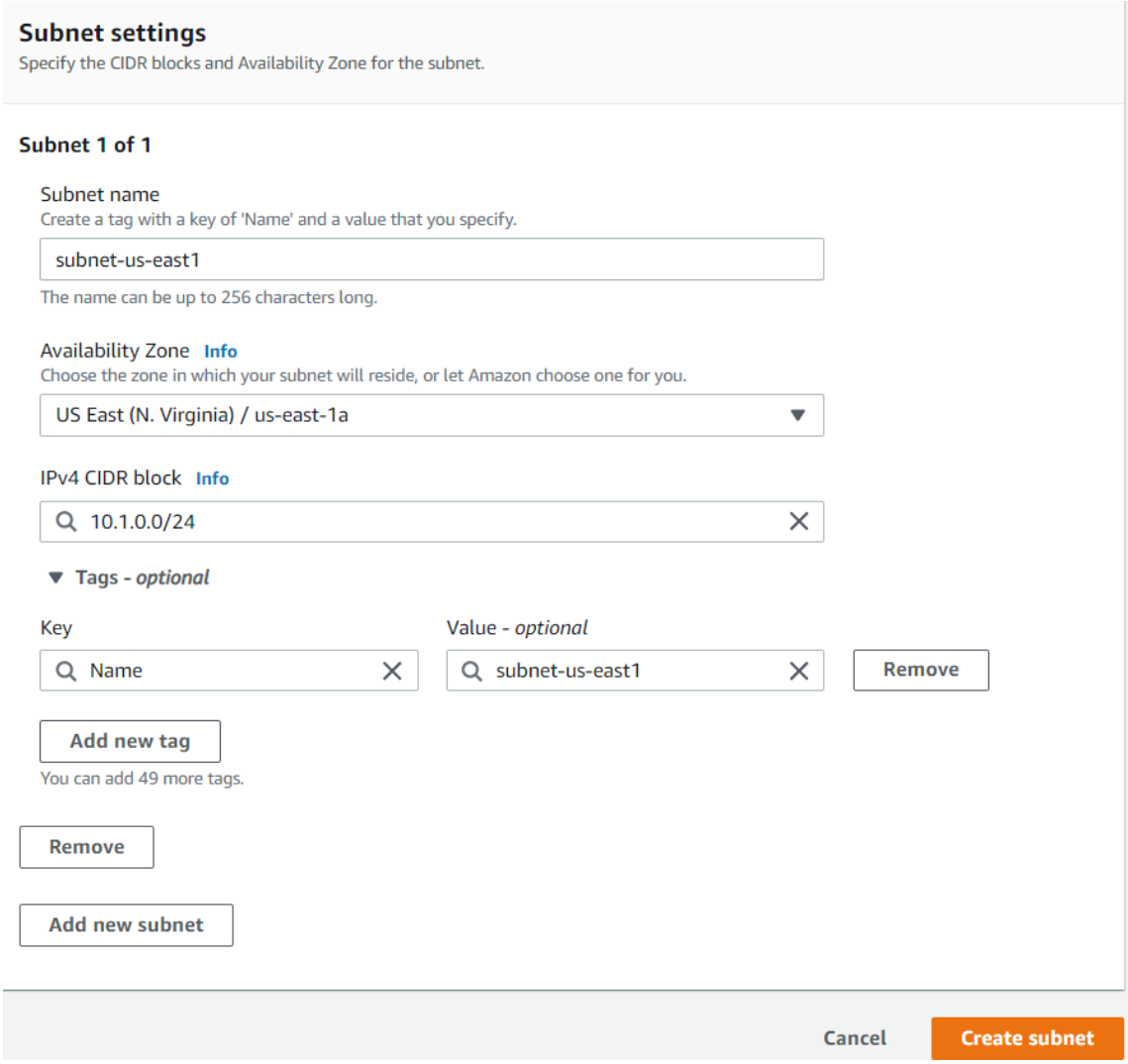

Ilustración 141 Creación de una subred en la VPC (3)

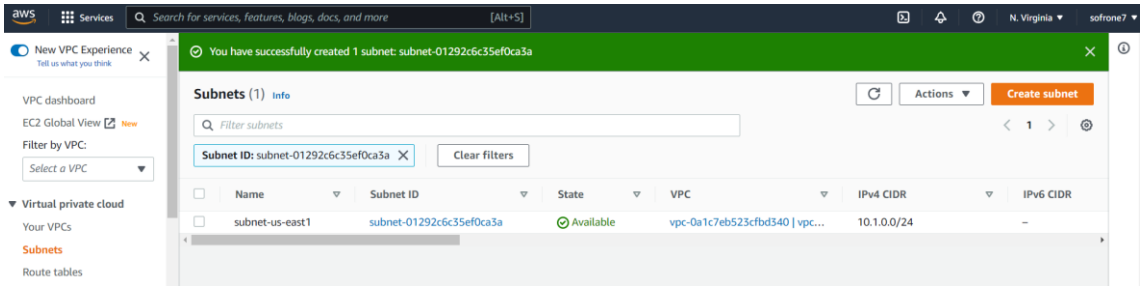

Ilustración 142 Listado de subredes en AWS

En el momento de crear la nube privada virtual, a su vez, se ha creado automáticamente una tabla de rutas (“Virtual private cloud -> Route tables”) asociada al router que viene implícito con

la red VPC. Procedemos a darle un nombre a la tabla de rutas para distinguirla de la tabla que tenemos de forma predeterminada:

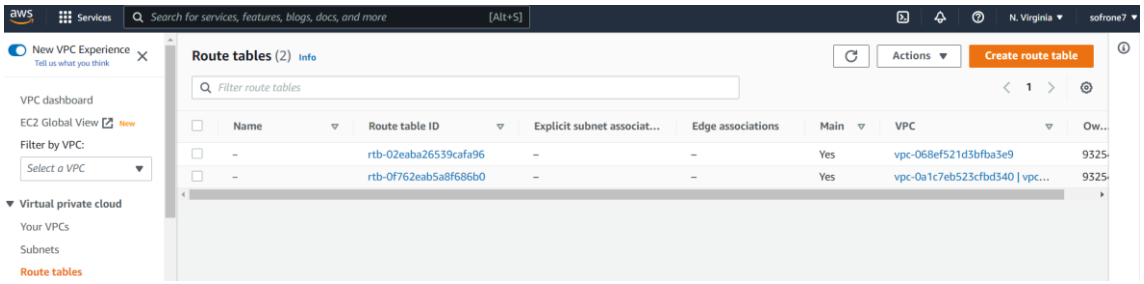

Ilustración 143 Configuración de la tabla de rutas en AWS (1)

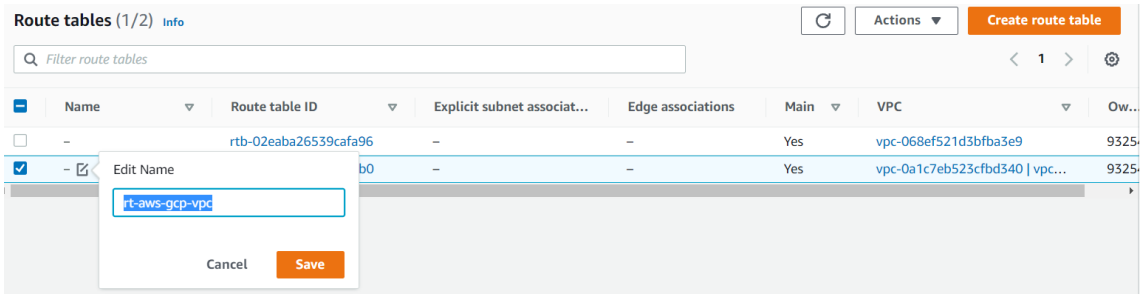

Ilustración 144 Configuración de la tabla de rutas en AWS (2)

Si accedemos a las rutas de la tabla podemos observar que existe una entrada que enrutar todo lo que está dentro del rango de direccionamiento de la red VPC (10.1.0.0/16) a la ruta “local”, una ruta predeterminada para la comunicación dentro de la VPC:

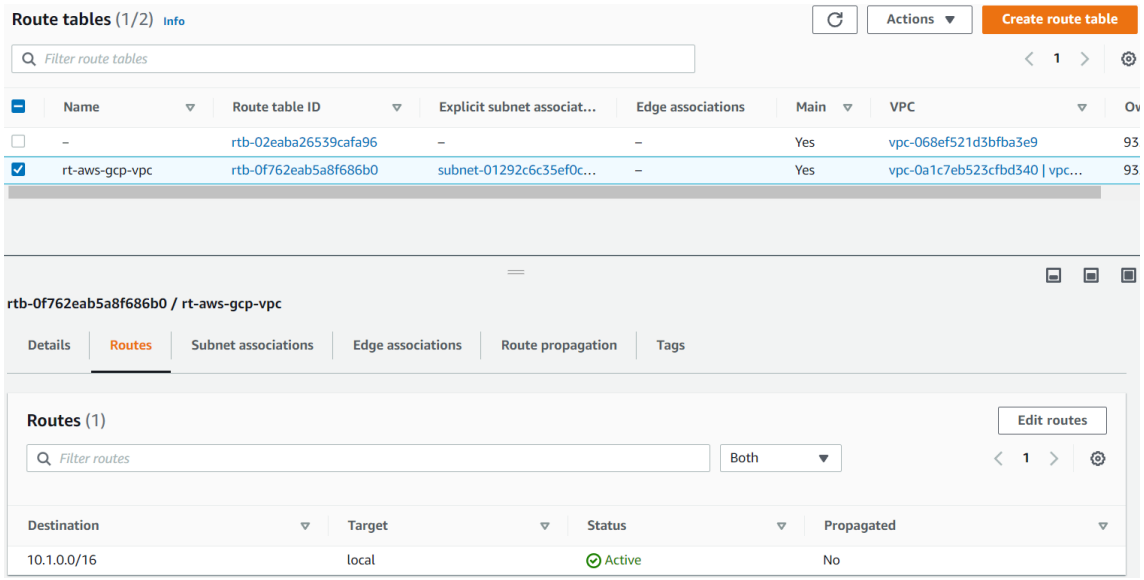

Ilustración 145 Configuración de la tabla de rutas en AWS (3)

Sin embargo, aunque la subred que acabamos de configurar entra dentro del rango de direccionamiento de la regla, eso no implica que el tráfico que se origina en ella se enrute de acuerdo con dicha ruta. AWS especifica que se debe asociar la subred a la tabla de rutas si queremos que esto sea posible. Para ello solamente tenemos que pinchar sobre la tabla, seleccionar “Subnet associations” y la subred que queremos asociar, en este caso “subnet-us-east1”:

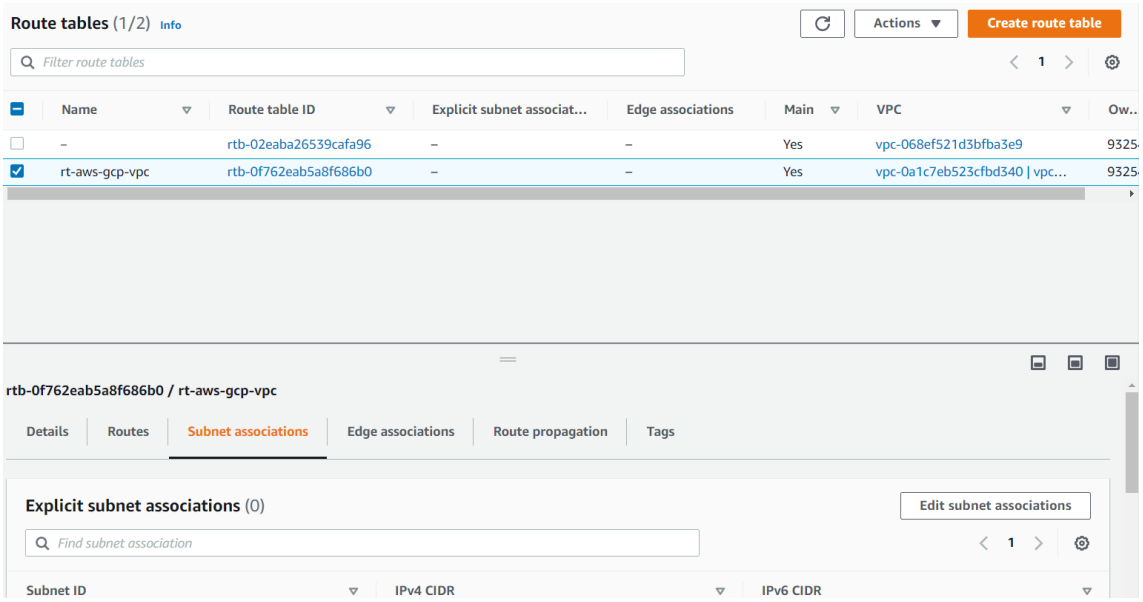

Ilustración 146 Configuración de la tabla de rutas en AWS (4)

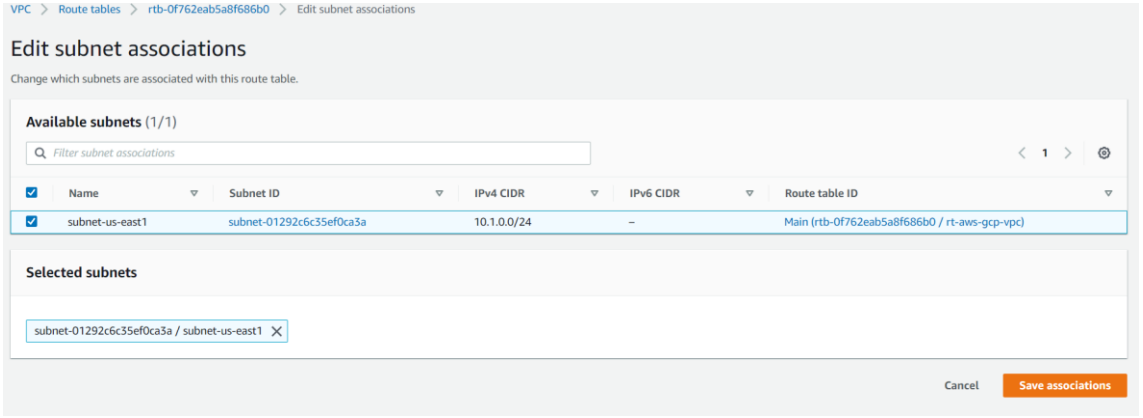

Ilustración 147 Configuración de la tabla de rutas en AWS (5)

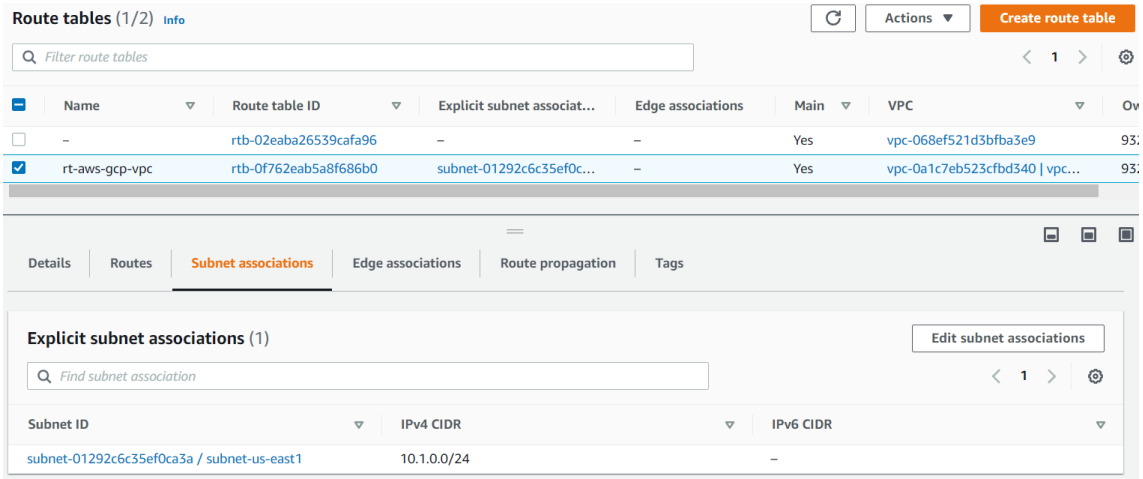

Ilustración 148 Configuración de la tabla de rutas en AWS (6)

Ahora que la tabla de rutas de la VPC ya tiene en cuenta para su enrutamiento la subred “subnet-us-east1”, nos falta por darle salida a Internet. Haremos uso de un “Internet Gateway”, un router virtual que se encarga de conectar una red VPC a Internet. Es tan simple como acceder a “VPC -> Internet gateways -> Create internet Gateway” y darle un nombre:

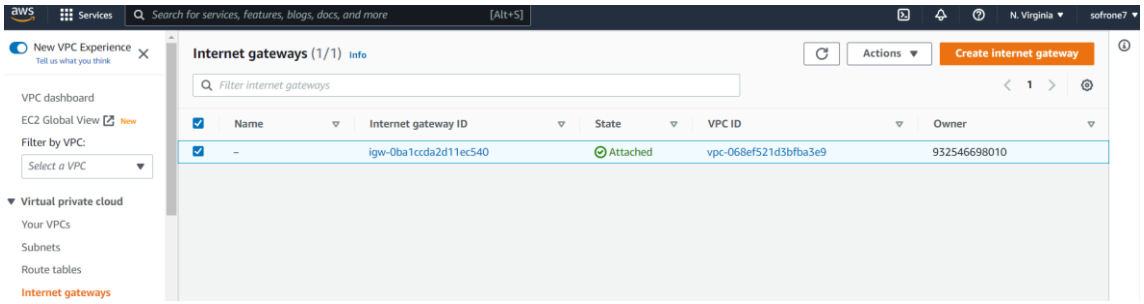

Ilustración 149 Configuración del internet gateway en AWS (1)

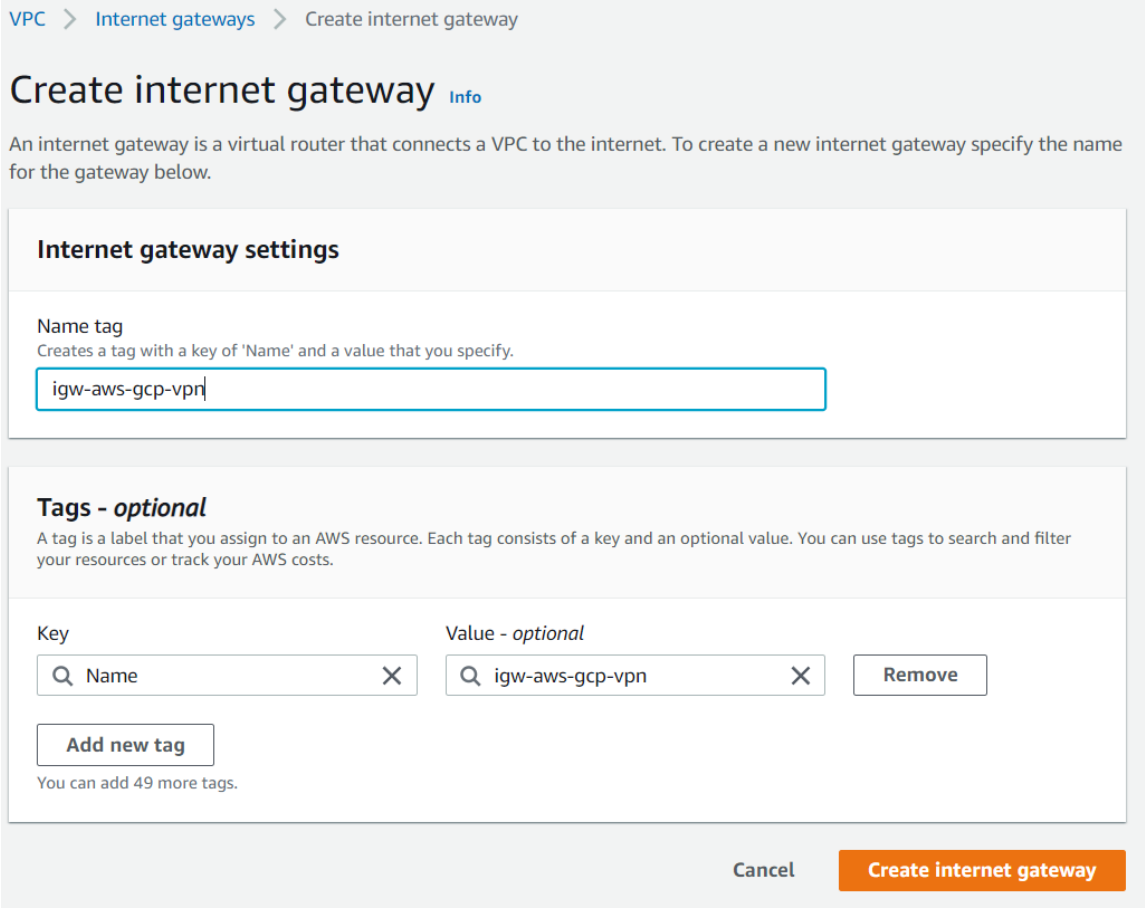

Ilustración 150 Configuración del internet gateway en AWS (2)

Una vez creado, nos aparecerá un cuadro de diálogo dándonos la opción de conectar dicha puerta de enlace de Internet a una VPC. Seleccionamos la opción “Attach to a VPC” del cuadro de diálogo y, de entre las VPCs disponibles, seleccionamos el identificador de la red “vpc-aws-gcp-vpn”:

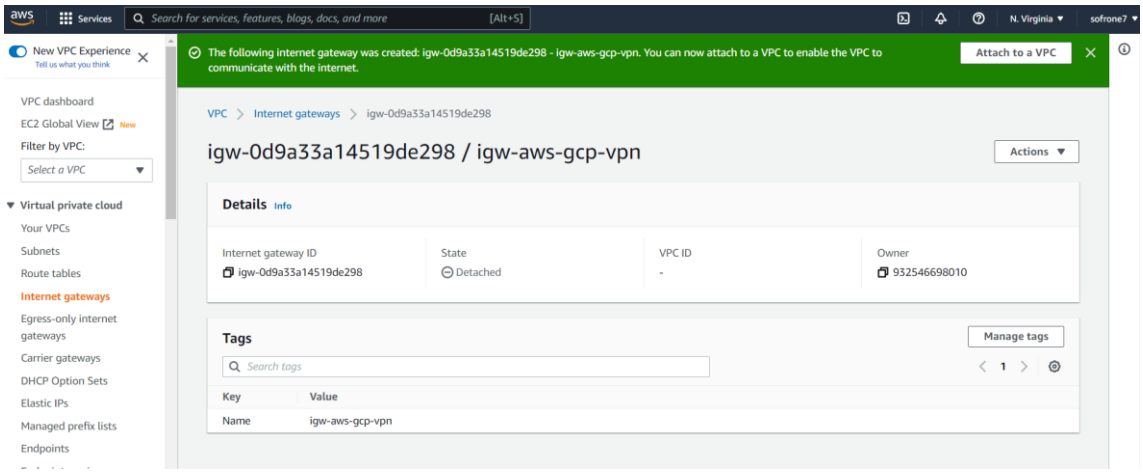

Ilustración 151 Configuración del internet gateway en AWS (3)

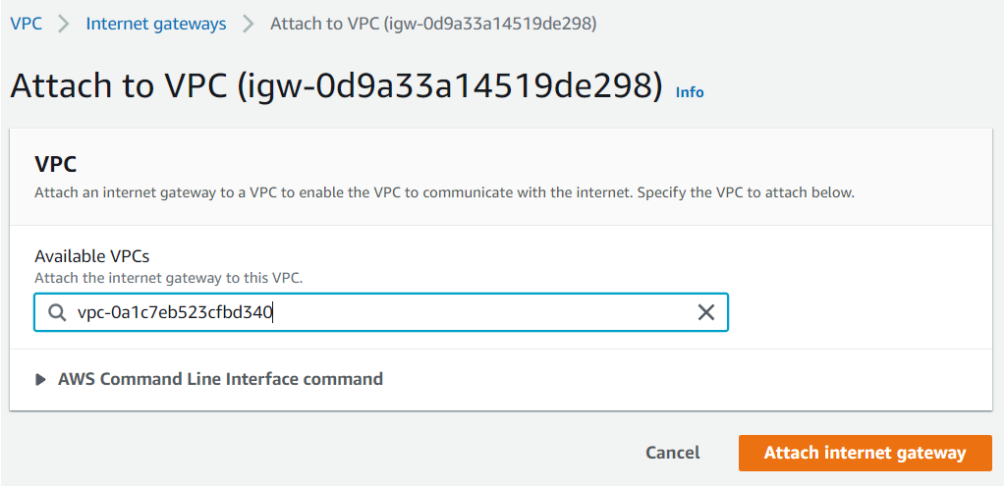

Ilustración 152 Configuración del internet gateway en AWS (4)

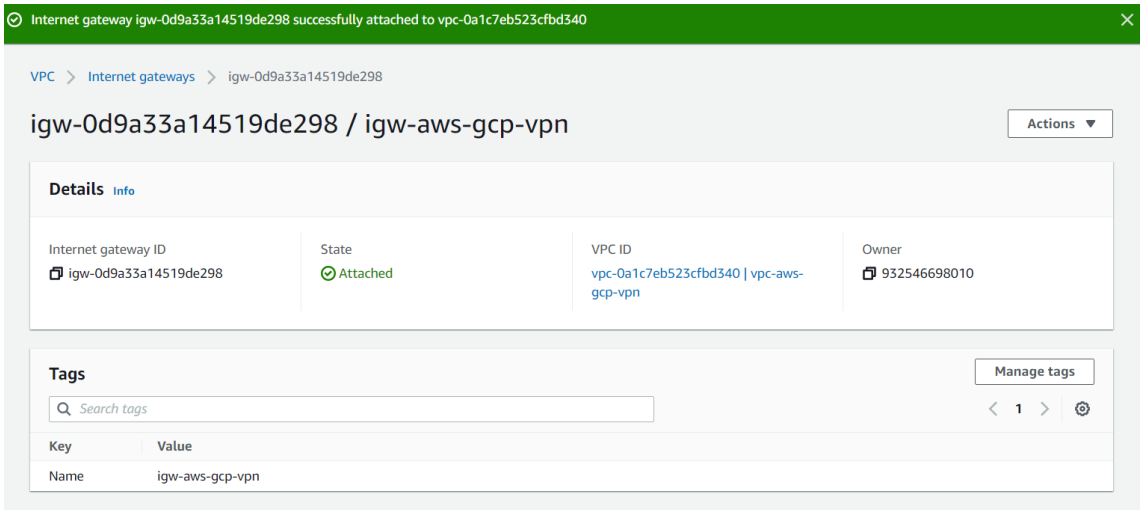

Ilustración 153 Configuración del internet gateway en AWS (5)

Una vez creada la puerta de enlace a Internet, y asociada a la VPC, para permitir que la red pueda acceder a Internet a través de ella seleccionamos “Edit routes” en el apartado “Routes” de la tabla de rutas y añadimos la siguiente ruta:

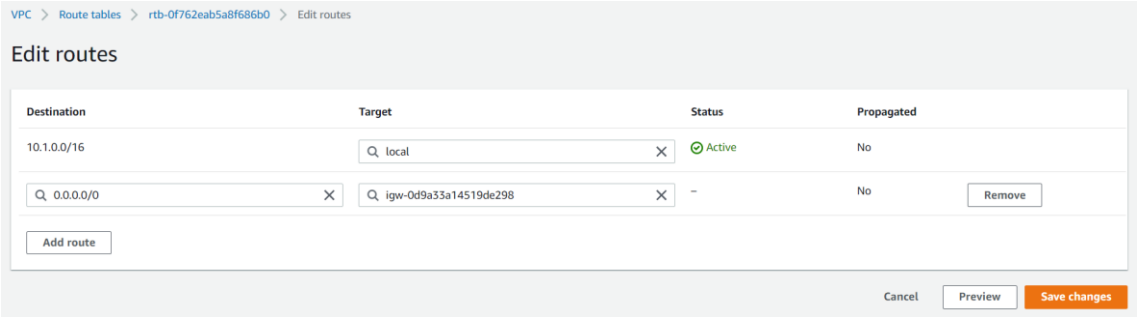

Ilustración 154 Configuración del internet gateway en AWS (6)

El rango de direcciones al que queremos que se dirija el tráfico es 0.0.0.0/0, que representa todas las direcciones IPv4. Y la puerta de enlace de Internet que está adjunta a la VPC es la conexión a través de la cual enviar el tráfico de destino.

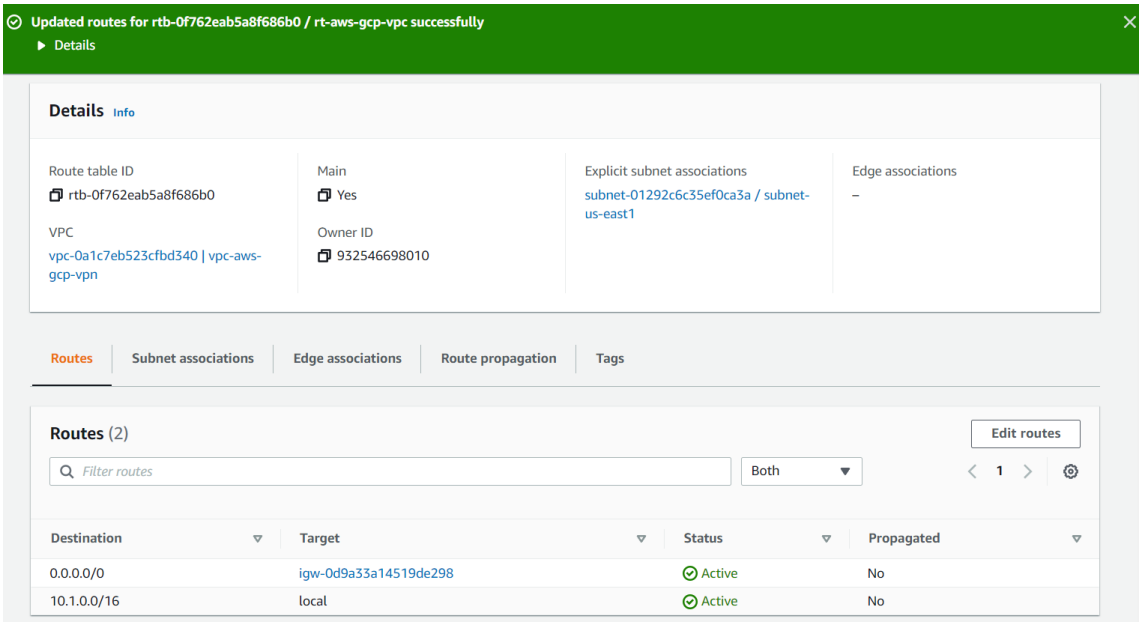

Ilustración 155 Configuración del internet gateway en AWS (7)

Para terminar con la configuración en AWS de este apartado, en caso de querer acceder a las instancias a partir de un nombre de DNS público debemos activar la opción “DNS hostnames”. Se selecciona la VPC en cuestión, en el desplegable “Actions” clickamos en “Edit DNS hostnames” y lo activamos:

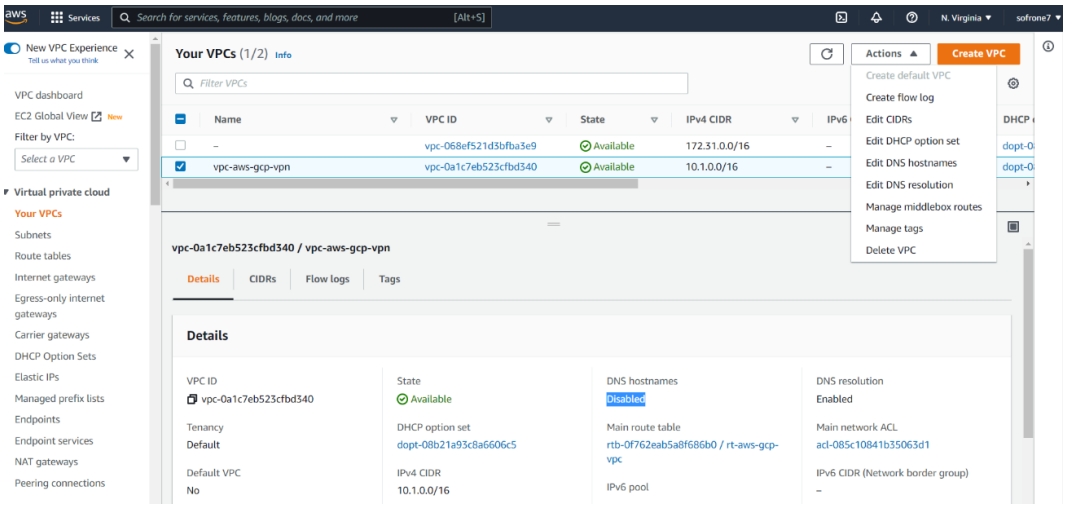

Ilustración 156 Configuración de acceso a las instancias por un nombre de DNS público (1)

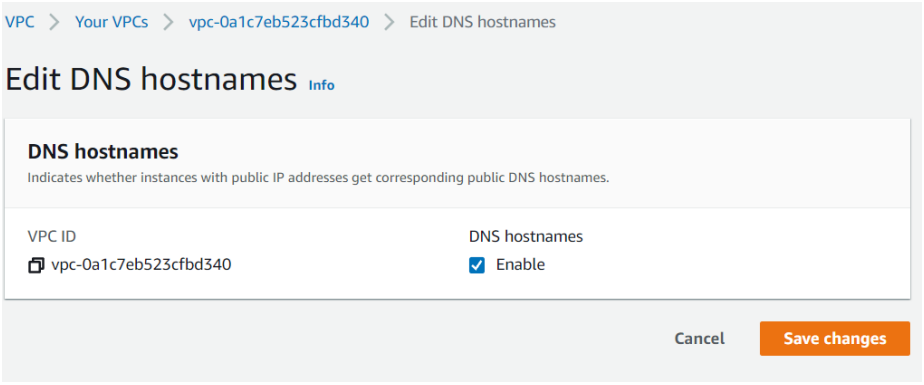

Ilustración 157 Configuración de acceso a las instancias por un nombre de DNS público (2)

7.3. Crear la puerta de enlace de VPN y Cloud Router en Google Cloud

Una vez configuradas las VPC en ambas nubes, llegó el momento de conectarlas para que ambas redes puedan comunicarse entre sí. Recordemos que del lado de Google Cloud necesitaremos configurar una VPN de alta disponibilidad (HA).

En primer lugar, deberemos crear un “Cloud Router” que nos permitirá la conexión entre la VPN de Google Cloud y la red de AWS. Accedemos a “Hybrid Connectivity -> Cloud Routers”, elegimos la opción “CREATE ROUTER” y lo configuramos con los siguientes datos:

- Name: cloud-router-aws-gcp-vpn
- Network: vpc-aws-gcp-vpn (la red VPC configurada en apartados anteriores)
- Region: us-east1
- Google ASN: 65420 (número de sistema autónomo privado en el rango 64512-65534 o 4200000000-4294967294, un identificador único que permite que nuestro sistema autónomo intercambie información de enrutamiento con otros sistemas mediante BGP)

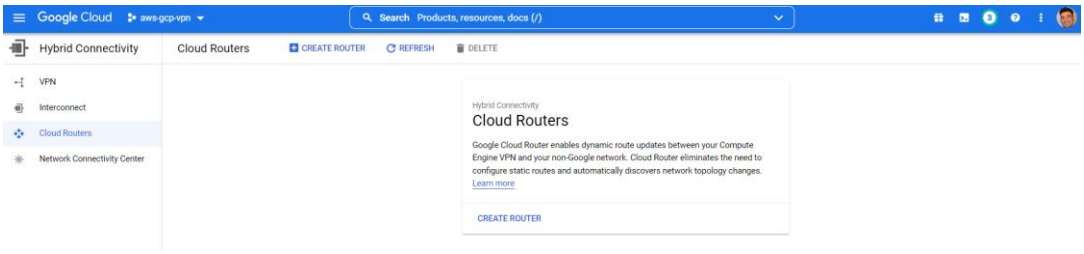

Ilustración 158 Configuración de un Cloud Router en GCP (1)

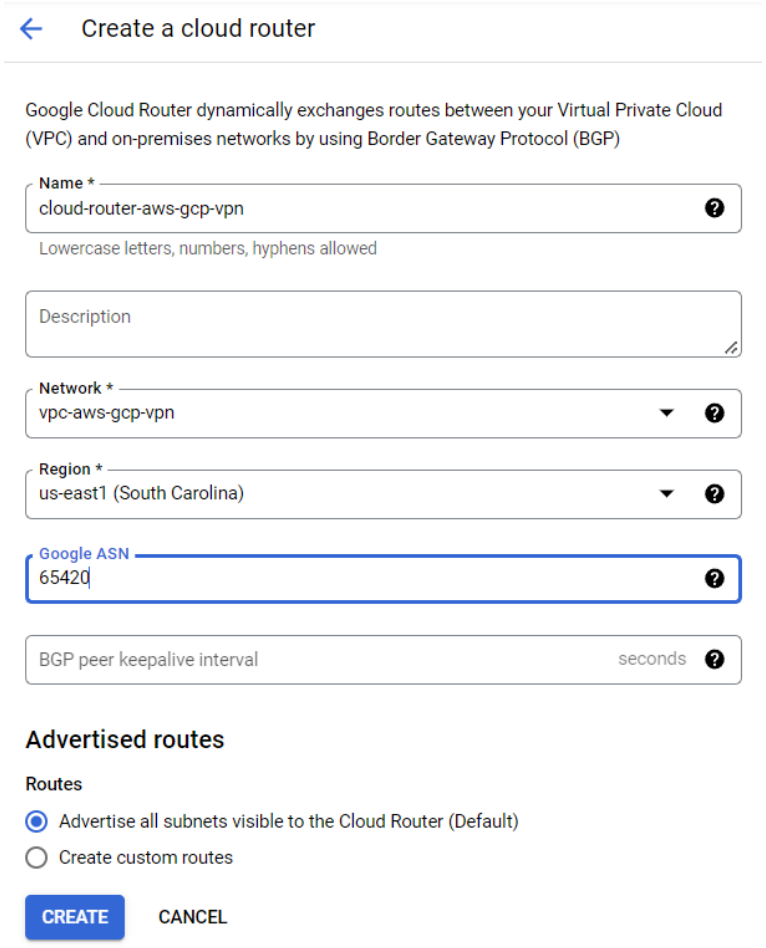

Ilustración 159 Configuración de un Cloud Router en GCP (2)

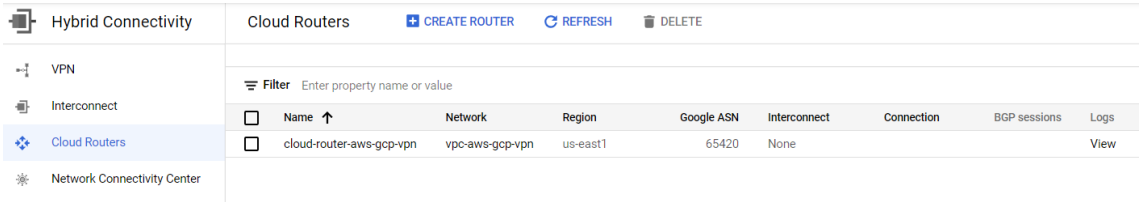

Ilustración 160 Configuración de un Cloud Router en GCP (3)

Ahora sí, procedemos a crear la red privada virtual correspondiente a Google Cloud. En el mismo apartado “Hybrid Connectivity” seleccionamos “VPN -> CREATE VPN CONNECTION”.

En primer lugar, tendremos que elegir por el tipo de VPN que queremos. Cómo ya he comentado con anterioridad, nos decantamos por una HA VPN, que nos proporciona una conexión segura y de alta disponibilidad:

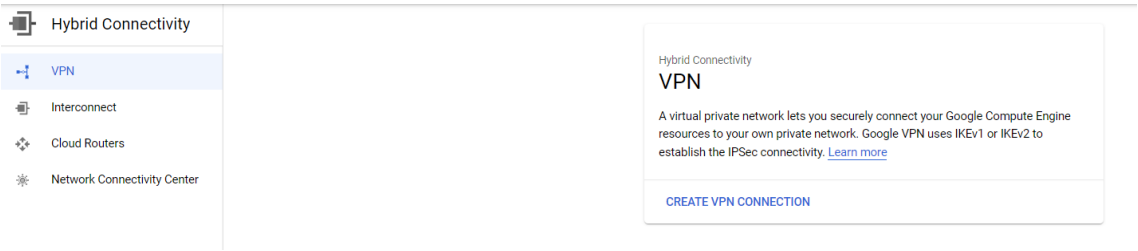

Ilustración 161 Configuración de una VPN en GCP (1)

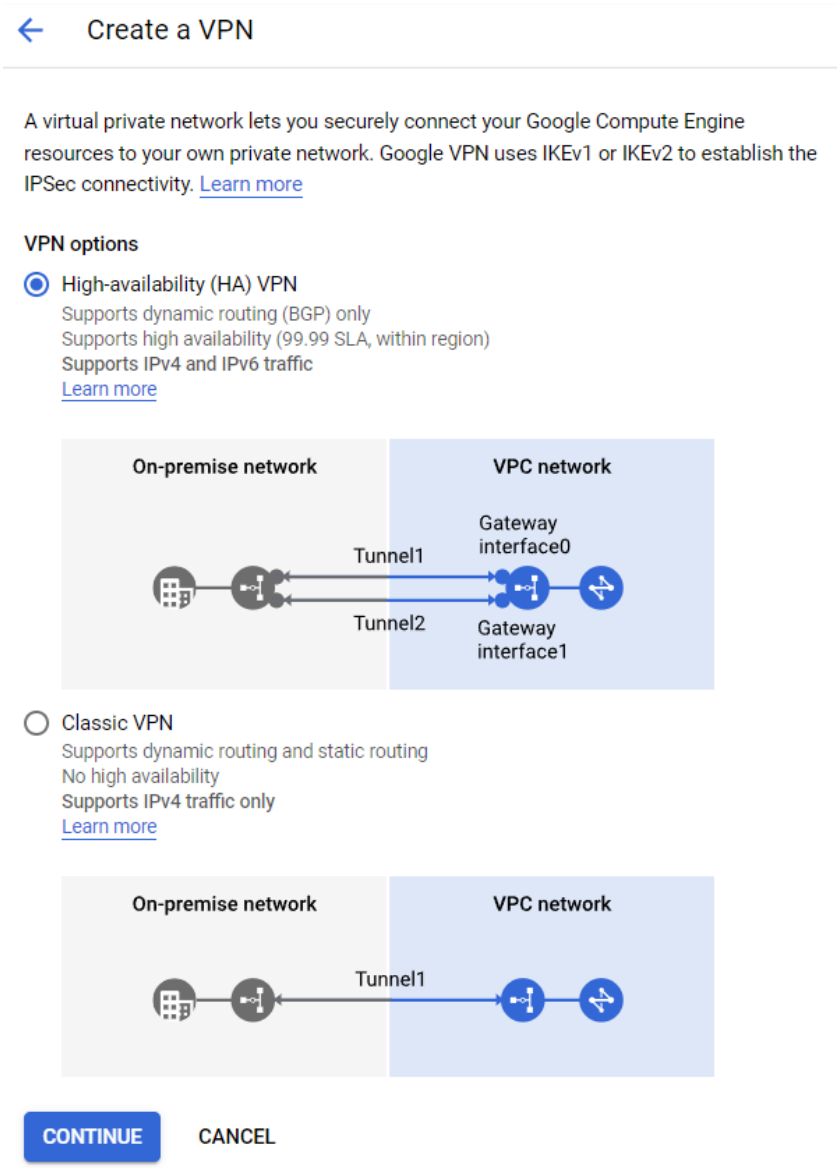

Ilustración 162 Configuración de una VPN en GCP (2)

El siguiente paso será configurar la puerta de enlace de alta disponibilidad. Le otorgamos un nombre, elegimos la red VPC en la que trabajará y la región:

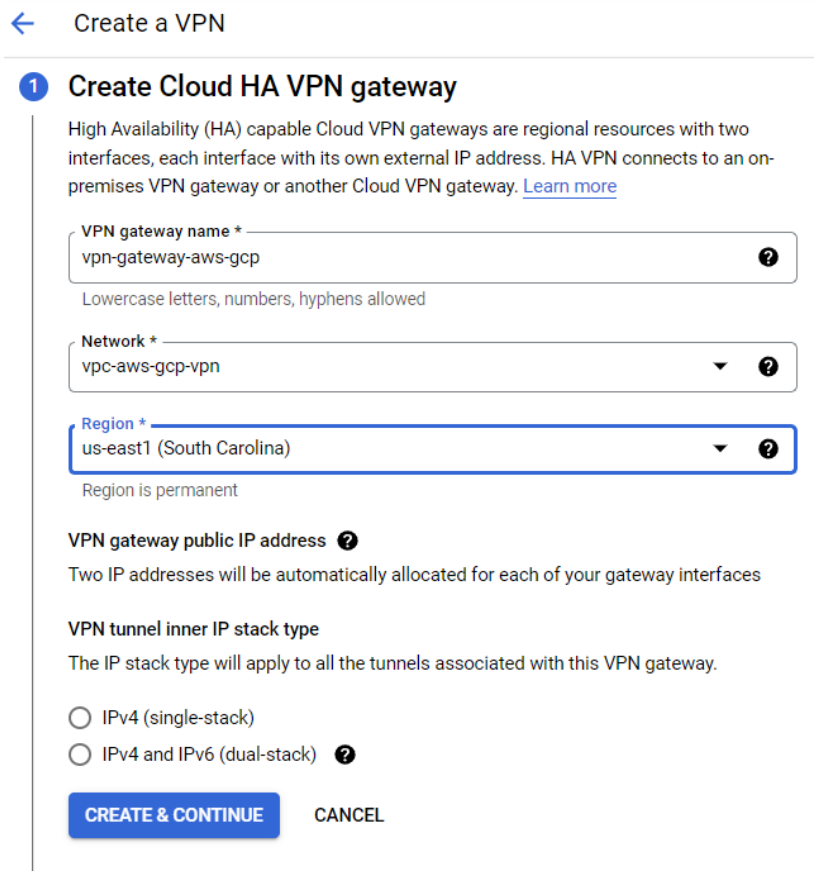

Ilustración 163 Configuración de una VPN en GCP (3)

Una vez aceptemos la configuración se creará la puerta de enlace con dos interfaces de VPN externas, cada una con una IP pública asignada. Se usarán los valores 0 y 1 para identificar cada interfaz junto a su respectiva IP pública. Deberemos apuntar dichas direcciones, ya que serán necesarias para la configuración de las puertas de enlace AWS de intercambio de tráfico.

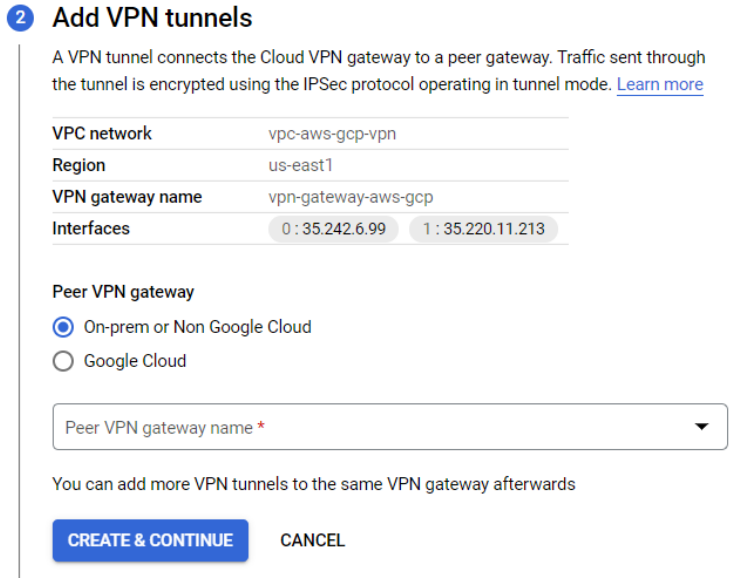

Ilustración 164 Configuración de una VPN en GCP (4)

7.4. Crear conexiones de VPN en AWS

Dejamos parada la configuración de la VPN en Google Cloud, ya que para comunicar ambas nubes utilizaremos conexiones de VPN Site-to-Site de Amazon [38], por tanto, antes de continuar debemos crear dichas conexiones. Se trata de conexiones seguras con cifrado IPsec entre las VPCs albergadas en AWS y la red que deseemos, es decir, a nuestra red VPC en GCP.

Una vez creadas las conexiones en AWS, utilizaremos la información de su configuración para continuar con el proceso del lado de Google Cloud y conectar ambas VPC. Por tanto, en esta sección seguiremos los pasos necesarios para levantar dichas conexiones VPN.

Serán necesarios básicamente dos componentes entre los cuáles realizaremos las conexiones VPN Site-to-Site:

- **Customer gateway**: recurso de AWS que representa el dispositivo o servicio de puerta de enlace en el lado de la conexión VPN de Google Cloud, en este caso el Cloud HA VPN gateway, con el que vamos a conectarnos. Recordemos que el gateway de GCP cuenta con dos interfaces, por tanto, deberemos establecer dos puertas de enlace de cliente para cada conexión VPN.
- **Virtual private gateway**: se trata de un concentrador VPN (un dispositivo de red que proporciona creación segura de conexiones VPN) ubicado en el lado de la conexión VPN de AWS. La puerta de enlace privada virtual se debe adjuntar a la VPC que hemos creado en Amazon para realizar las conexiones VPN.

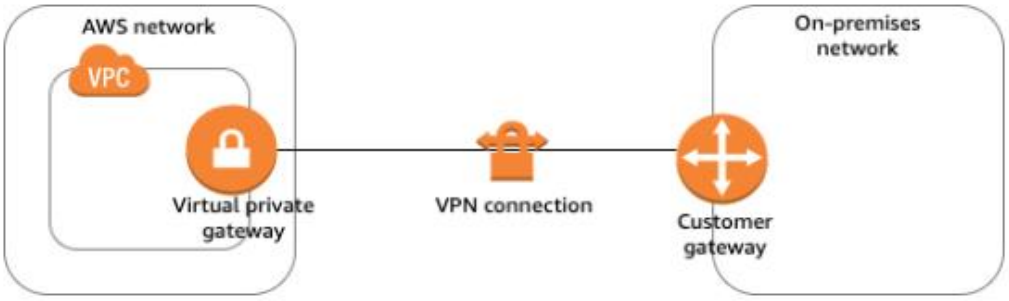

Ilustración 165 Esquema conexión VPN Site-to-Site AWS [38]

7.4.1. Establecer las puertas de enlace de cliente

En el apartado “Virtual private network” accedemos a “Customer gateways” para crear las dos puertas de enlace de cliente asociadas a las dos interfaces externas de nuestra VPN en Google Cloud. Seleccionamos “Create customer gateway” e indicamos a cada puerta de enlace una de las direcciones IP públicas de la VPN en GCP, junto al número ASN (65420) que elegimos al configurar el Cloud Router:

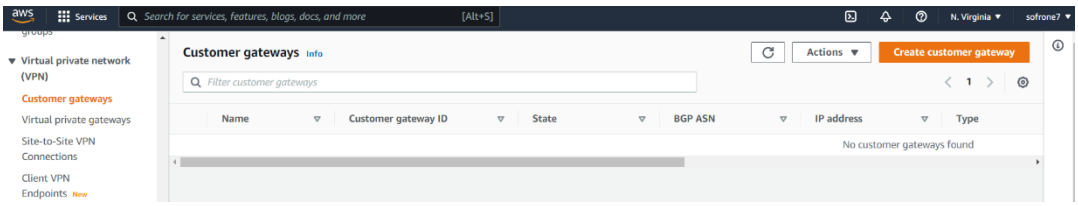

Ilustración 166 Configuración de los Customer Gateways en AWS (1)

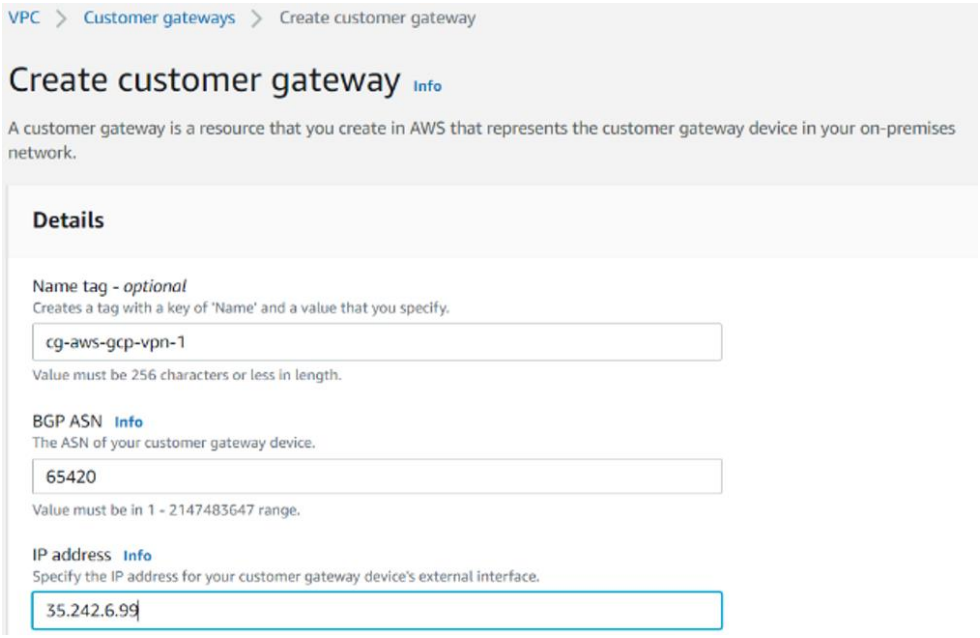

Ilustración 167 Configuración de los Customer Gateways en AWS (2)

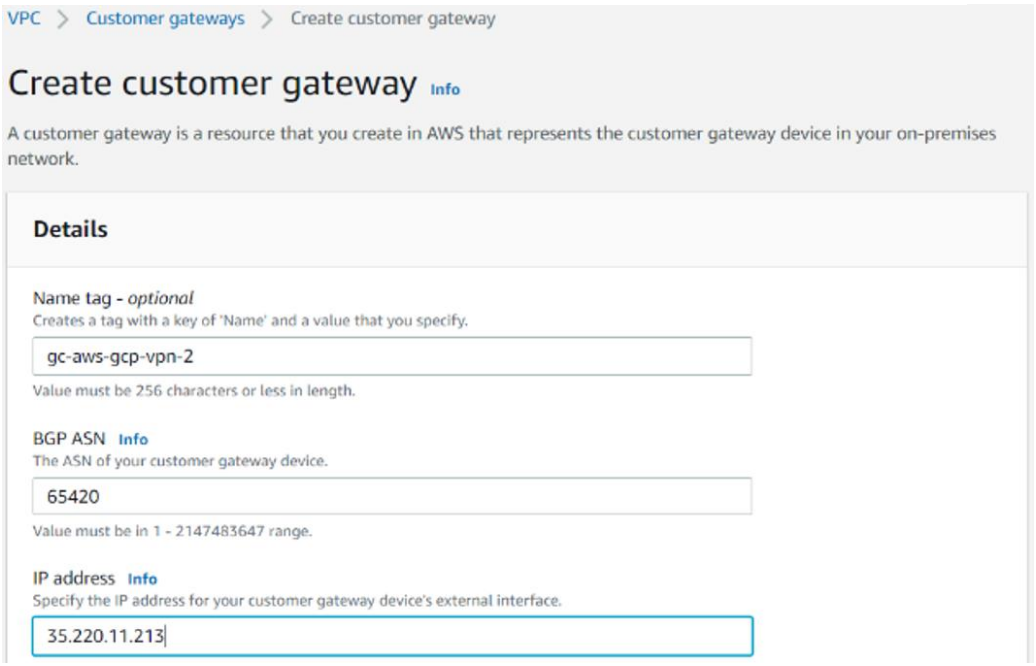

Ilustración 168 Configuración de los Customer Gateways en AWS (3)

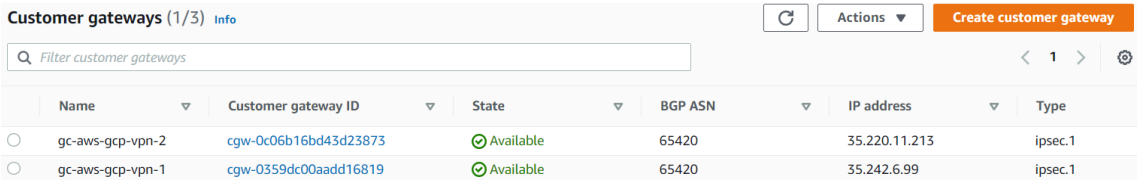

Ilustración 169 Configuración de los Customer Gateways en AWS (4)

7.4.2. Establecer la puerta de enlace privada virtual

En el mismo apartado de “Virtual private network” seleccionamos “Virtual private gateways” del menú desplegable. Para crear la puerta de enlace privada virtual será tan sencillo cómo pinchar en “Create virtual private gateway”, proporcionarle un nombre (vgp-aws-gcp) y un número ASN (por ejemplo, 64512) diferente al que estamos utilizando en el lado de GCP:

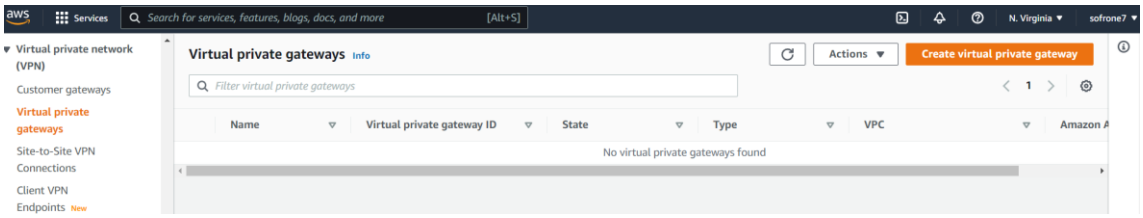

Ilustración 170 Configuración de la Virtual Private Gateway en AWS (1)

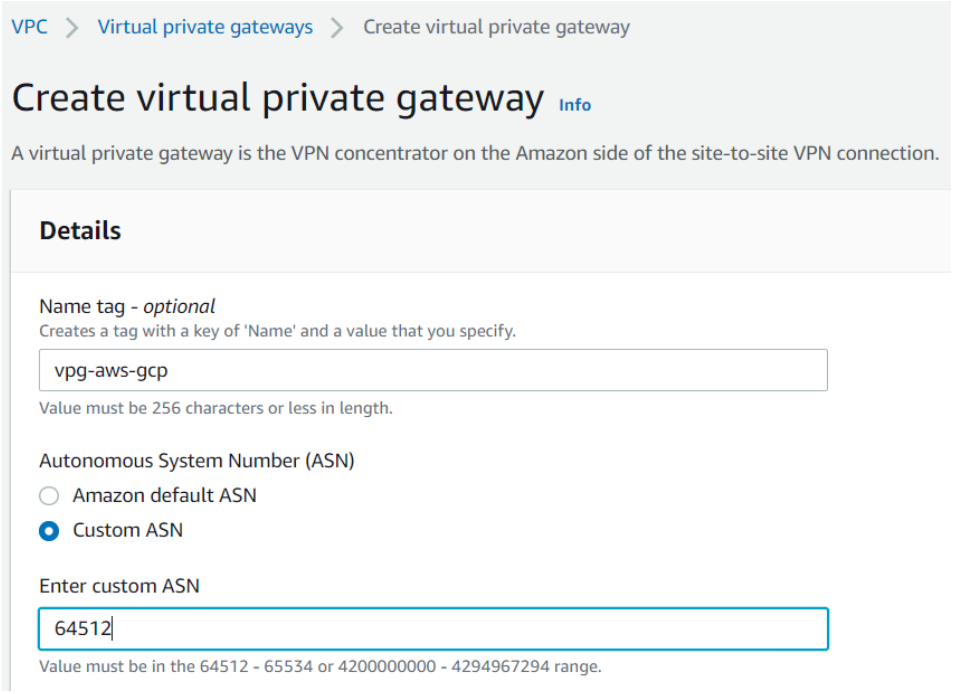

Ilustración 171 Configuración de la Virtual Private Gateway en AWS (2)

Una vez generada la puerta de enlace privada virtual, accedemos al menú desplegable “Actions” y seleccionamos “Attach to VPC”. Cómo comentábamos con anterioridad, debemos elegir la VPC que hemos creado para realizar la conexión VPN Site-to-Site, es decir, “vpc-aws-gcp-vpn”:

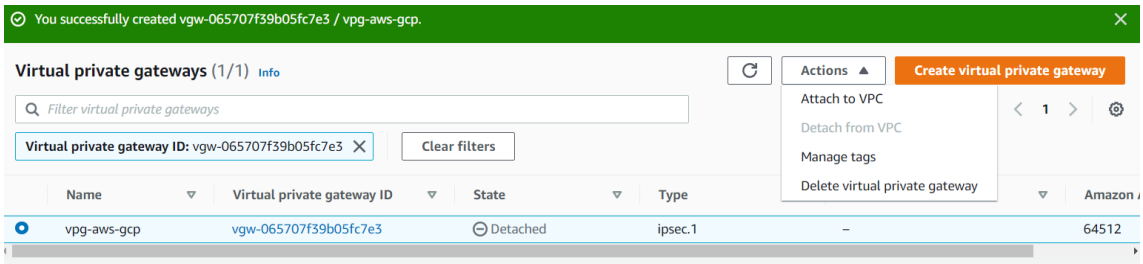

Ilustración 172 Configuración de la Virtual Private Gateway en AWS (3)

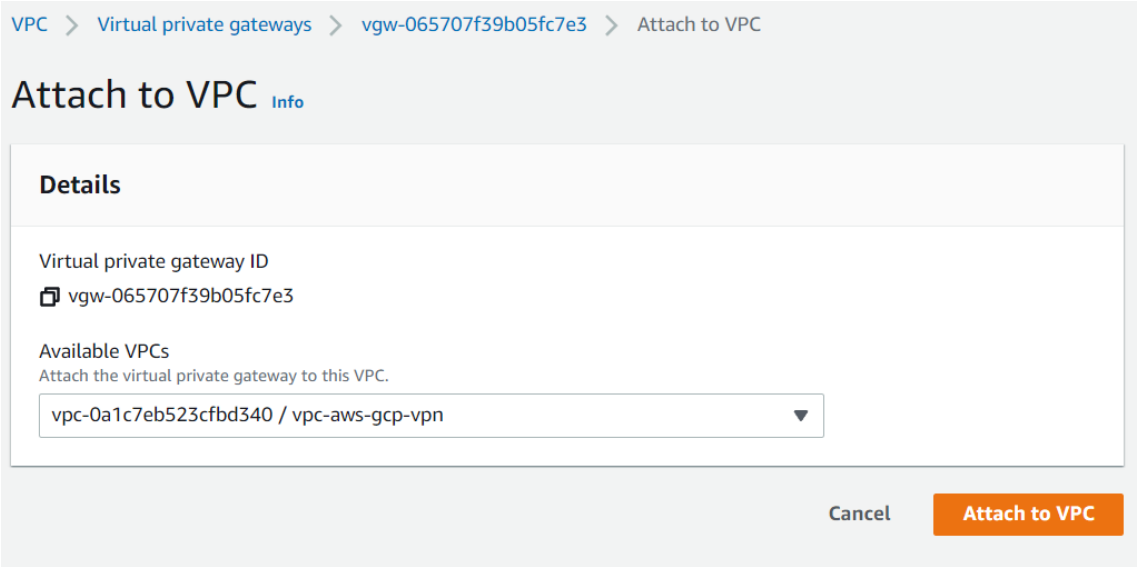

Ilustración 173 Configuración de la Virtual Private Gateway en AWS (4)

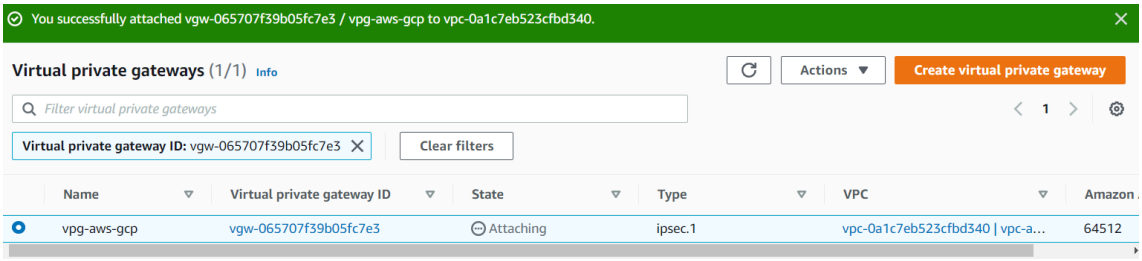

Ilustración 174 Configuración de la Virtual Private Gateway en AWS (5)

En vistas a evitar tener que añadir manualmente las rutas VPN a las tablas de rutas, activaremos la opción “Route propagation”. Esto permitirá que la puerta de enlace privada virtual propague las rutas de forma automática a las tablas de rutas.

Tendremos que movernos a “Virtual private cloud -> Route tables”, pinchamos en la tabla de rutas de nuestra VPC y en el apartado “Route propagation” podremos observar que la propagación de rutas de la VGW que acabamos de crear está desactivada. Paras activarla seleccionamos “Edit route propagation” y clickamos en activar la propagación en la puerta de enlace correspondiente:

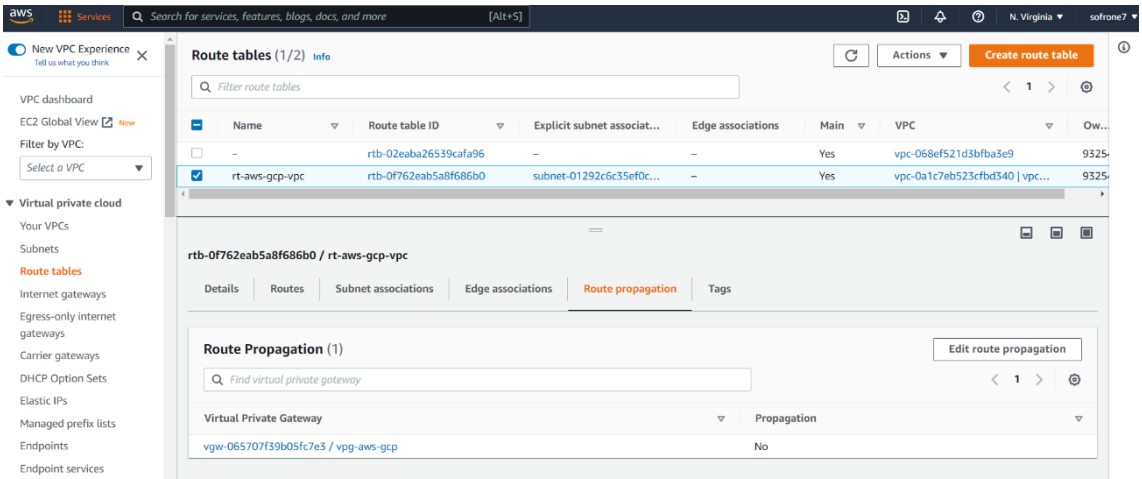

Ilustración 175 Configuración de la Virtual Private Gateway en AWS (6)

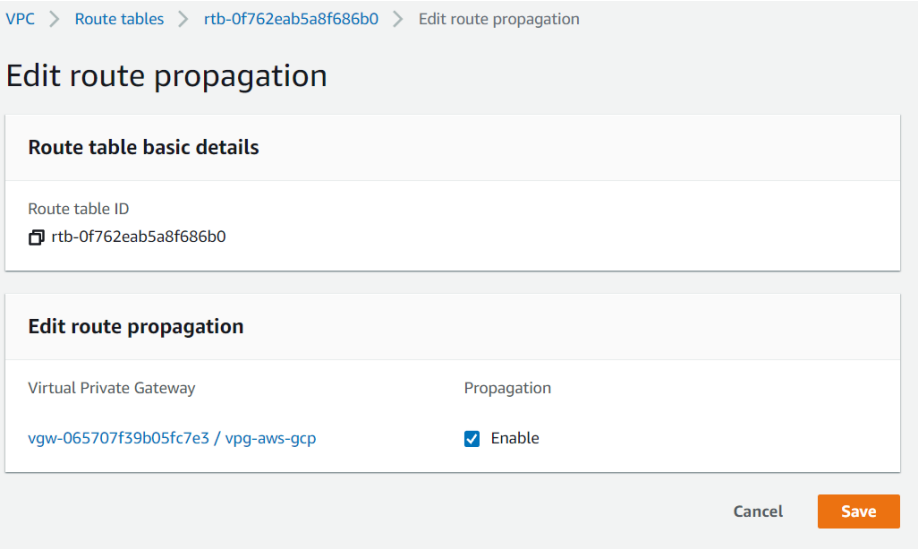

Ilustración 176 Configuración de la Virtual Private Gateway en AWS (7)

7.4.3. Crear las conexiones VPN Site-to-Site

Una vez generada la puerta de enlace privada virtual y las dos de cliente ya podemos crear las conexiones VPN de Amazon. Para ellos, nos dirigimos a "Virtual private network (VPN) -> Site-to-Site VPN Connections" y seleccionamos "Create VPN connection".

A la hora de crear una nueva conexión VPN primero se nos abrirá una primera pestaña donde deberemos seleccionar los recursos a partir de los cuáles se va a generar dicha conexión, así como opciones de configuración adicionales. Para cada una de las dos conexiones VPN que necesitamos para nuestro entorno, tendremos en cuenta las siguientes opciones:

- Un nombre que nos permita distinguir fácilmente una conexión de otra, por ejemplo, "vpn-aws-gcp-1" y "vpn-aws-gcp-2".
- Seleccionamos "Virtual private gateway" como tipo de puerta de enlace de destino. De esta forma, podremos elegir el identificador de la puerta de enlace privada virtual que recién creamos ("vpg-aws-gcp") para que trabaje como puerta de enlace de destino.
- En el apartado "Customer gateway" clickamos en "Existing". Seguido, relacionamos las conexiones con cada una de las puertas de enlace de cliente que tenemos a nuestra disposición, "vpn-aws-gcp-1" con "gc-aws-gcp-vpn-1" y "vpn-aws-gcp-2" con "gc-aws-gcp-vpn-2".
- Por último, en las opciones de enrutamiento seleccionamos "Dynamic", para utilizar el enrutamiento BGP dinámico y así determinar los rangos de CIDR de cada VPC. Esto es posible gracias a que el dispositivo de gateway de cliente (es decir, el Cloud HA VPN gateway del lado de Google) admite el protocolo ASN, ya que configuramos el Cloud Router para ello.

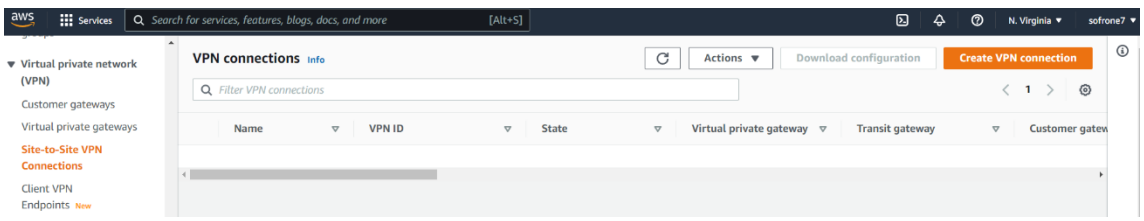

Ilustración 177 Configuración de las conexiones VPN Site-to-Site en AWS (1)

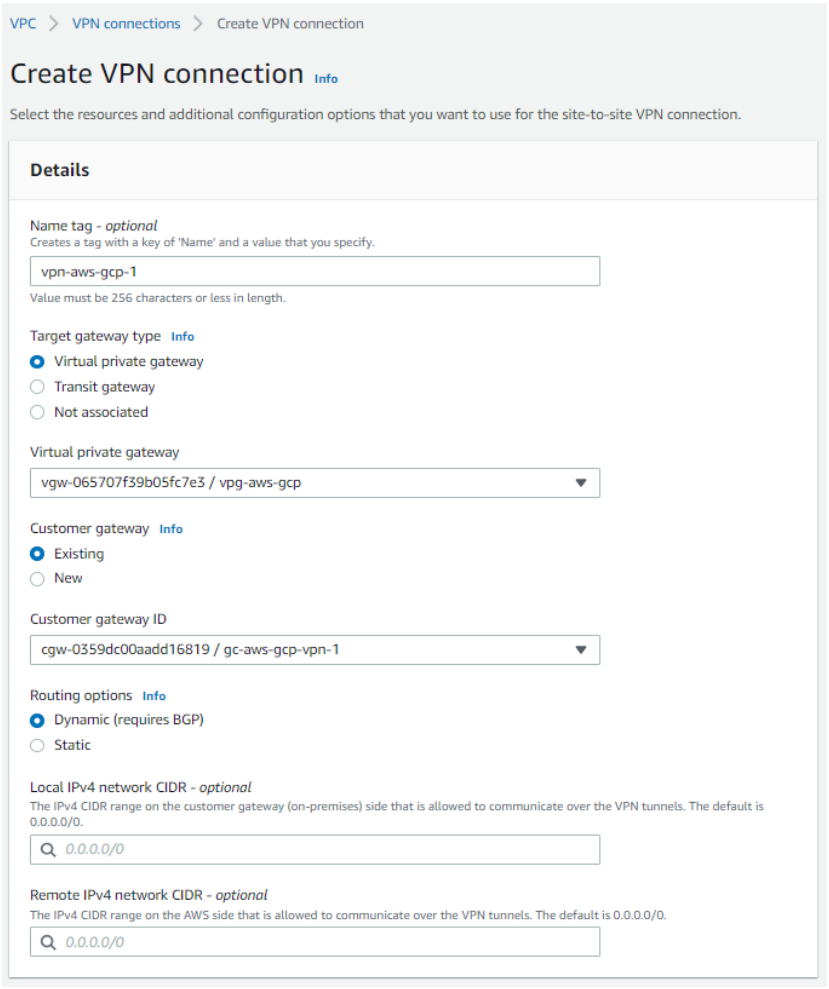

Ilustración 178 Configuración de las conexiones VPN Site-to-Site en AWS (2)

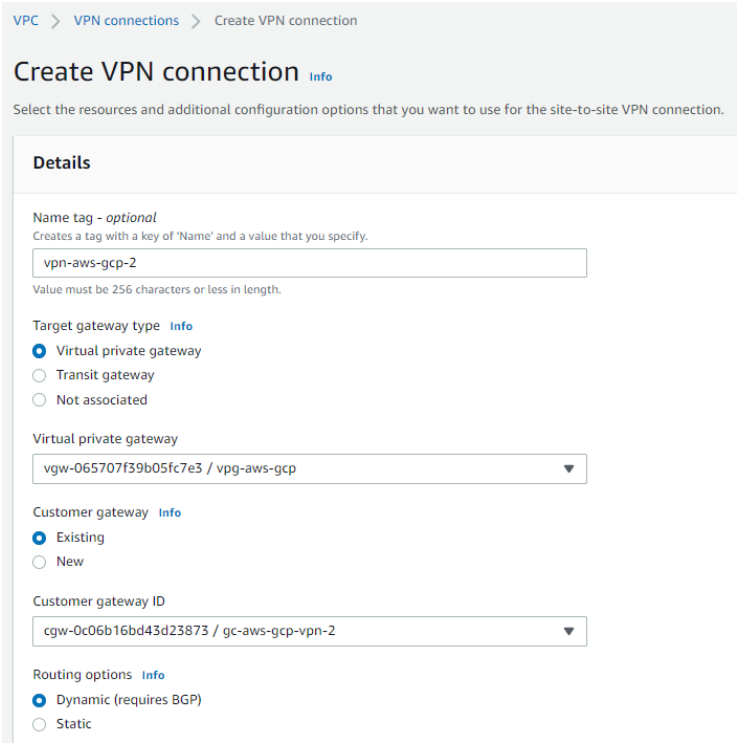

Ilustración 179 Configuración de las conexiones VPN Site-to-Site en AWS (3)

Una vez terminada la configuración inicial, podremos pasar a las opciones de los túneles. Cada conexión VPN incluye dos túneles VPN, se tratan de enlaces cifrados que pueden trabajar simultáneamente para transmitir datos desde la red albergada en Google Cloud hacia o desde AWS.

Utilizaremos la configuración muestra el usuario Tony Leung en su entorno [39] para ambos túneles. Utilizaremos el protocolo de encriptación IKEv2 y los algoritmos de cifrado y de integridad, así como los números de grupo Diffie-Hellman, que se permiten para la fase 1 y 2 de las negociaciones IKE.

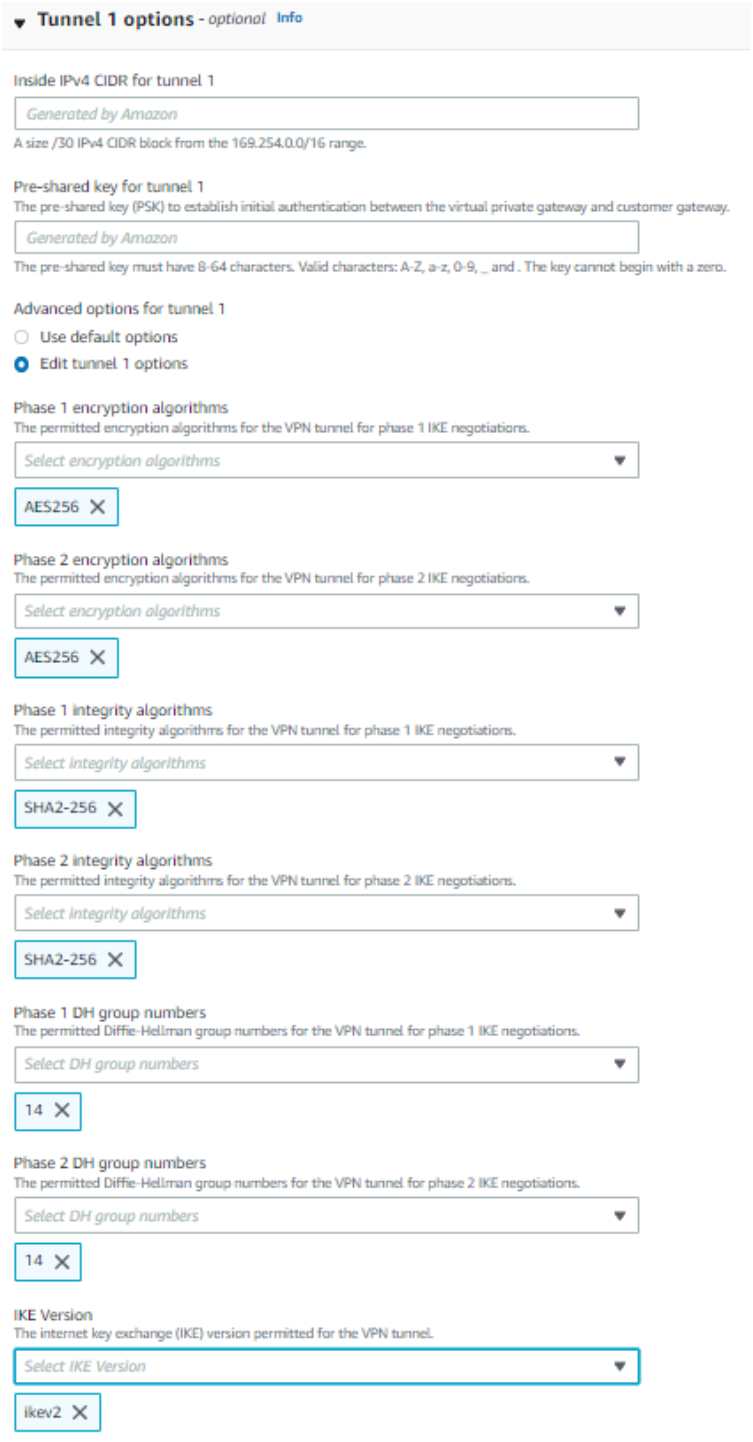

Ilustración 180 Configuración de las conexiones VPN Site-to-Site en AWS (3)

Una vez terminada la configuración y generadas las conexiones VPN Site-to-Site, podemos observar que el estado de ambas conexiones está caído:

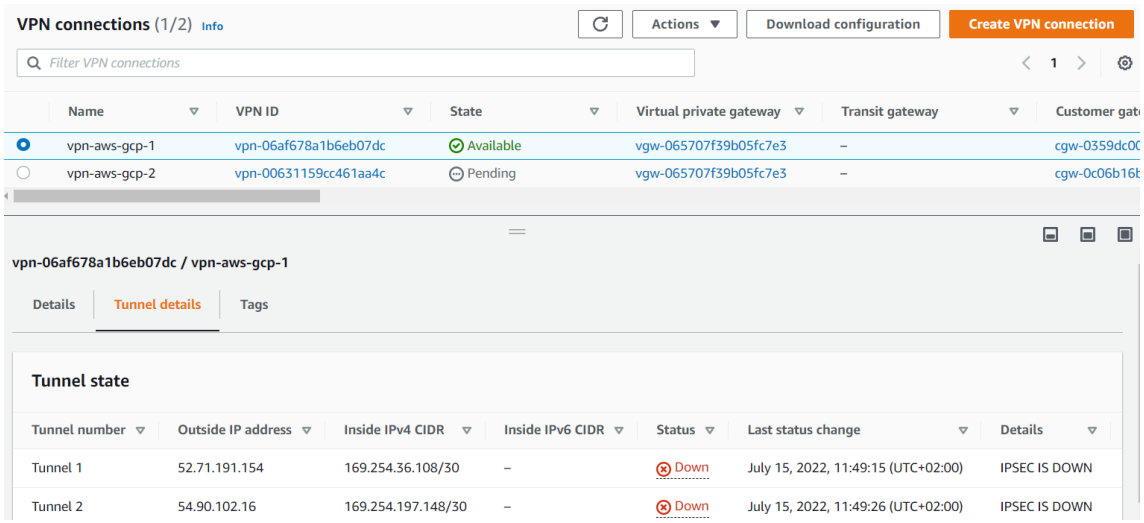

Ilustración 181 Configuración de las conexiones VPN Site-to-Site en AWS (4)

Esto se debe a que aún no hemos configurado el Cloud HA VPN gateway de GCP para conectar correctamente ambas VPC, debemos recordar que dejamos su configuración a medias para llegar justo a este punto. Para poder continuarla, debemos descargar dos archivos en AWS (uno por cada conexión VPN) de configuración específico para el dispositivo de puerta de enlace del cliente. Estos archivos contienen información para configurar el dispositivo, incluida la información para configurar cada túnel.

Pinchamos en “Download configuration” y seleccionamos las siguientes opciones:

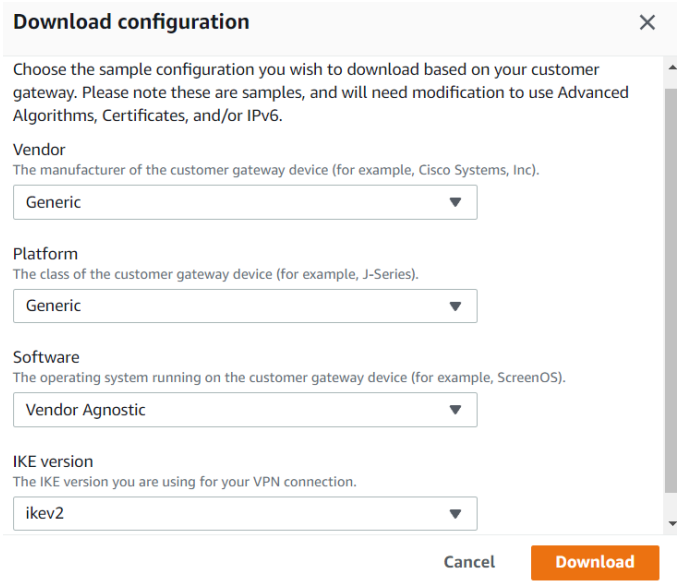

Ilustración 182 Descarga de los archivos de configuración de AWS

Se descargarán dos archivos; **1-vpn-06af678a1b6eb07dc.txt** y **2-vpn-00631159cc461aa4c.txt**. Debido a su extensión, en vez de incluirlos en el anexo estos archivos se han almacenado en el

repositorio https://github.com/sofrone7/archivos_tuneles_vpn. Una vez obtenidos estos ficheros de texto volvemos a GCP.

7.5. Continuación puerta de enlace de VPN en Google Cloud

7.5.1. Crear una peer gateway

Continuamos la configuración de la puerta de enlace Cloud HA VPN por donde la dejamos. En este apartado haremos uso de todos los recursos creados hasta el momento para, al fin, establecer la conexión entre ambas nubes.

Para ello debemos utilizar una "peer gateway", una puerta de enlace a la que se conectará la Cloud HA VPN gateway "vpn-gateway-aws-gcp". La "peer gateway" puede ser local o ajena a Google Cloud, en nuestro caso utilizaremos la puerta de enlace privada virtual que creamos en Amazon para las conexiones VPN. Por tanto, seleccionaremos "On-perm or Non Google Cloud" como tipo de puerta de enlace y "CREATE NEW PEER VPN GATEWAY" en el apartado "Peer VPN gateway name":

Ilustración 183 Configuración del VPN Gateway en GCP (1)

Le asignamos un nombre e indicamos el número de interfaces que tendrá la "peer gateway". Dado que en nuestra topología contamos con dos conexiones VPN Site-to-Site, y cada una cuenta con dos direcciones IP externas, se generaron cuatro IPs en total para la puerta de enlace

privada virtual en AWS. Por tanto, serán necesarias cuatro interfaces para realizar la conexión con dichas direcciones:

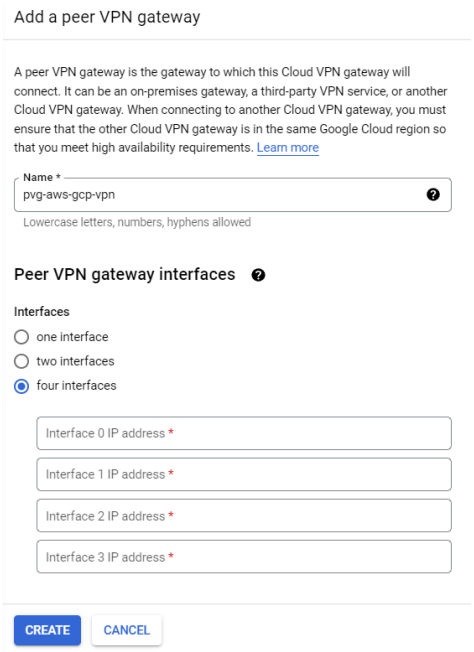

Ilustración 184 Configuración del VPN Gateway en GCP (2)

Las dos primeras interfaces estarán asociadas a la interfaz VPN externa de la Cloud HA VPN identificado con el valor 0 (0:35.242.6.99), mientras qué, las dos siguientes estarán asociadas a la interfaz de VPN externa 1 (1:35.220.11.213). De esta forma, uniremos la puerta de enlace de cliente de AWS "cg-aws-gcp-vpn-1" con la interfaz de VPN externa 0 y "cg-aws-gcp-vpn-2" con la interfaz 1, haciendo uso de sus respectivas IPs. Las direcciones IP para cada interfaz las encontramos en los archivos de configuración que descargamos al generar las conexiones VPN en AWS. A continuación, indicaré dónde buscar cada una:

- **Para la dirección IP del interfaz 0:** En el archivo "1-vpn" buscamos -> "IPSec Tunnel #1" -> "#3: Tunnel Interface Configuration" -> "Outside IP Addresses" -> "Virtual Private Gateway"

```
Outside IP Addresses:
  - Customer Gateway                : 35.242.6.99
  - Virtual Private Gateway         : 52.71.191.154
```

Ilustración 185 Configuración del VPN Gateway en GCP (3)

- **Para la dirección IP del interfaz 1:** En el archivo "1-vpn" buscamos -> "IPSec Tunnel #2" -> "#3: Tunnel Interface Configuration" -> "Outside IP Addresses" -> "Virtual Private Gateway"

```
Outside IP Addresses:
  - Customer Gateway                : 35.242.6.99
  - Virtual Private Gateway         : 54.90.102.16
```

Ilustración 186 Configuración del VPN Gateway en GCP (4)

- **Para la dirección IP del interfaz 2:** En el archivo "2-vpn" buscamos -> "IPSec Tunnel #1" -> "#3: Tunnel Interface Configuration" -> "Outside IP Addresses" -> "Virtual Private Gateway"

```
Outside IP Addresses:
  - Customer Gateway                : 35.220.11.213
  - Virtual Private Gateway         : 3.217.186.65
```

Ilustración 187 Configuración del VPN Gateway en GCP (5)

- **Para la dirección IP del interfaz 3:** En el archivo "2-vpn" buscamos -> "IPSec Tunnel #2" -> "#3: Tunnel Interface Configuration" -> "Outside IP Addresses" -> "Virtual Private Gateway"

```
Outside IP Addresses:
  - Customer Gateway                : 35.220.11.213
  - Virtual Private Gateway         : 54.157.75.96
```

Ilustración 188 Configuración del VPN Gateway en GCP (6)

Quedaría así:

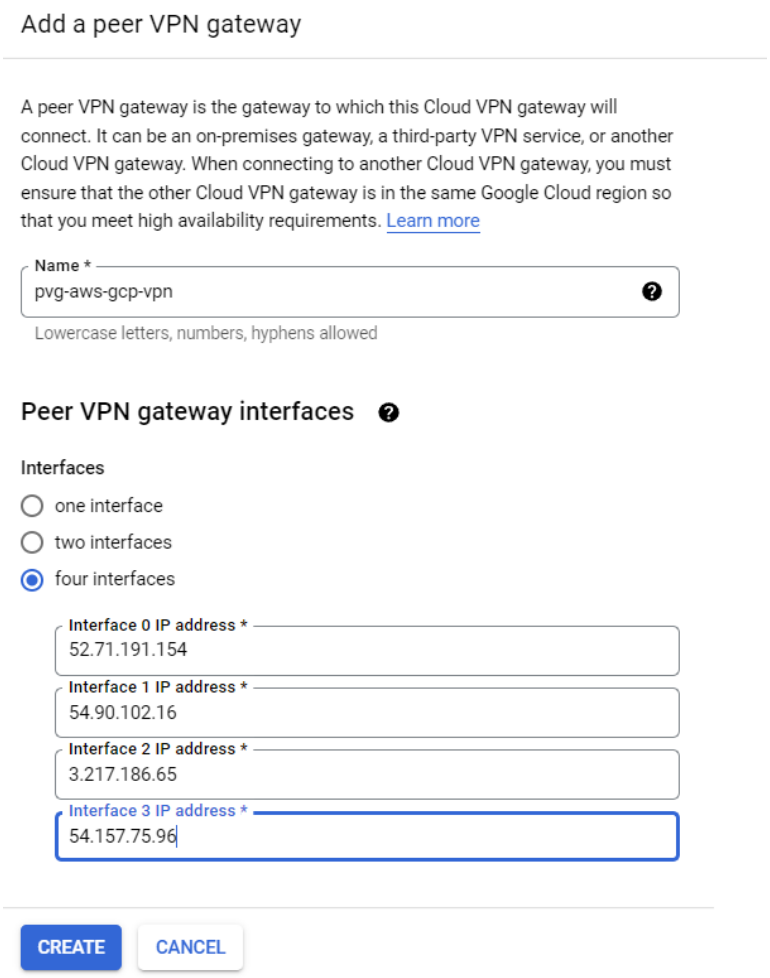

Ilustración 189 Configuración del VPN Gateway en GCP (7)

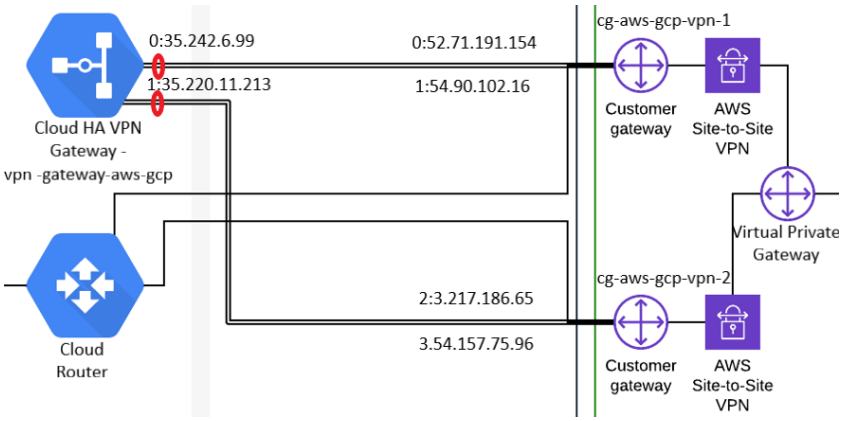

Ilustración 190 Esquema de conexiones

7.5.2. Crear túneles VPN

Una vez pinchemos en el botón "CREATE" se creará la "peer vpn gateway" y tendremos a nuestra disposición cuatro túneles VPN sin configurar. Antes de hacerlo, tenemos que relacionar esta puerta de enlace con el Cloud Router que configuramos en apartados anteriores para usar el protocolo de puerta de enlace fronteriza (BGP) e intercambiar rutas entre la red VPC de Google y la de Amazon:

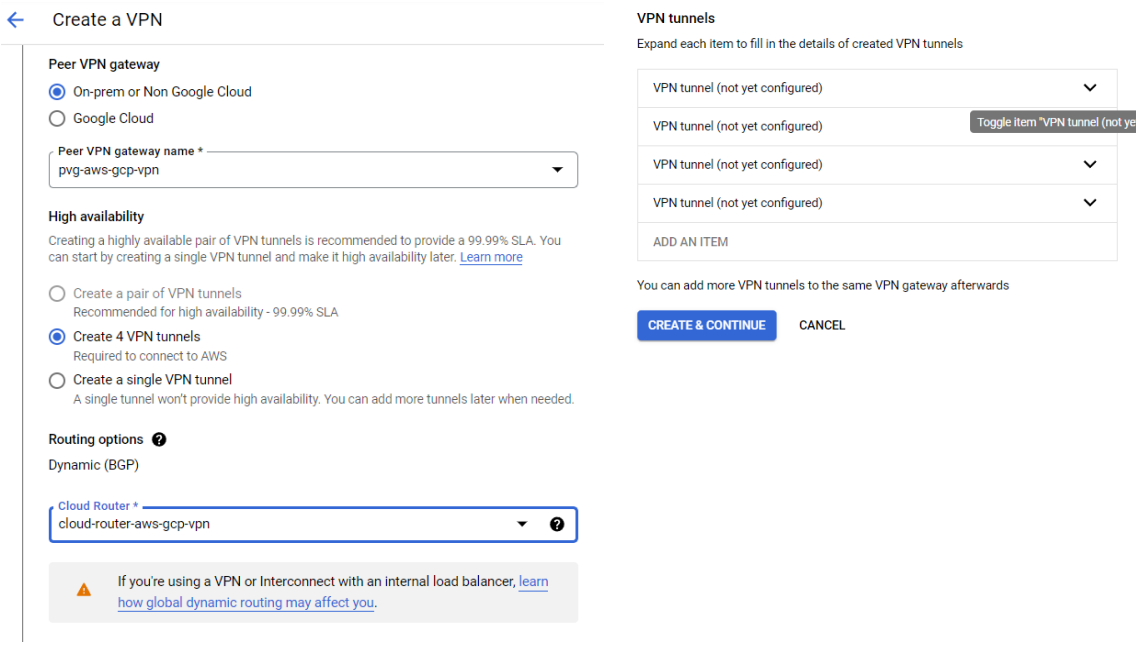

Ilustración 191 Configuración de los túneles VPN (1)

Ahora sí podremos acceder a cada túnel y editarlo. Le otorgaremos a cada uno un nombre distintivo según el túnel AWS al que está relacionado (vpn-tunnel-1-1, vpn-tunnel-1-2, vpn-tunnel-2-1 y vpn-tunnel-2-2) y les especificamos una clave precompartida IKE. Las claves deben coincidir con las claves precompartidas de cada túnel asociadas que creamos en las conexiones VPN de Amazon. Recordemos que dicha información la encontramos en sus archivos de configuración:

- **Para el túnel vpn-tunnel-1-1:** En el archivo "1-vpn" buscamos -> "IPSec Tunnel #1" -> "Pre-Shared Key"

```
- IKE version             : IKEv2
- Authentication Method   : Pre-Shared Key
- Pre-Shared Key          : sc6j6eQSB13ovP..3grRXu0nHtpNdMcQ
- Authentication Algorithm : sha1
- Encryption Algorithm    : aes-128-cbc
- Lifetime                : 28800 seconds
- Phase 1 Negotiation Mode : main
- Diffie-Hellman          : Group 2
```

Ilustración 192 Configuración de los túneles VPN (2)

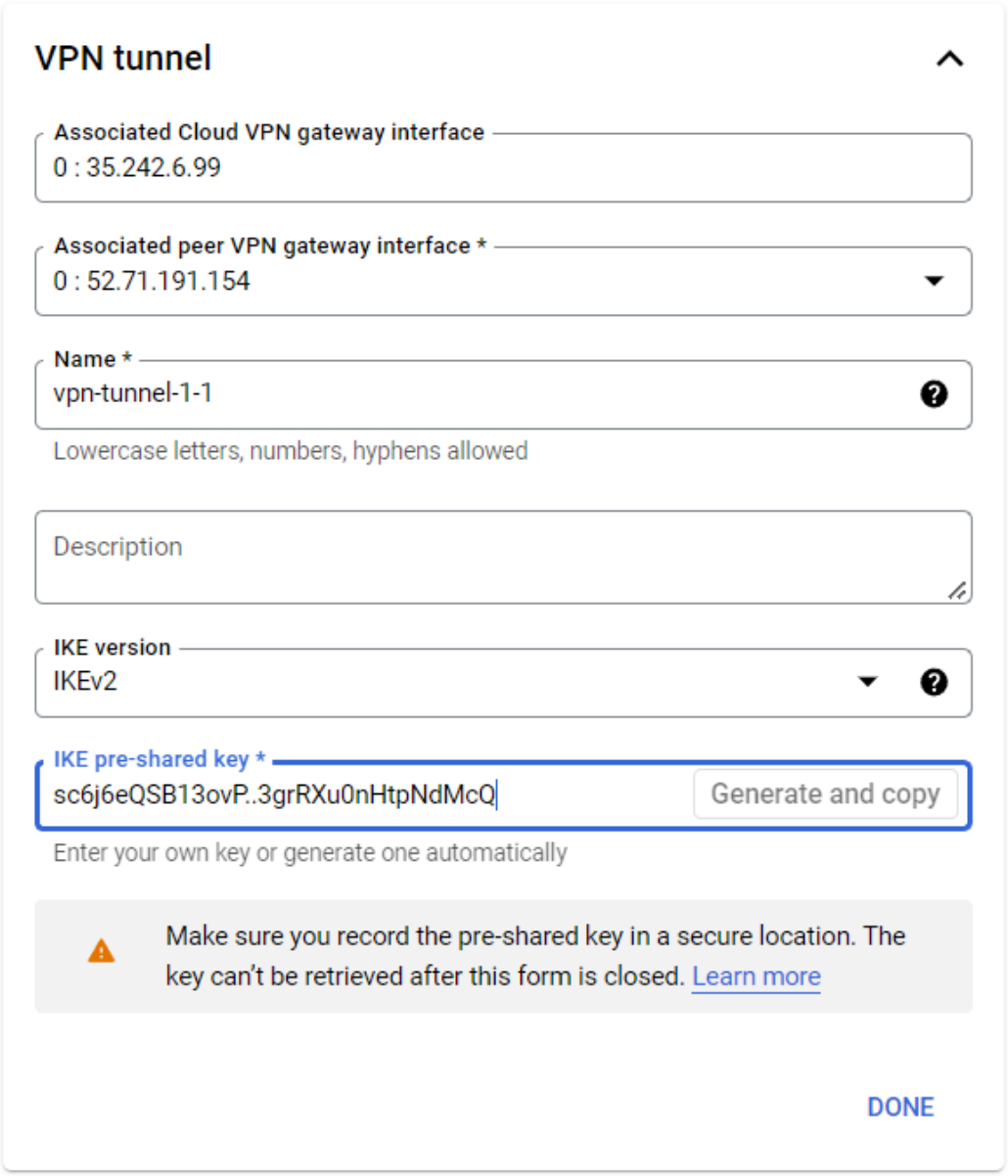

Ilustración 193 Configuración de los túneles VPN (3)

- **Para el túnel vpn-tunnel-1-2:** En el archivo "1-vpn" buscamos -> "IPSec Tunnel #2" -> "Pre-Shared Key"

```
- IKE version                : IKEv2
- Authentication Method      : Pre-Shared Key
- Pre-Shared Key             : AsHpX6HzxlIUlSJOEz7O6j4IBtY77yGZ
- Authentication Algorithm   : sha1
- Encryption Algorithm       : aes-128-cbc
- Lifetime                   : 28800 seconds
- Phase 1 Negotiation Mode   : main
- Diffie-Hellman             : Group 2
```

Ilustración 194 Configuración de los túneles VPN (4)

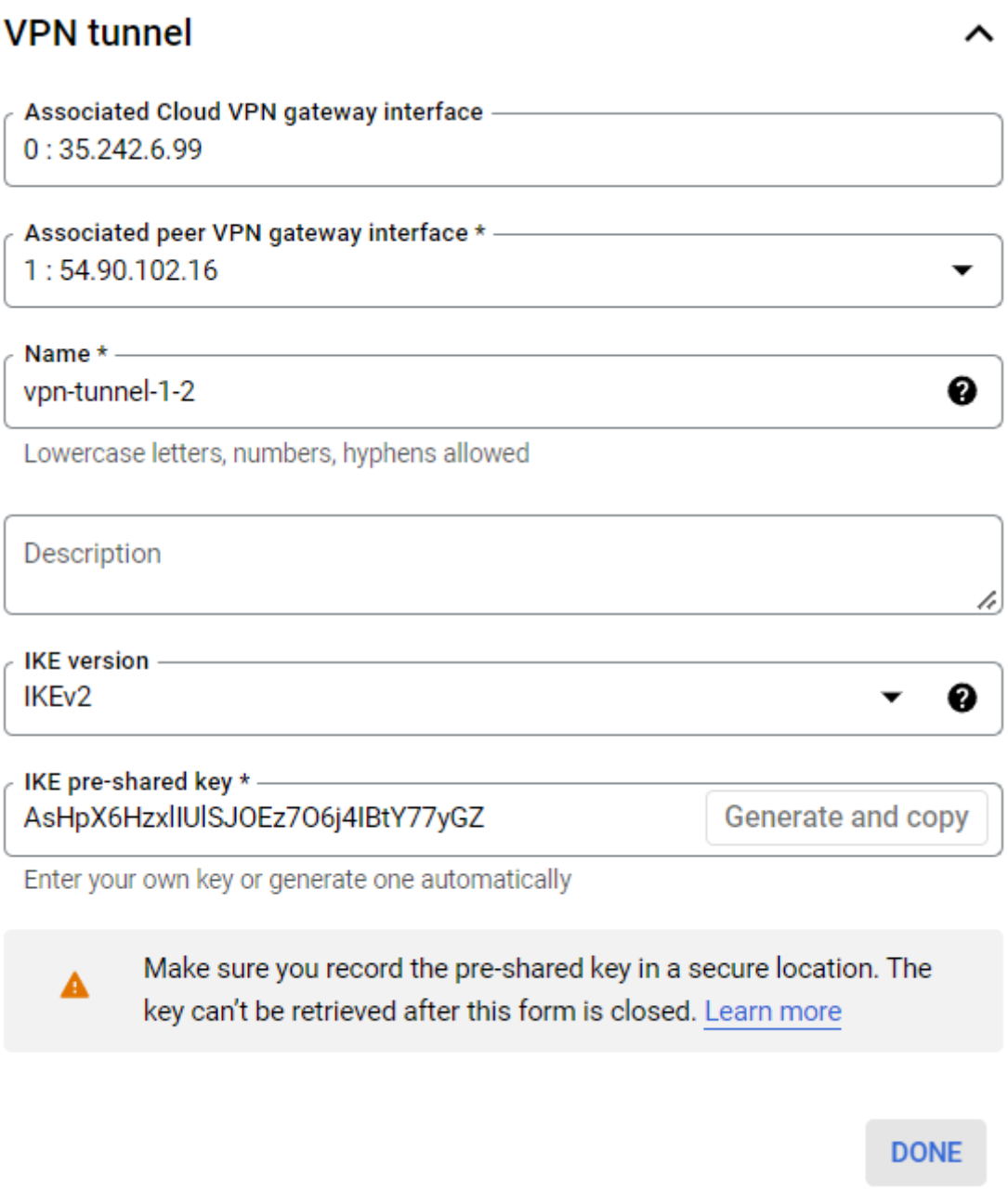

Ilustración 195 Configuración de los túneles VPN (5)

- **Para el túnel vpn-tunnel-2-1:** En el archivo “2-vpn” buscamos -> “IPSec Tunnel #1” -> “Pre-Shared Key”

```
- IKE version             : IKEv2
- Authentication Method   : Pre-Shared Key
- Pre-Shared Key          : e5gbOVMKLXggM4FjGsA_teD2eGiLr.te
- Authentication Algorithm : sha1
- Encryption Algorithm    : aes-128-cbc
- Lifetime                : 28800 seconds
- Phase 1 Negotiation Mode : main
- Diffie-Hellman          : Group 2
```

Ilustración 196 Configuración de los túneles VPN (6)

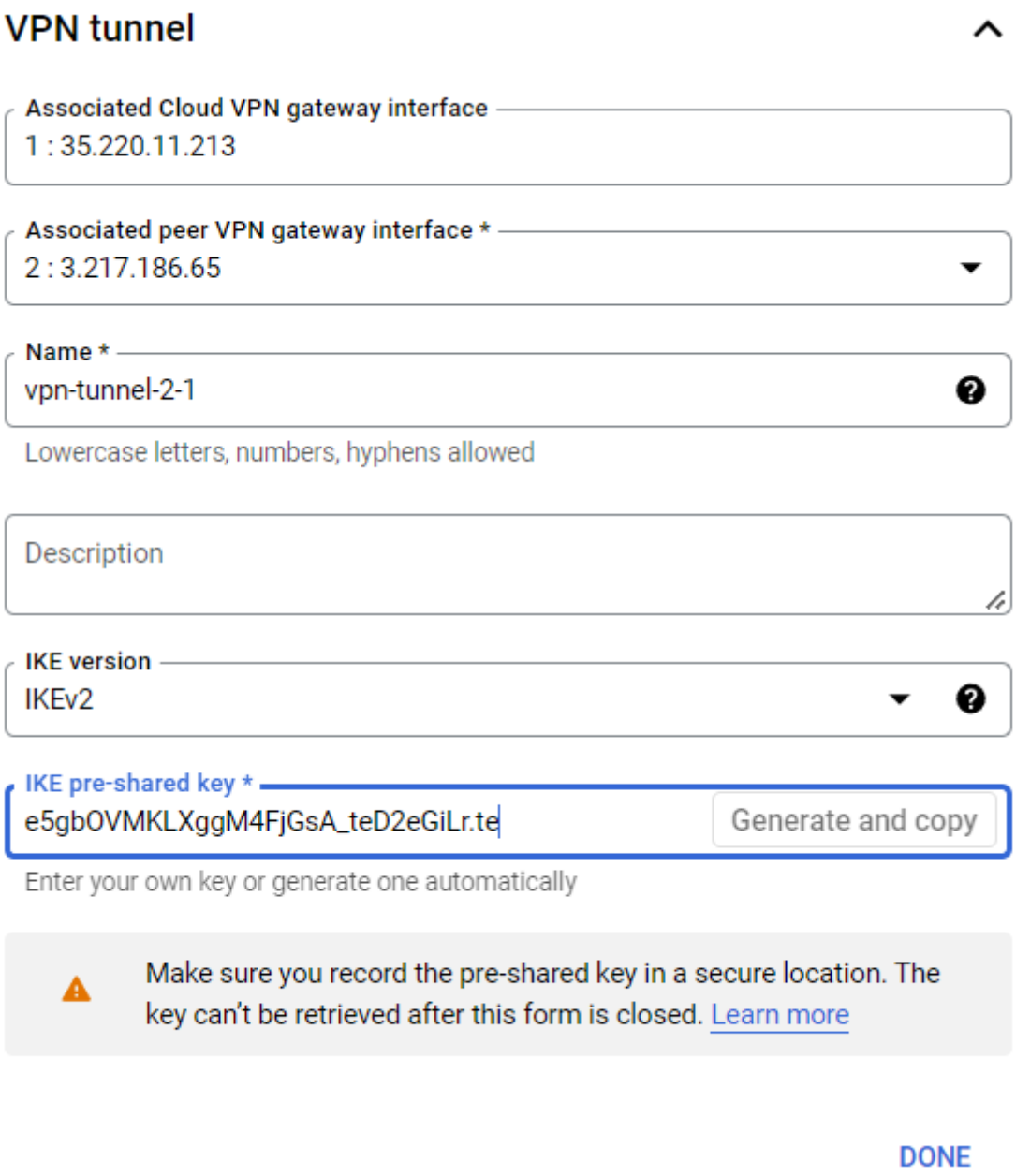

Ilustración 197 Configuración de los túneles VPN (7)

- **Para el túnel vpn-tunnel-2-2:** En el archivo “2-vpn” buscamos -> “IPSec Tunnel #2” -> “Pre-Shared Key”

```
- IKE version              : IKEv2
- Authentication Method    : Pre-Shared Key
- Pre-Shared Key           : h3dlVe6gG3EIgemXez_5HlMzBERzD.58
- Authentication Algorithm : sha1
- Encryption Algorithm     : aes-128-cbc
- Lifetime                 : 28800 seconds
- Phase 1 Negotiation Mode : main
- Diffie-Hellman           : Group 2
```

Ilustración 198 Configuración de los túneles VPN (8)

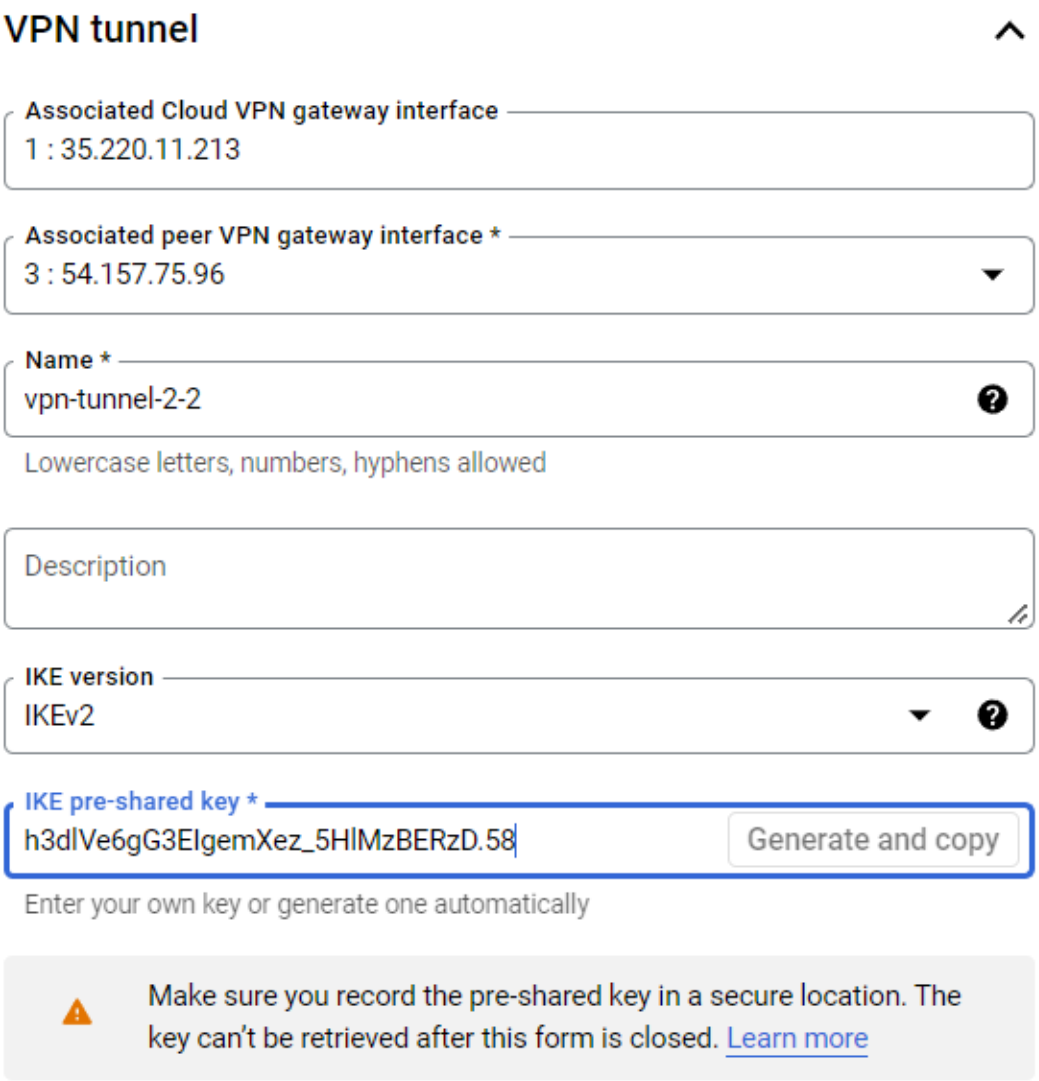

Ilustración 199 Configuración de los túneles VPN (9)

Una vez los túneles VPN estén configurados pinchamos en “CREATE & CONTINUE”:

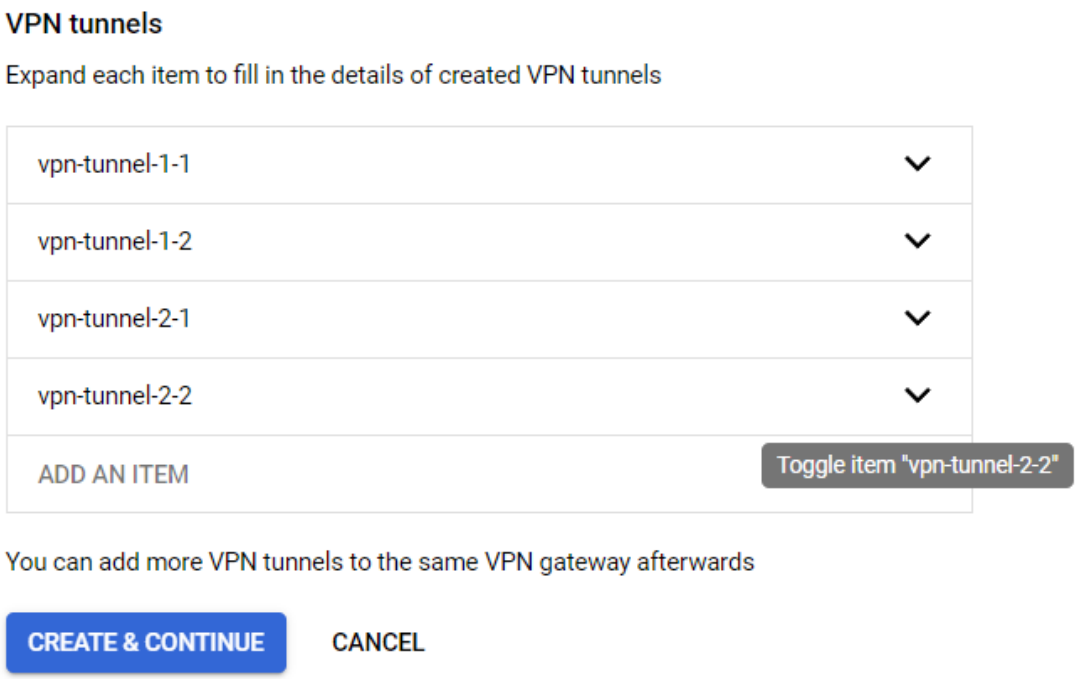

Ilustración 200 Configuración de los túneles VPN (10)

7.5.3. Configurar las sesiones BGP

Una vez creados los túneles VPN deberíamos ver la siguiente pantalla:

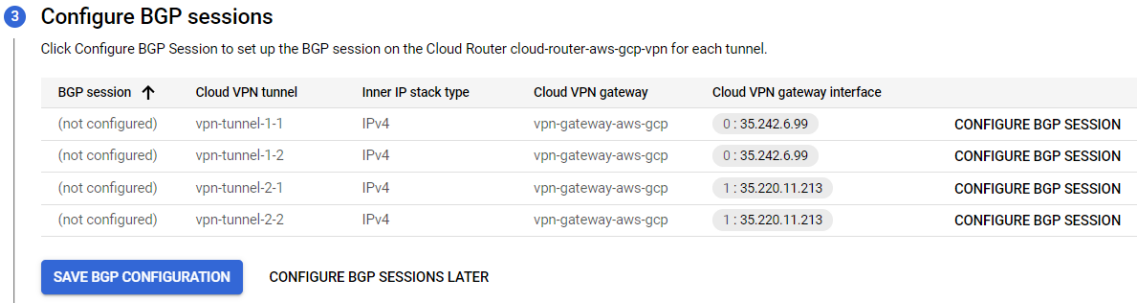

Ilustración 201 Configuración de las sesiones BGP (1)

Desde aquí configuraremos las sesiones BGP de cada túnel pinchando en “CONFIGURE BGP SESSION”. Será necesario otorgarle un nombre a cada sesión (bgp-tunnel-1-1, bgp-tunnel-1-2, bgp-tunnel-2-1 y bgp-tunnel-2-2) y el número ASN relacionado al “peer gateway”, es decir, el ASN configurado para la puerta de enlace privada virtual de AWS. Recordemos que podemos encontrarlo en “Virtual private network -> Virtual private gateways -> Amazon ASN”:

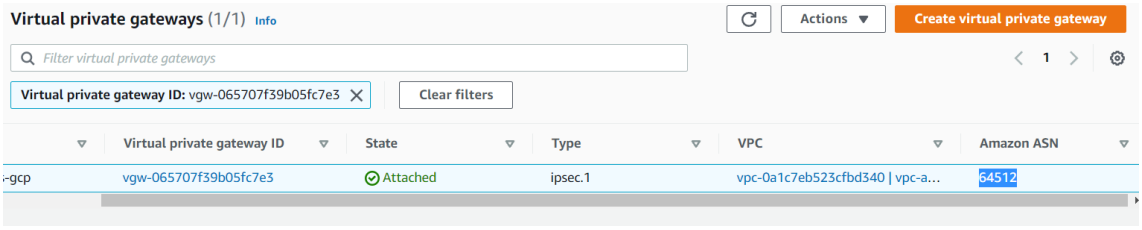

Ilustración 202 Número ASN del Virtual Private Gateway en AWS

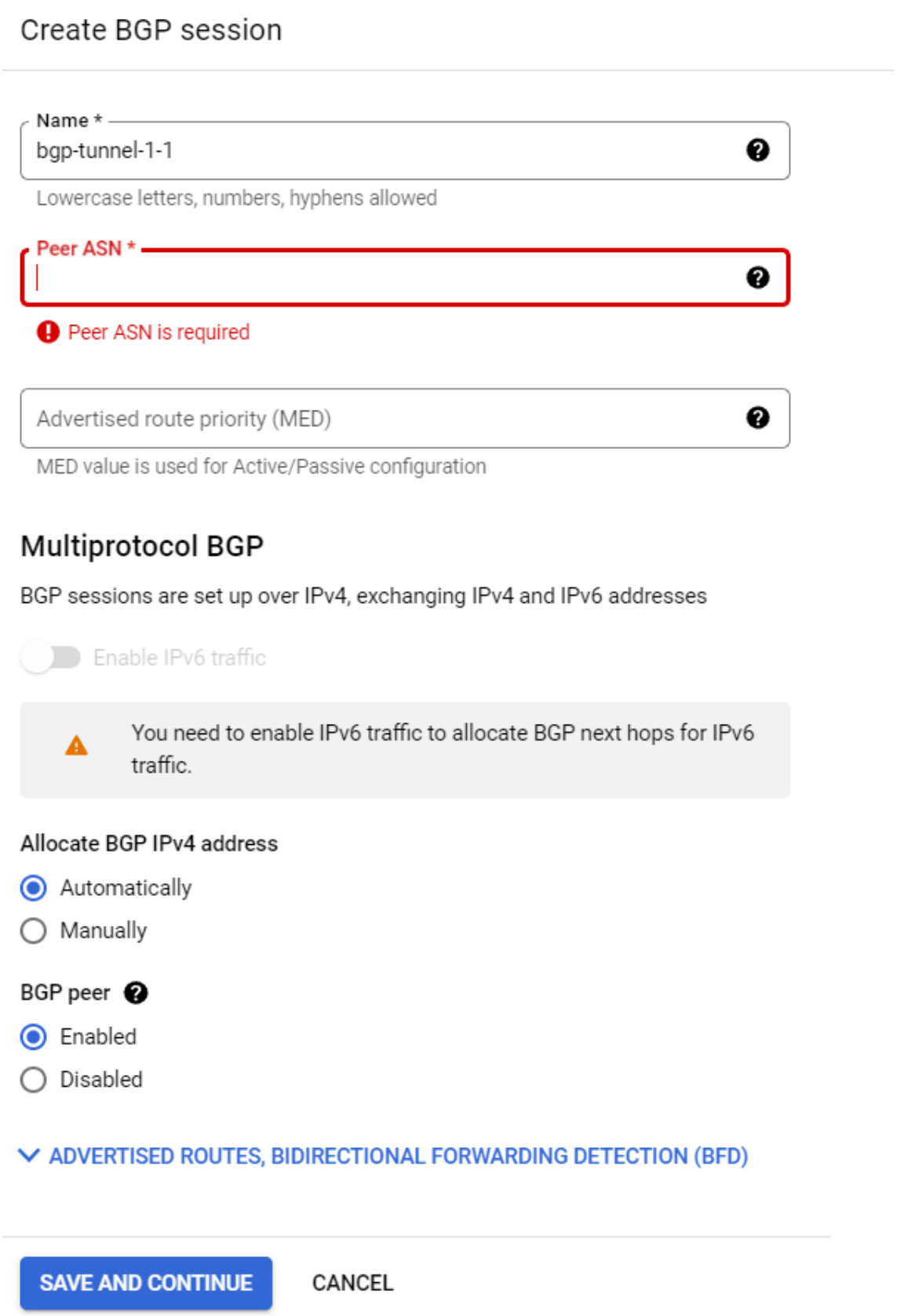

Ilustración 203 Configuración de las sesiones BGP (2)

Las sesiones BGP se establecen sobre IPv4, nosotros las asignaremos de forma manual a partir de los archivos de configuración de VPN de AWS. Necesitamos la dirección de la puerta de enlace de cliente y privada virtual de cada túnel, las introduciremos en “Cloud Router BGP IP” y en “BGP

Peer IP" respectivamente. Deben ser direcciones locales de enlace que pertenezcan a un CIDR /30 del bloque 169.254.0.0/16.

Para encontrar dichas direcciones nos dirigiremos al archivo de configuración relacionado al túnel de cada sesión. Buscamos "#3: Tunnel Interface Configuration" -> "Inside IP Addresses" -> "Customer Gateway" y "Virtual Private Gateway". Copiaremos sólo la dirección IP, sin la máscara:

```
Inside IP Addresses
  - Customer Gateway                : 169.254.36.110/30
  - Virtual Private Gateway         : 169.254.36.109/30
```

Ilustración 204 Configuración de las sesiones BGP (3)

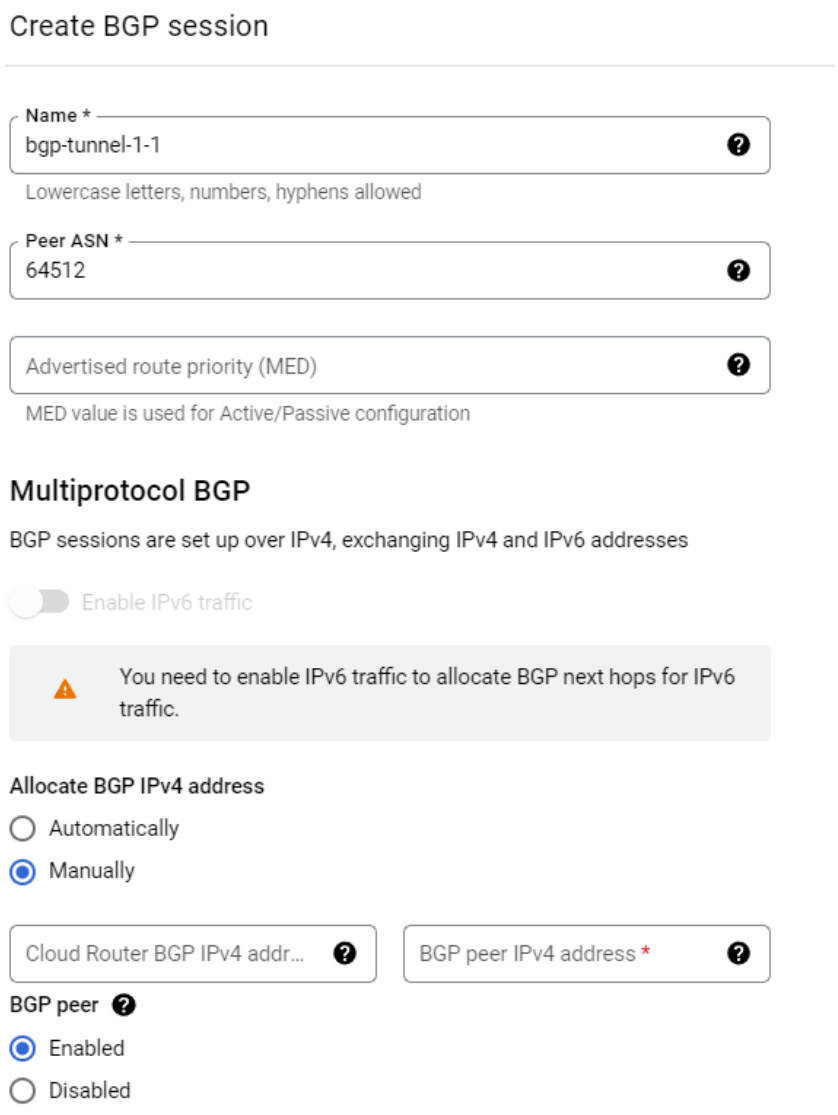

Ilustración 205 Configuración de las sesiones BGP (4)

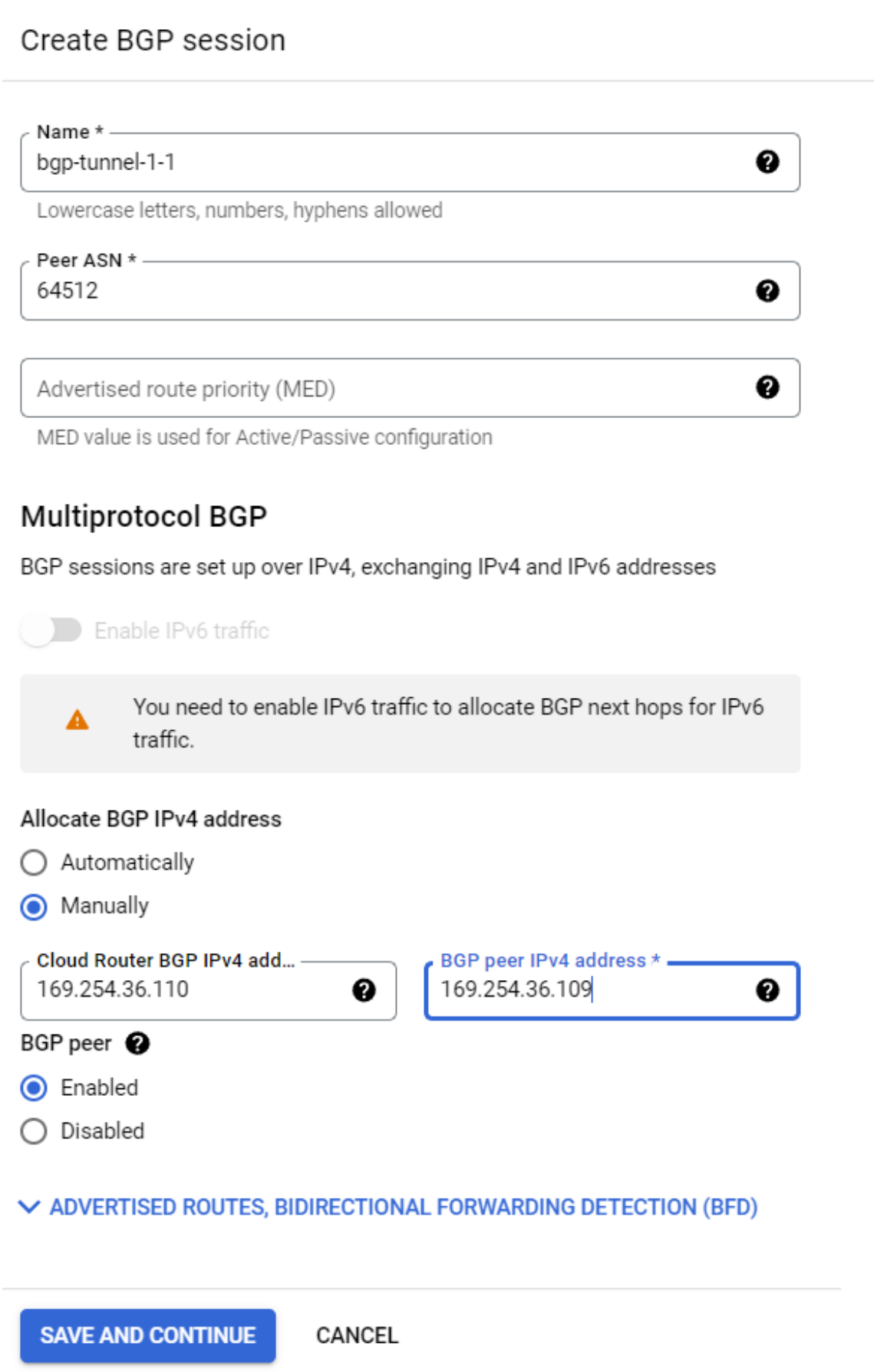

Ilustración 206 Configuración de las sesiones BGP (5)

Repetimos este proceso para todos los túneles:

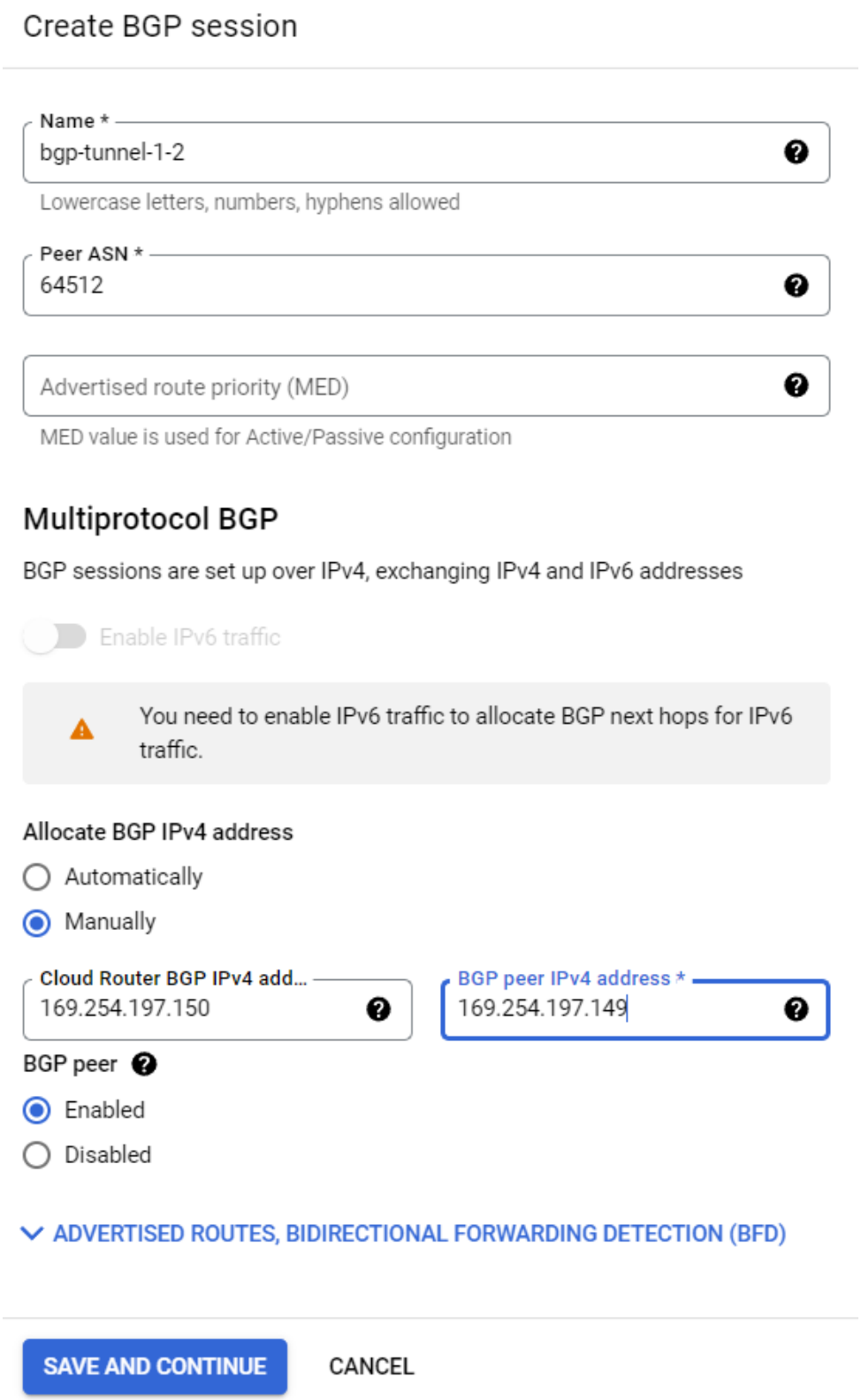

Ilustración 207 Configuración de las sesiones BGP (6)

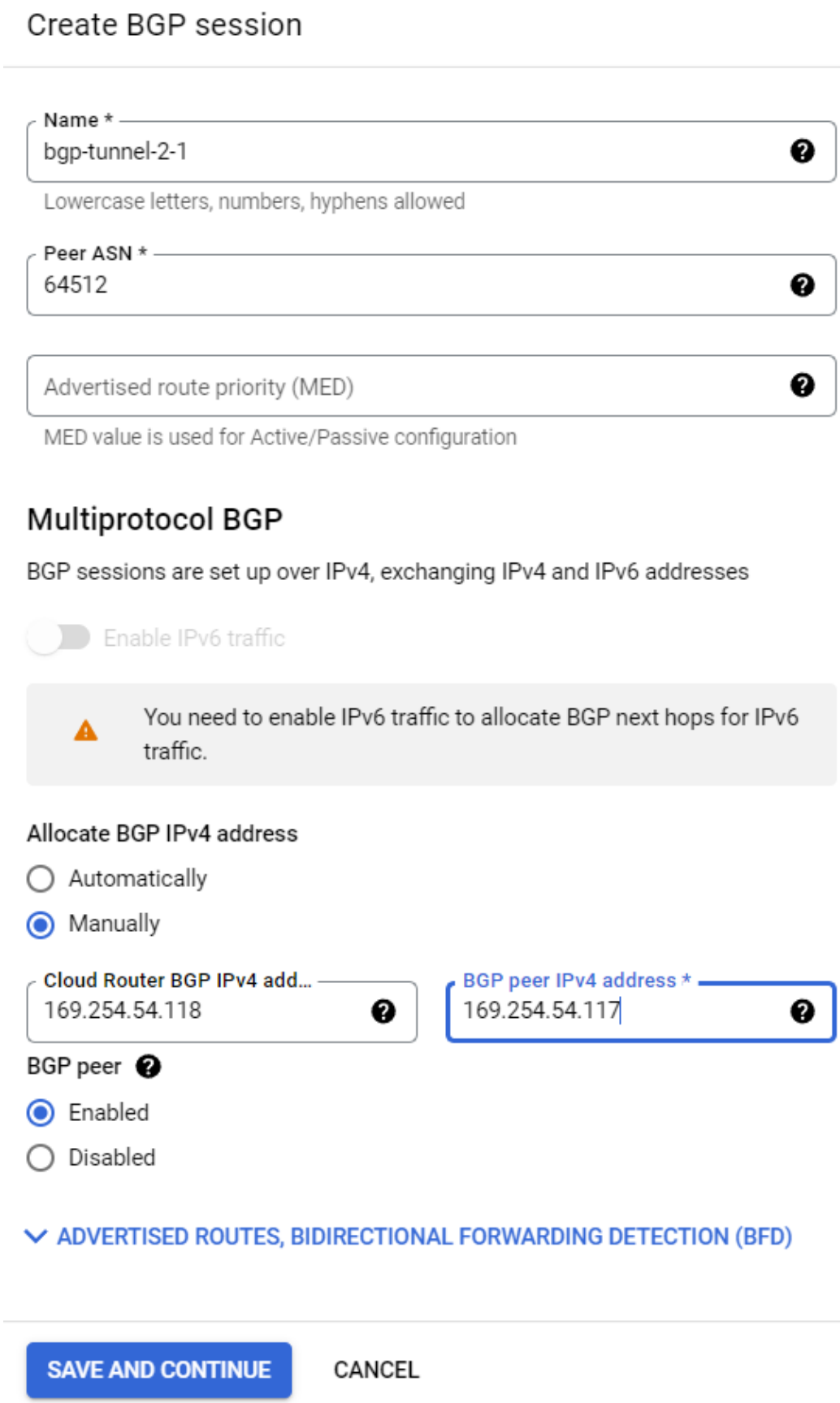

Ilustración 208 Configuración de las sesiones BGP (7)

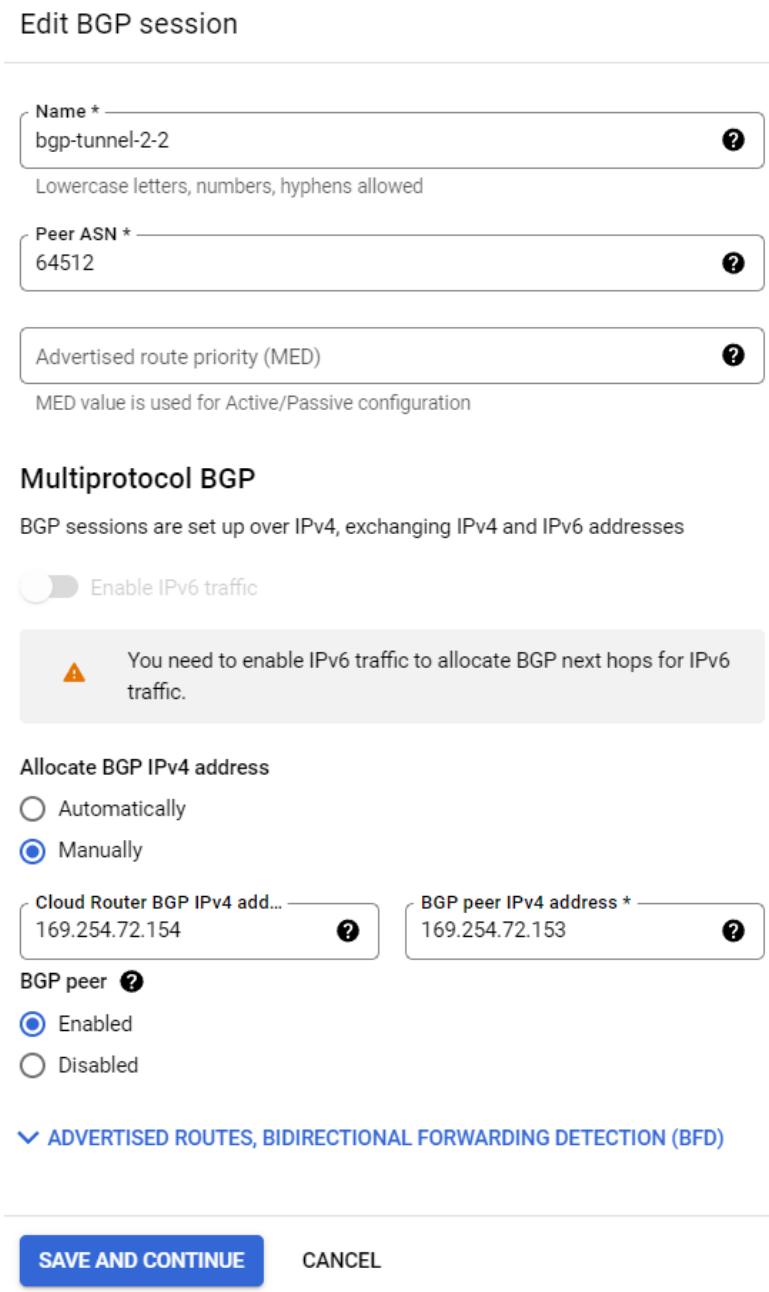

Ilustración 209 Configuración de las sesiones BGP (8)

Una vez terminada la configuración BGP pinchamos en “SAVE BGP CONFIGURATION”:

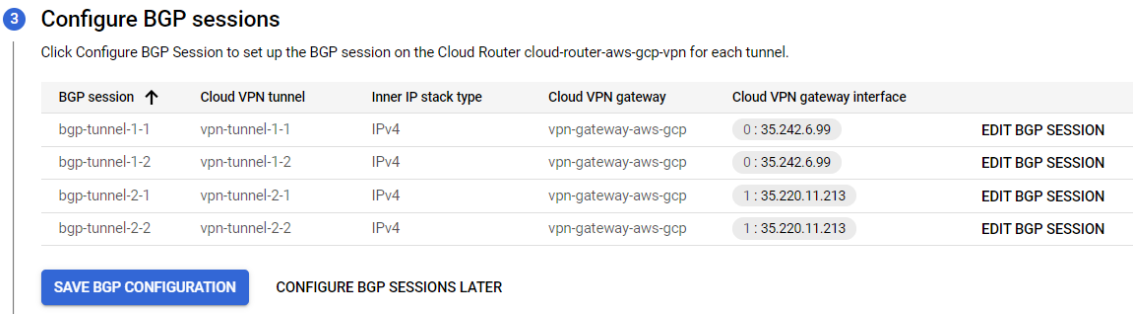

Ilustración 210 Configuración de las sesiones BGP (9)

A continuación, observaremos un resumen de la configuración de la VPN. Si esperamos unos minutos deberíamos ver que el estado de los túneles VPN y las sesiones BGP indican "Established" para los cuatro túneles:

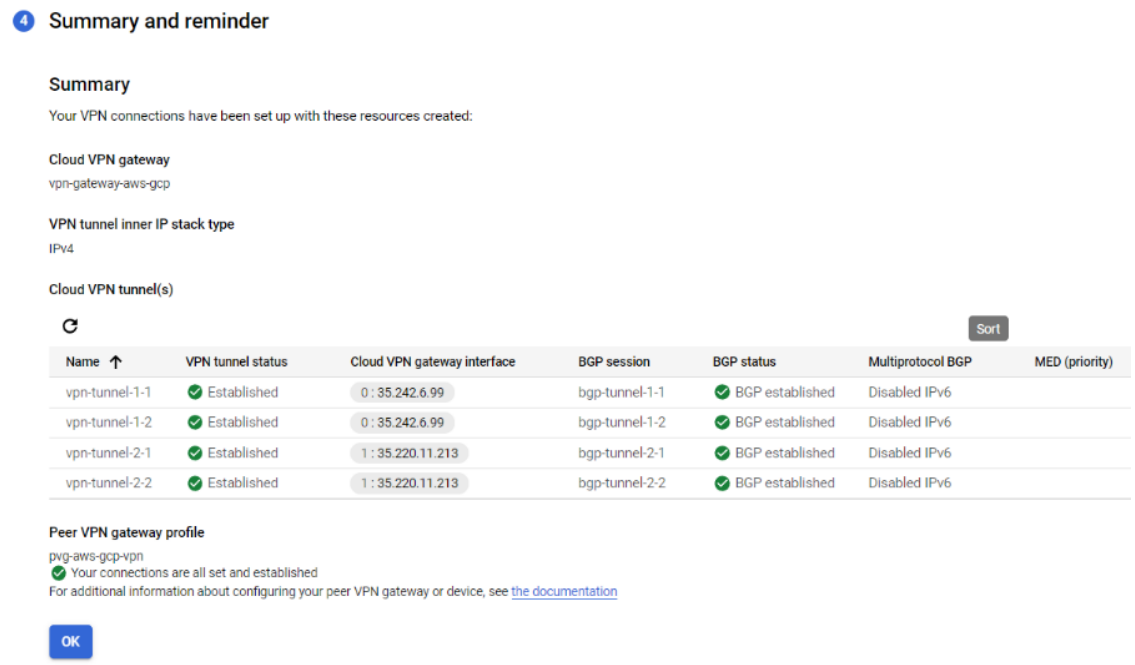

Ilustración 211 Comprobación del estado de los túneles en GCP

Y si volvemos a la consola de AWS, en “Virtual private network (VPN) -> Site-to-Site VPN Connections”, ahora los túneles de cada conexión se encuentran en estado “Up”, indicándonos que la conexión entre ambas nubes se ha establecido correctamente:

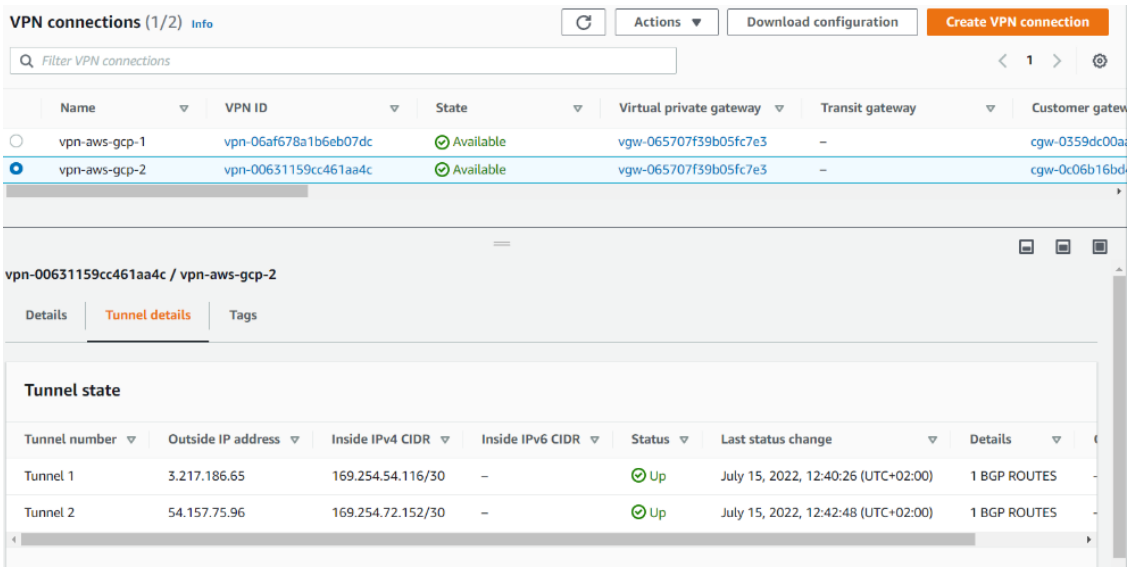

Ilustración 212 Comprobación del estado de los túneles en AWS

7.6. Test ICMP

Por último, vamos a verificar si la conexión entre ambas nubes funciona correctamente. Lanzaremos una instancia tanto en GCP como en AWS, y comprobaremos si se pueden comunicar a partir de su dirección IP privada.

7.6.1. Lanzar instancia en Amazon

Empezamos creando una instancia en Amazon, ubicada en la subnet de la VPC que hemos configurado para que tenga conexión con GCP. Nos dirigimos a "EC2 -> Instances" y pinchamos en "Launch instances". Utilizaremos una instancia con características mínimas, ya que sólo nos interesa hacer ping entre ellas, no necesitamos nada complejo:

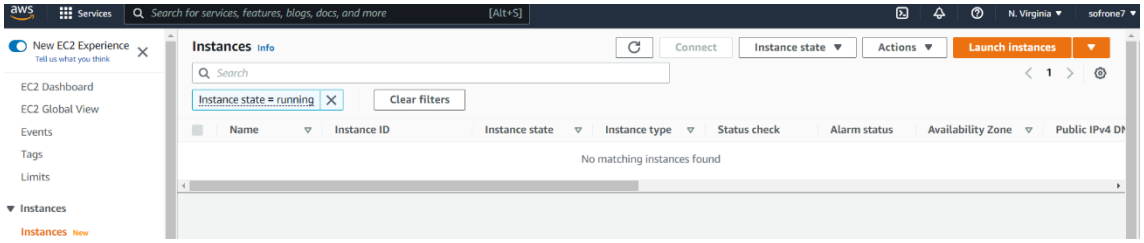

Ilustración 213 Creación de una instancia en AWS (1)

- Name: instance-aws
- OS Image: Amazon Linux

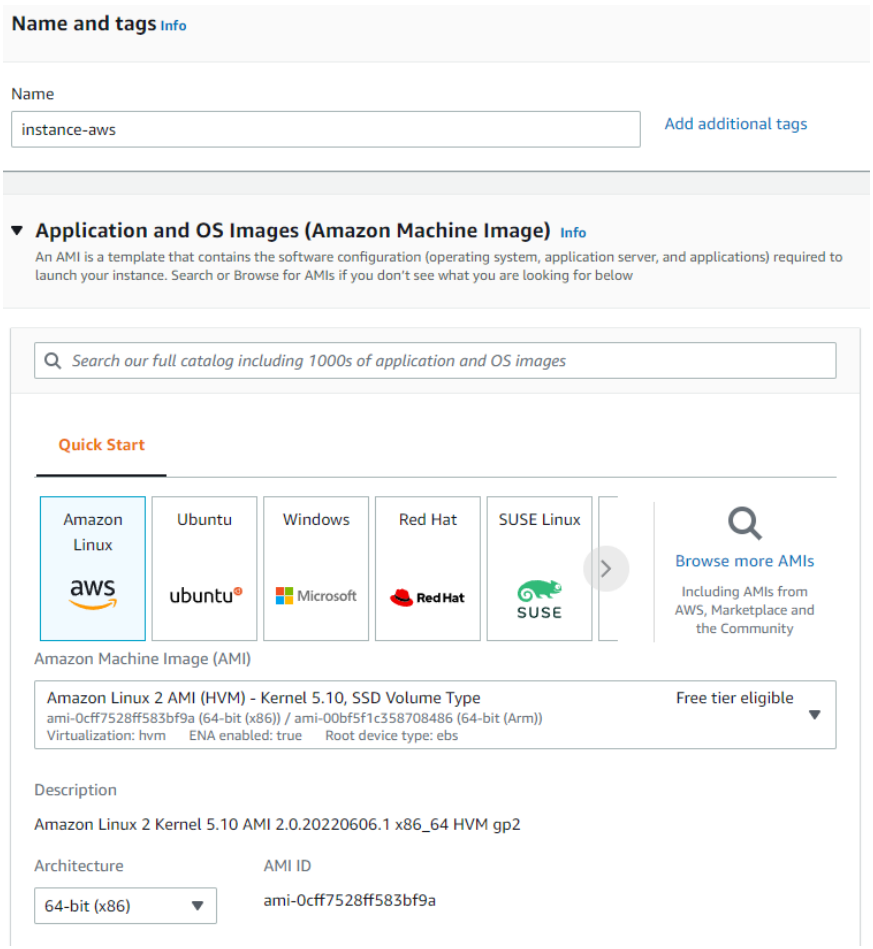

Ilustración 214 Creación de una instancia en AWS (2)

- Instance type: t2-micro

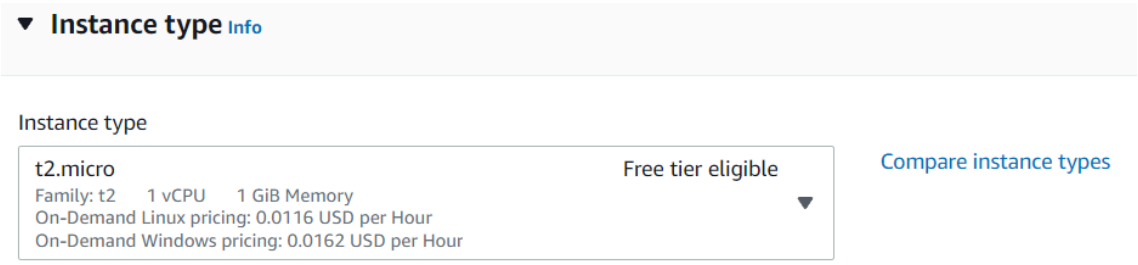

Ilustración 215 Creación de una instancia en AWS (3)

- Key pair: La configuración de la instancia requiere un par de claves relacionadas a ella. En nuestro caso las crearemos, ya que no contamos con unas.

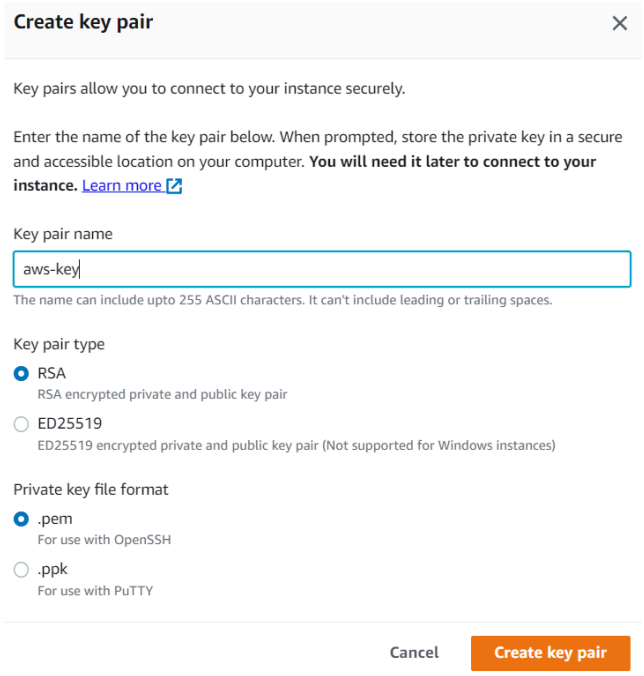

Ilustración 216 Creación de una instancia en AWS (4)

- VPC: vpc-aws-gcp-vpn
- Subnet: subnet-us-east1
- Auto-assign public IP: Enable

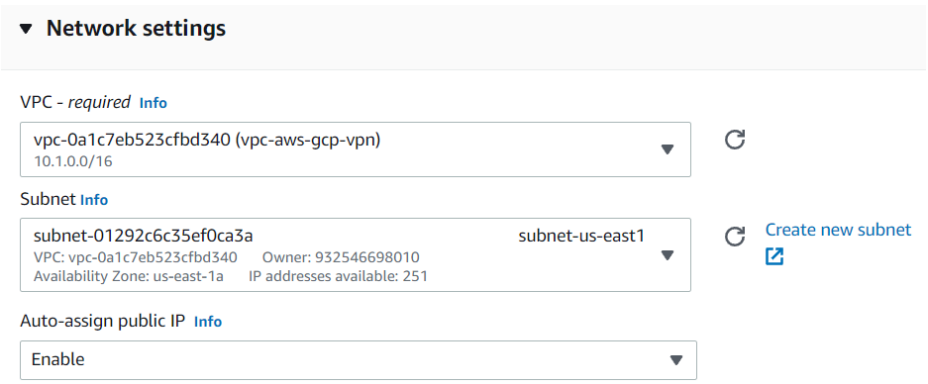

Ilustración 217 Creación de una instancia en AWS (5)

- Inbound security groups rules: En cuanto a las reglas que controlan el tráfico de entrada a las instancias, contamos con una regla predeterminada que nos permite realizar conexiones SSH desde cualquier dirección IP. Para un entorno de pruebas como el nuestro, dicha regla nos sirve para conectarnos a la máquina sin mucha complicación desde la consola de Amazon así que la dejaremos. Sin embargo, debemos ser conscientes de que en un caso real puede ser muy peligroso. Es por ello se recomienda permitir el acceso sólo a direcciones IP conocidas, por ejemplo, a la dirección IP de nuestro equipo (Source: My IP), desde la cual podremos utilizar nuestro par de claves para conectarnos de forma segura. También será necesario añadir una regla que nos permita responder a cualquier dirección que nos haga ping y así poder comprobar el correcto funcionamiento de nuestra red:

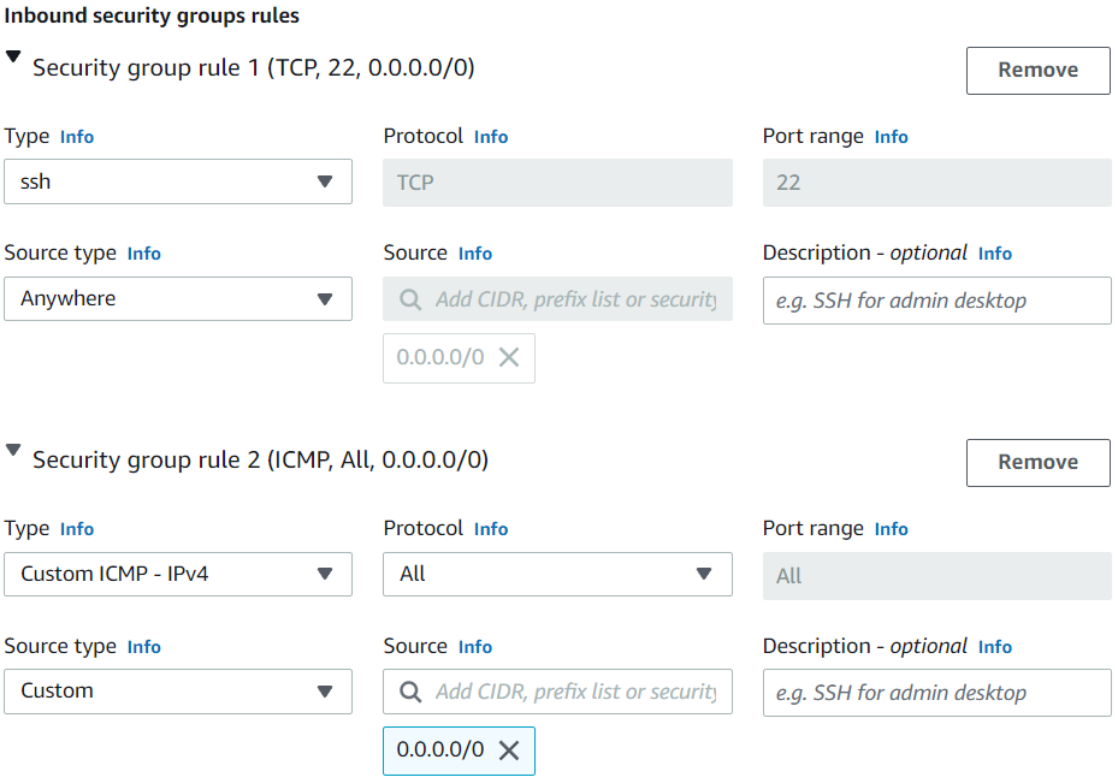

Ilustración 218 Creación de una instancia en AWS (6)

Una vez aceptada la configuración, esperamos un par de minutos hasta que la instancia esté activa. La dirección IP pública asignada es la 54.236.183.15 y la privada 10.1.0.184, acorde al rango de direccionamiento de la subred “subnet-us-east1” (10.1.0.0/16):

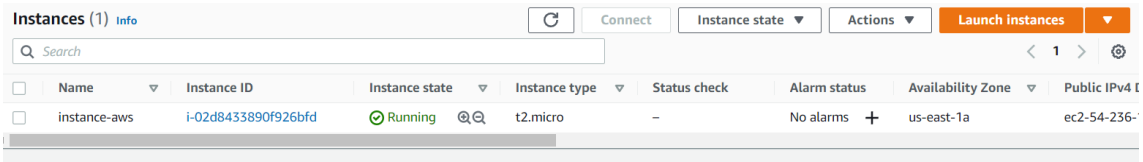

Ilustración 219 Creación de una instancia en AWS (7)

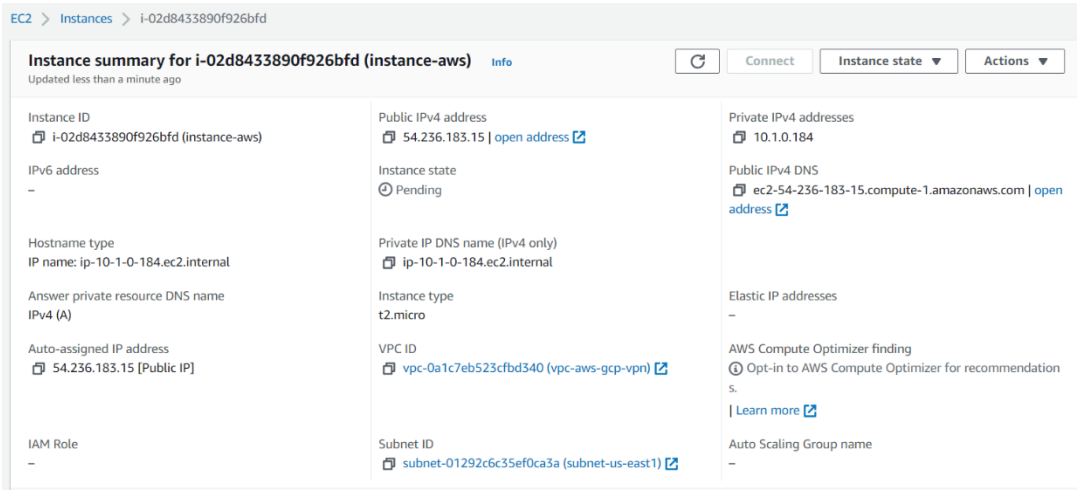

Ilustración 220 Creación de una instancia en AWS (8)

7.6.2. Lanzar instancia en Google Cloud

Ahora toca configurar la máquina del lado de GCP. Accedemos a "Compute Engine -> VM Instances" y seleccionamos "CREATE INSTANCE":

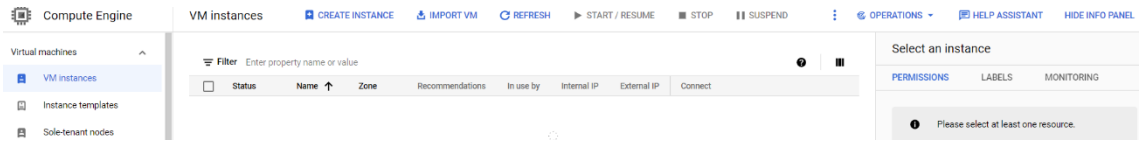

Ilustración 221 Creación de una instancia en GCP (1)

- Name: instance-1
- Region: us-east1 (debido a que es donde se encuentra nuestra VPC)
- Zone: us-east1-b (no es relevante)
- Series: E2
- Machine type: e2-micro

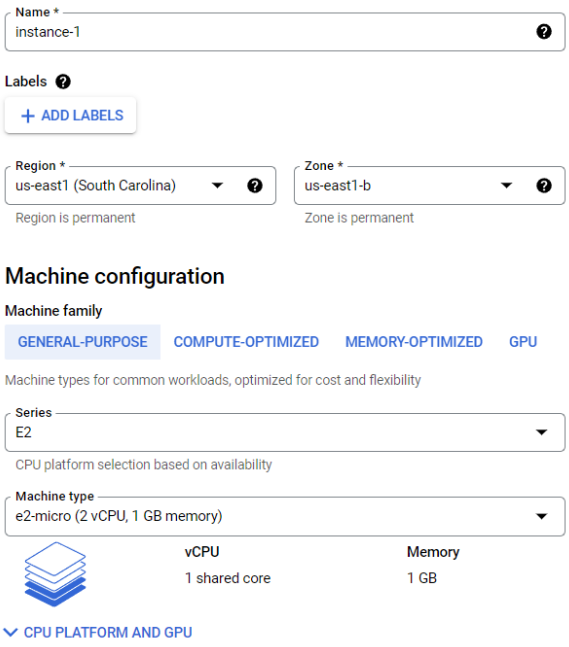

Ilustración 222 Creación de una instancia en GCP (2)

- Boot disk:
  - Operating system: Ubuntu
  - Version: Ubuntu 18.04 LTS

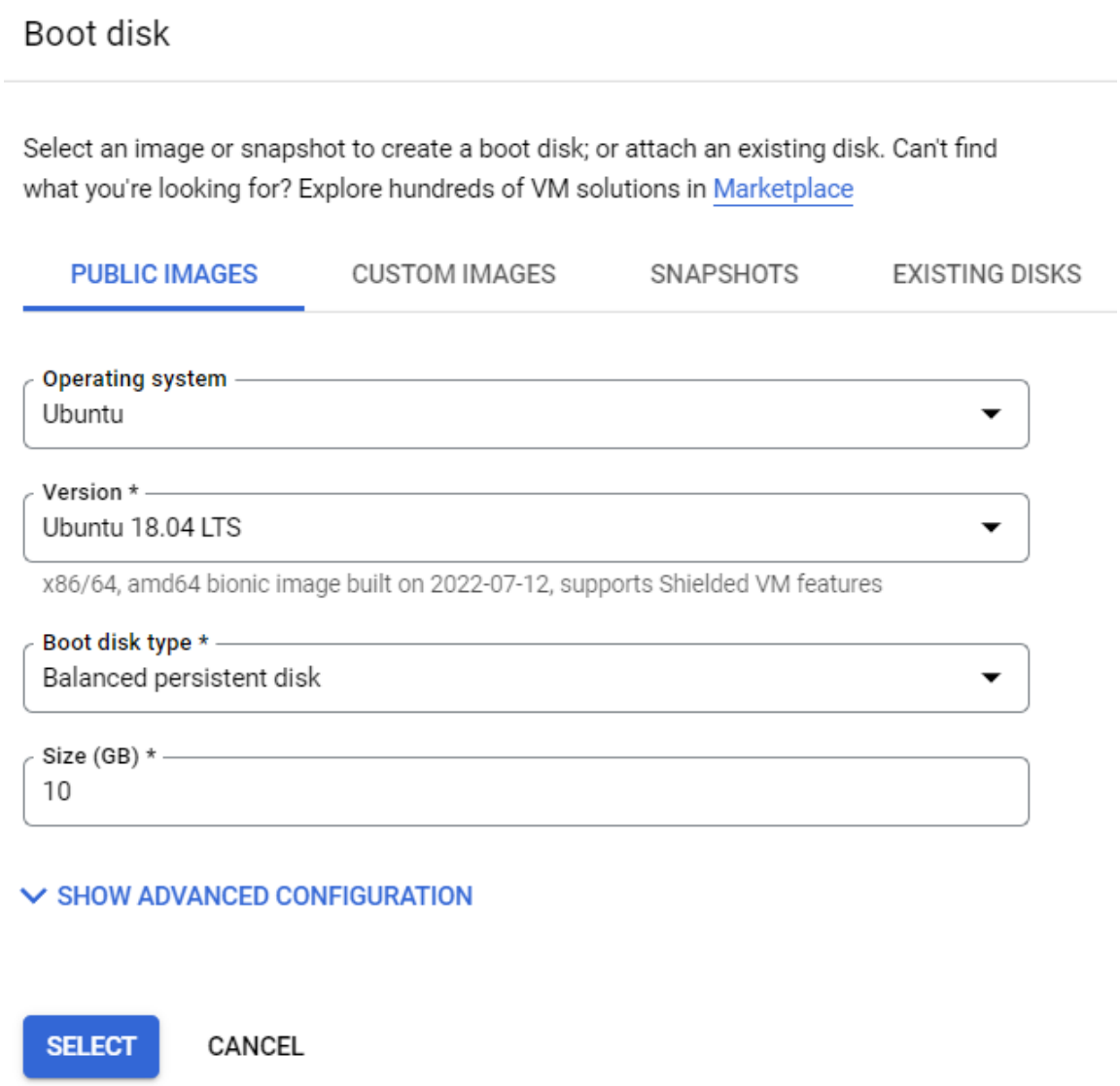

Ilustración 223 Creación de una instancia en GCP (3)

- Networking -> Network interfaces:
  - Network: vpc-aws-gcp-vpn

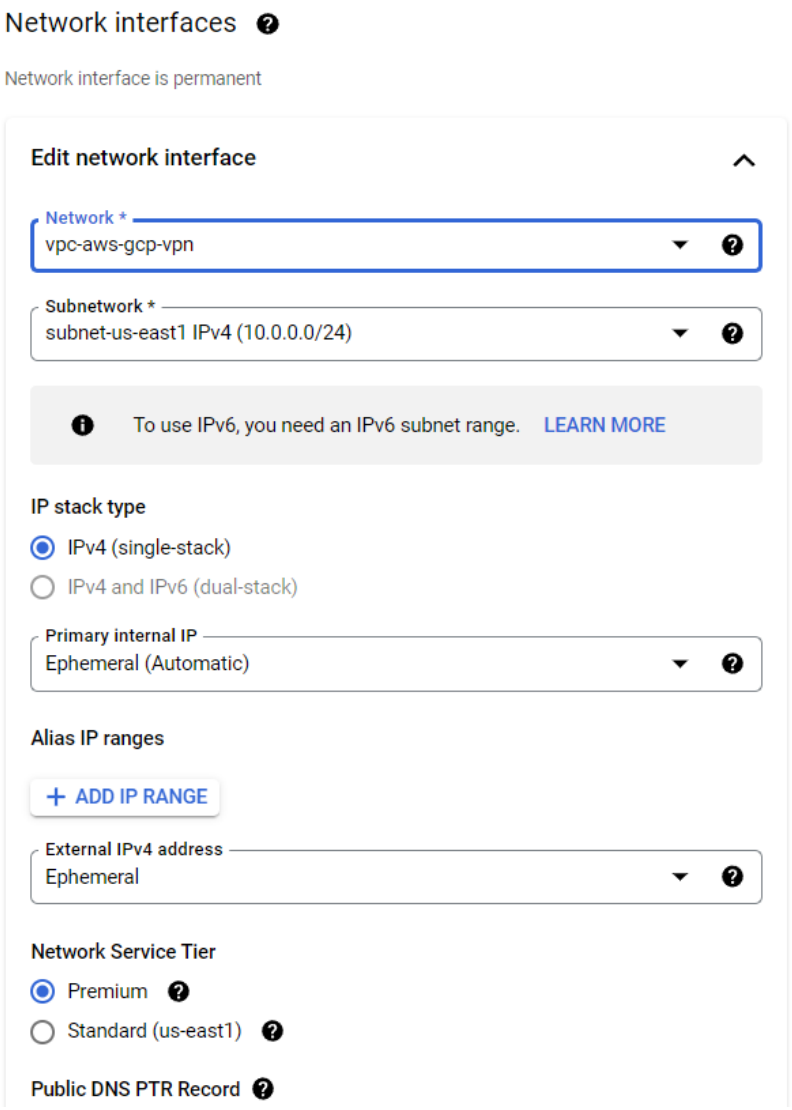

Ilustración 224 Creación de una instancia en GCP (4)

Pinchamos en “Create” y esperamos a que nuestra instancia esté lista. Una vez disponible, identificamos que su dirección IP pública es la 35.231.152.133 y la privada la 10.0.0.2 acorde al rango de direccionamiento de la subred correspondiente (10.0.0.0/24):

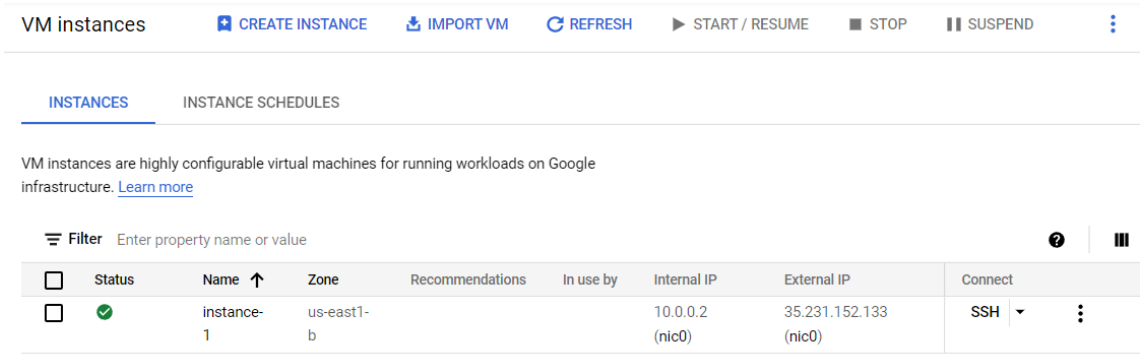

Ilustración 225 Creación de una instancia en GCP (5)

7.6.3. Test final de conectividad

Antes de continuar, echemos un vistazo al entorno final creado junto con las instancias desplegadas:

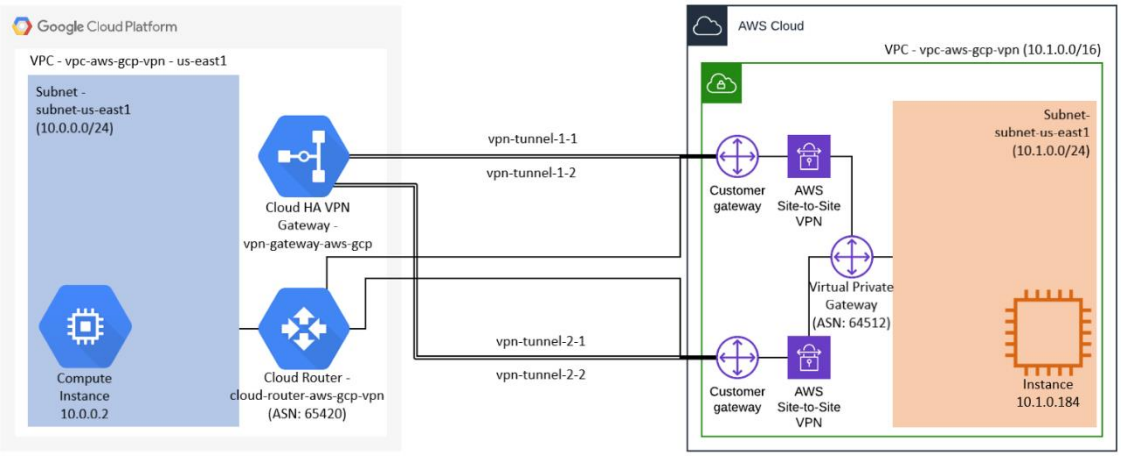

Ilustración 226 Esquema del entorno multi-cloud final

Si realizamos un ping desde nuestro sistema local a las direcciones 35.231.152.133 y 54.236.183.15 obtenemos respuesta. Se tratan de las direcciones IP públicas de cada máquina y las reglas de seguridad permiten responder a los mensajes ICMP de cualquier dirección. Sin embargo, como es de esperar, no obtendremos respuesta de las direcciones privadas ya que nuestro sistema local no tiene la información y configuración necesaria para alcanzar dichas direcciones:

```
silviu@LAPTOP-RKK3LQU3: ~
silviu@LAPTOP-RKK3LQU3:~$ ping 35.231.152.133
PING 35.231.152.133 (35.231.152.133) 56(84) bytes of data.
64 bytes from 35.231.152.133: icmp_seq=1 ttl=56 time=365 ms
64 bytes from 35.231.152.133: icmp_seq=2 ttl=56 time=228 ms
^C
--- 35.231.152.133 ping statistics ---
3 packets transmitted, 2 received, 33.3333% packet loss, time 2002ms
rtt min/avg/max/mdev = 227.566/296.124/364.683/68.558 ms
silviu@LAPTOP-RKK3LQU3:~$ ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1002ms

silviu@LAPTOP-RKK3LQU3:~$ ping 54.236.183.15
PING 54.236.183.15 (54.236.183.15) 56(84) bytes of data.
64 bytes from 54.236.183.15: icmp_seq=1 ttl=237 time=93.8 ms
64 bytes from 54.236.183.15: icmp_seq=2 ttl=237 time=153 ms
64 bytes from 54.236.183.15: icmp_seq=3 ttl=237 time=174 ms
^C
--- 54.236.183.15 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2010ms
rtt min/avg/max/mdev = 93.779/140.268/174.398/34.053 ms
silviu@LAPTOP-RKK3LQU3:~$ ping 10.1.0.184
PING 10.1.0.184 (10.1.0.184) 56(84) bytes of data.
^C
--- 10.1.0.184 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1004ms

silviu@LAPTOP-RKK3LQU3:~$
```

Ilustración 227 Test de conectividad de la multi-cloud (1)

En primer lugar, comprobaremos si la conexión VPN funciona del lado de Google Cloud. Clickamos en nuestra instancia GCP y seleccionamos “SSH” para conectarnos a la máquina desde “Google Cloud Console”:

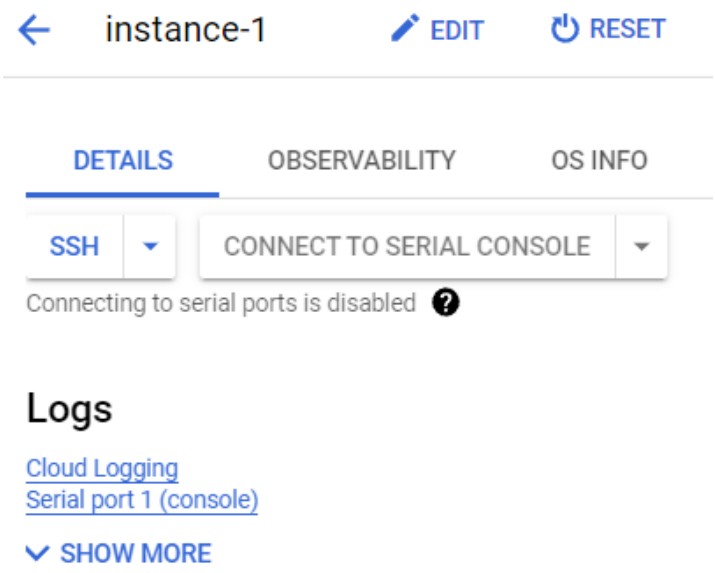

Ilustración 228 Test de conectividad de la multi-cloud (2)

Verificamos que desde “intance-1” (10.0.0.2) podemos realizar ping a la dirección IP privada (10.1.0.184) de la máquina ubicada en la VPC de Amazon, así como a su dirección pública:

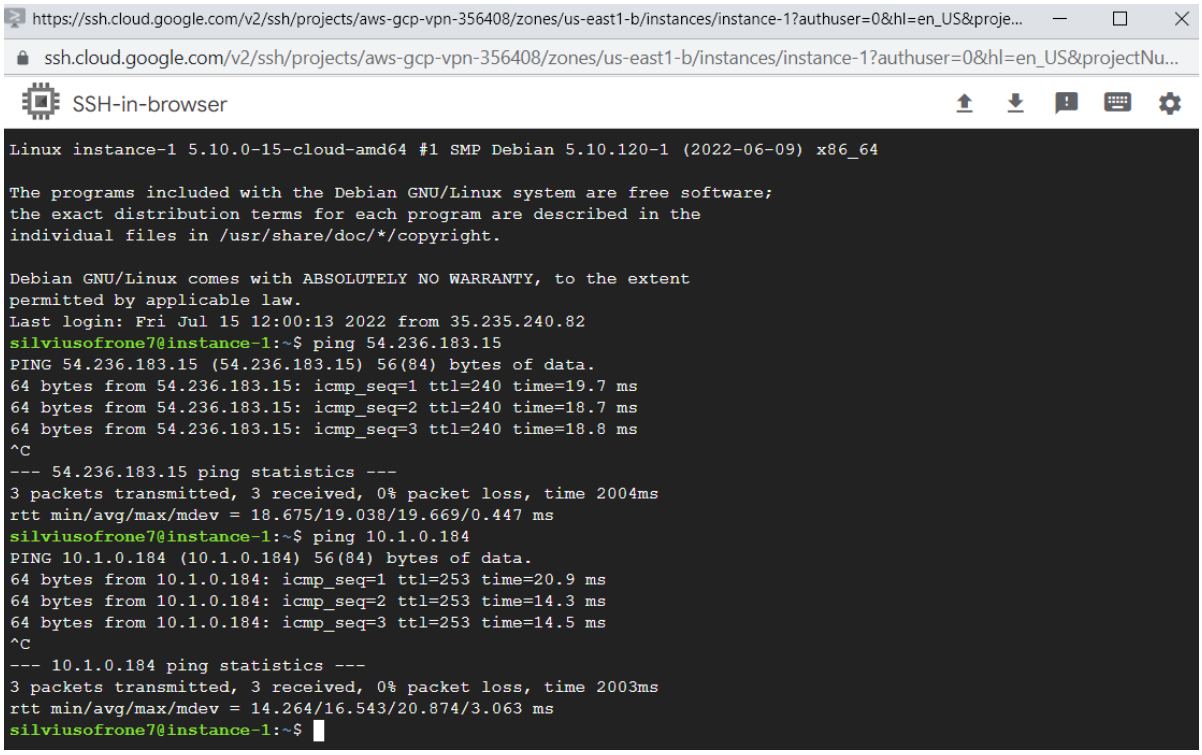

Ilustración 229 Test de conectividad de la multi-cloud (3)

Por último, realizamos la misma prueba del lado de AWS. Para conectarnos desde la consola de Amazon EC2 a “instance-aws” pinchamos en el nombre de la instancia y accedemos a “Connect to instance -> EC2 Instance Connect -> Connect”:

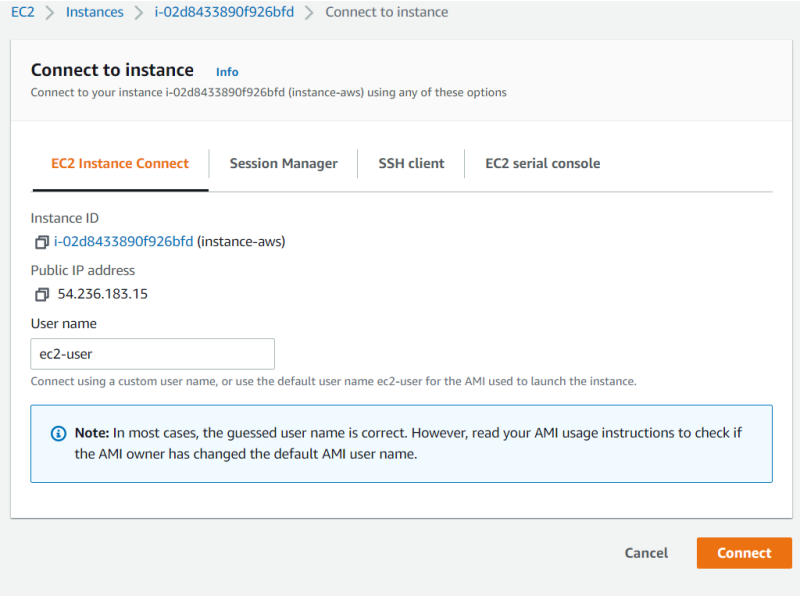

Ilustración 230 Test de conectividad de la multi-cloud (4)

Ahora podemos asegurar que la conexión VPN funciona en ambos sentidos. Desde la instancia ubicada en la VPC de Amazon (10.1.0.184) somos capaces de hacer ping a la dirección privada (10.0.0.2) perteneciente a la máquina de GCP:

Ilustración 231 Test de conectividad de la multi-cloud (5)

8. Conclusiones

A lo largo de este proyecto hemos profundizado en el trabajo con técnicas de despliegue OpenStack como DevStack y, en especial, OpenStack-Ansible. Los conocimientos adquiridos sobre este último nos dotan de la capacidad para configurar un entorno OpenStack privado moldeado a nuestras necesidades. Esto nos abre un amplio abanico de posibilidad a la hora de embarcarnos en la tarea de conformar nubes más competentes en términos de escala y funcionalidades. De ser así el caso, la presente memoria contiene la información necesaria para entender el funcionamiento de OSA, así como los puntos a configurar en base a las exigencias de cada entorno. En adición, también supone un buen punto de partida para el cometido de escalar OpenStack a una nube híbrida, presentando algunas de las principales opciones que tenemos a nuestra disposición (tanto si la idea es enfrentarnos a ello por nuestra propia cuenta o con la ayuda de las soluciones que hay en el mercado), a la vez que se ha intentado integrarlas con nuestro entorno. Y, aunque esta última parte ha resultado en una tarea compleja, hemos sabido adaptarnos a los problemas que han ido aconteciendo durante el proyecto, haciendo uso de la aplicación Juju, y de los recursos de Amazon Web Services y Google Cloud, para replantear los objetivos enunciados en un principio.

Por otro lado, si bien Juju ha resultado ser una herramienta muy potente para el rápido despliegue y gestión de aplicaciones en nuestra nube híbrida, se nos queda un poco corta en comparación a las funcionalidades añadidas de creación de redes e imágenes que nos ofrecía el sistema Omni. Aun así, debido a la gran variedad de entornos a los que tiene acceso, nos ha permitido interactuar con más de una nuble pública, siendo Google Cloud y AWS las elegidas.

La comunicación entre nuestra nube OpenStack y las públicas ha sido otro tema que se nos ha atragantado. Sin embargo, al igual que con otros problemas que han surgido durante este trabajo, se ha buscado un camino alternativo que nos permita avanzar y conformar una solución parecida a la comunicación entre una nube privada y una pública. Esto ha desembocado, aprovechando la virtualización, en la configuración de una multi-cloud donde la comunicación entre GCP y AWS sea posible, dándonos espacio a su vez para ahondar en el uso de los servicios *cloud* que proporcionan cada una.

Por último, como línea de trabajo a futuro, se propone el estudio y despliegue de OpenStack mediante una instalación manual, con el objetivo de cumplir los requisitos que exigía el sistema Omni para poder implementar el controlador de Neutron. Es decir, realizar una instalación sin el agente Neutron L3 y el driver ML2 configurados.

9. Presupuesto

A continuación, expondremos el tiempo empleado en desarrollar cada parte del proyecto y el coste de los medios empleados para tales fines. Comenzamos con el coste de los dispositivos que han sido utilizado para el despliegue de la nube OpenStack-Ansible:

- Dos ordenadores Intel Core i7-9700K, 32GB RAM, 8 vCPUs: 2x1500€
- Switch Netgear GS716T: 300€

El coste total asciende a 3300€.

Para calcular el número de horas invertidas en todo el proceso, separaremos las tareas realizadas en las siguientes fases:

- **Estudio sobre los métodos de despliegue de OpenStack:** fase inicial dedicada a la búsqueda del proceso de instalación más adecuado para nuestro cometido.
- **Estudio, configuración y despliegue de OpenStack-Ansible:** todo este proceso abarca tareas como la elección de un entorno apropiado para la instalación, el preparado de las máquinas y de la red sobre la que se va a realizar, la configuración de los archivos del despliegue y el propio proceso de instalación.
- **Operaciones con la nube OpenStack-Ansible:** dedicado a comprobaciones básicas del funcionamiento de OpenStack, toma de contacto con los diferentes servicios a nuestra disposición (creación y manejo de imágenes e instancias y sus respectivos grupos de seguridad, generar y proporcionar direcciones IP flotantes, etc) y tareas propias del conformado de una nube privada (creación de una red virtual desde la que la nube OpenStack pueda conectarse al exterior).
- **Estudio, configuración y despliegue de OpenStack-Omni:** tiempo empleado en el estudio sobre la integración del sistema Omni con la nube OSA, así como con el proceso de instalación DevStack y sus respectivos despliegues. Engloba tareas como la creación de las credenciales de la API de AWS, configuración del script **stack.sh** y solución de errores.
- **Estudio e implementación de la nube híbrida con Juju**: fase dedicada a la conexión entre la herramienta de gestión de aplicaciones Juju y las diferentes nubes elegidas para el conformado de una nube híbrida; OpenStack (la nube privada OSA), Google Cloud y Amazon Web Services. Abarca tareas comunes (agregado de las credenciales, unión de las nubes y despliegue de aplicaciones) como propias de cada entorno (creación de metadatos de imágenes OpenStack y de credenciales GCP).
- **Estudio e implementación de la multi-cloud AWS-GCP:** abarca toda la configuración y conformado de un entorno multi-cloud entre Amazon Web Services y Google Cloud. Creación de las VPCs que contienen las redes a conectar, configuración de routers y puertas de enlace de cada proveedor y conexiones mediante túneles VPN.
- **Elaboración de la memoria final.**

Estimación en horas de cada tarea:

Fase	Horas
Estudio sobre los métodos de despliegue de OpenStack	20
Estudio, configuración y despliegue de OpenStack-Ansible	87
Operaciones con la nube OpenStack-Ansible:	98
Estudio, configuración y despliegue de OpenStack-Omni	115

Estudio e implementación de la nube híbrida con Juju	105
Estudio e implementación de la multi-cloud AWS-GCP	60
Elaboración de la memoria final	95

Obtenemos un total de 580 horas de trabajo.

10. Bibliografía

[1] Equipo de OpenStack. “Introduction to OpenStack”. Docs.openstack.org; 06/2021 [fecha de consulta: 02/2022]. Disponible en: https://docs.openstack.org/security-guide/introduction/introduction-to-openstack.html

[2] Thomas. “The Benefits and Challenges of Building an OpenStack Based Cloud.”. Eurovps.com; 02/2021 [fecha de consulta: 02/2022]. Disponible en: https://www.eurovps.com/blog/openstack-cloud-benefits-challenges/

[3] Siddheshwar More. “6 OpenStack Deployment tools that are awesome for your Project.”. Opcito.com; 05/2016 [fecha de consulta: 02/2022]. Disponible en: https://www.opcito.com/blogs/6-openstack-deployment-tools-that-are-awesome-for-your-project-and-why

[4] Alok Shrivastwa. “Hybrid Cloud for Architects”. Packt Publishing; 2018.

[5] Vijay Dwivedi. “Ansible Tutorial: Deploy OpenStack with Ansible”. Platform9.com; 06/2016 [fecha de consulta: 02/2022]. Disponible en: https://platform9.com/blog/install-openstack-using-openstack-ansible/

[6] Equipo de OpenStack. “OPENSTACK DOCS: CONFIGURING THE NETWORK”. Docs.openstack.org; 07/2019 [fecha de consulta: 02/2022]. Disponible en: https://docs.openstack.org/project-deploy-guide/openstack-ansible/queens/targethosts-networkconfig.html

[7] Equipo de OpenStack. “Test environment example — openstack-ansible 25.1.0.dev68 documentation”. Docs.openstack.org; 05/2018 [fecha de consulta: 02/2022]. Disponible en: https://docs.openstack.org/openstack-ansible/latest/user/test/example.html

[8] Equipo de OpenStack. “Network architectures — openstack-ansible 25.1.0.dev68 documentation”. Docs.openstack.org; 02/2021 [fecha de consulta: 02/2022]. Disponible en: https://docs.openstack.org/openstack-ansible/latest/user/network-arch/example.html

[9]Equipo de OpenStack. “Configure the deployment — openstack-ansible 25.1.0.dev53 documentation”. Docs.openstack.org; 08/2021 [fecha de consulta: 02/2022]. Disponible en: https://docs.openstack.org/project-deploy-guide/openstack-ansible/latest/configure.html

[10] Equipo de OpenStack. “Run playbooks — openstack-ansible 25.1.0.dev53 documentation”. Docs.openstack.org; 08/2021 [fecha de consulta: 02/2022]. Disponible en: https://docs.openstack.org/project-deploy-guide/openstack-ansible/latest/run-playbooks.html

[11]Equipo de OpenStack. “Overview — openstack-ansible 25.1.0.dev53 documentation”. Docs.openstack.org; 08/2021 [fecha de consulta: 02/2022]. Disponible en: https://docs.openstack.org/project-deploy-guide/openstack-ansible/latest/overview.html

[12] Equipo de NETGEAR. “GS716T and GS724T Gigabit Smart Switches - Software Administration Manual”; 2012.

[13] Equipo de OpenStack. “Prepare the deployment host — openstack-ansible 25.1.0.dev53 documentation”. Docs.openstack.org; 08/2021 [fecha de consulta: 02/2022]. Disponible en:

https://docs.openstack.org/project-deploy-guide/openstack-ansible/latest/deploymenthost.html

[14] Equipo de OpenStack. “OpenStack Networking — Neutron 21.1.0.dev2 documentation”. Docs.openstack.org; 05/2021 [fecha de consulta: 02/2022]. Disponible en: https://docs.openstack.org/neutron/latest/admin/intro-os-networking.html

[15] Equipo de OpenStack. “Launch and manage instances — horizon 22.2.1.dev31 documentation”. Docs.openstack.org; 01/2018 [fecha de consulta: 02/2022]. Disponible en: https://docs.openstack.org/horizon/latest/user/launch-instances.html

[16] Equipo de OpenStack. “OpenStack Docs: Verify operation”. Docs.openstack.org; 08/2019 [fecha de consulta: 03/2022]. Disponible en: https://docs.openstack.org/mitaka/install-guide-obs/glance-verify.html

[17] Equipo de OpenStack. “OpenStack Docs: Manage flavors”. Docs.openstack.org; 04/2021 [fecha de consulta: 03/2022]. Disponible en: https://docs.openstack.org/nova/pike/admin/flavors2.html

[18] Equipo de OpenStack. “OpenStack Docs: Launch an instance”. Docs.openstack.org; 08/2019 [fecha de consulta: 03/2022]. Disponible en: https://docs.openstack.org/newton/install-guide-ubuntu/launch-instance.html#launch-instance-networks

[19] Equipo de OpenStack. “OpenStack Docs: Provider network”. Docs.openstack.org; 08/2019 [fecha de consulta: 03/2022]. Disponible en: https://docs.openstack.org/newton/install-guide-ubuntu/launch-instance-networks-provider.html

[20] Equipo de OpenStack. “OpenStack Docs: Self-service network”. Docs.openstack.org; 08/2019 [fecha de consulta: 03/2022]. Disponible en: https://docs.openstack.org/newton/install-guide-ubuntu/launch-instance-networks-selfservice.html

[21] Damian Igbe. “Identifying and Troubleshooting Neutron Namespaces”. Mirantis.com; 11/2013 [fecha de consulta: 03/2022]. Disponible en: https://www.mirantis.com/blog/identifying-and-troubleshooting-neutron-namespaces/

[22] Equipo de RedHat. “Chapter 8. Troubleshoot Provider Networks Red Hat OpenStack Platform 10”. Redhat.com; [fecha de consulta: 03/2022]. Disponible en: https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/10/html/networking_guide/sec-neutron-troubleshooting

[23] Amrish Kapoor. OpenStack Omni, the Open-Source Alternative to VMware and AWS for Hybrid Cloud. Plataform9.com; 01/2018 [fecha de consulta: 04/2022]. Disponible en: https://platform9.com/blog/openstack-omni-the-open-source-alternative-to-vmware-aws-for-hybrid-cloud/

[24] Equipo de DevStack. “DevStack — DevStack documentation”. Docs.openstack.org; 08/2019 [fecha de consulta: 04/2022]. Disponible en: https://docs.openstack.org/devstack/latest/

[25] Equipo de DevStack. “Configuration — DevStack documentation”. Docs.openstack.org; 02/2020 [fecha de consulta: 04/2022]. Disponible en: https://docs.openstack.org/devstack/latest/configuration.html

[26] Canonical. “Juju (software)”. Wikipedia.org; 02/2022 [fecha de consulta: 05/2022]. Disponible en: https://es.wikipedia.org/wiki/Juju_(software)

[27] Canonical. “Juju | Controller”. Juju.is; 08/2021 [fecha de consulta: 05/2022]. Disponible en: https://juju.is/docs/olm/controllers

[28] Sean Shuping. “Hybrid Cloud with MAAS and JuJu”. Devzero.co.za; 08/2018 [fecha de consulta: 05/2022]. Disponible en: https://devzero.co.za/2018/08/23/hybrid-cloud-with-maas-and-juju/

[29] Canonical. “Juju | How to use OpenStack with Juju”. Juju.is; 04/2021 [fecha de consulta: 05/2022]. Disponible en: https://juju.is/docs/olm/openstack

[30] Canonical. “Juju | How to add a cloud credential to the Juju client”. Juju.is; 08/2021 [fecha de consulta: 05/2022]. Disponible en: https://juju.is/docs/olm/add-credentials

[31] Canonical. “Juju | How to configure your OpenStack cloud image metadata”. Juju.is; 01/2022 [fecha de consulta: 05/2022]. Disponible en: https://juju.is/docs/olm/cloud-image-metadata

[32] Canonical. “Juju | How to use Amazon AWS with Juju”. Juju.is; 07/2021 [fecha de consulta: 06/2022]. Disponible en: https://juju.is/docs/olm/amazon-aws

[33] Canonical. “Juju | How to use Google GCE with Juju”. Juju.is; 01/2022 [fecha de consulta: 06/2022]. Disponible en: https://juju.is/docs/olm/google-gce

[34] Google Cloud. “Service accounts - IAM Documentation”. Cloud.google.com. [fecha de consulta: 06/2022]. Disponible en: https://cloud.google.com/iam/docs/service-accounts

[35] Claudio Gomboli. “JAAS Dashboard, the new Juju GUI”. Discourse.charmhub.io; 04/2020 [fecha de consulta: 07/2022]. Disponible en: https://discourse.charmhub.io/t/jaas-dashboard-the-new-juju-gui/2978

[36] Canonical. “Juju | How to install Juju”. Juju.is; 01/2022 [fecha de consulta: 07/2022]. Disponible en: https://juju.is/docs/olm/installing-juju

[37] Google Cloud. “Build HA VPN connections between Google Cloud and AWS”. Cloud.google.com; 05/2021 [fecha de consulta: 07/2022]. Disponible en: https://cloud.google.com/architecture/build-ha-vpn-connections-google-cloud-aws

[38] AWS. “¿Qué es AWS Site-to-Site VPN?”. Doc.aws.amazon.com [fecha de consulta: 07/2022]. Disponible en: https://docs.aws.amazon.com/es_es/vpn/latest/s2svpn/VPC_VPN.html

[39] Tony Leung. “Connecting an AWS and GCP VPC using an IPSec VPN Tunnel with BGP”. Medium.com; 07/2020 [fecha de consulta: 07/2022]. Disponible en: https://tinyurl.com/connecting-an-aws-and-gcp-vpc

11. Anexos

11.1. Fichero /etc/network/interfaces del host de control

```
# interfaces(5) file used by ifup(8) and ifdown(8)

# Include files from /etc/network/interfaces.d:

source-directory /etc/network/interfaces.d


auto eth0

iface eth0 inet manual


auto eth1

iface eth1 inet static

      address 172.29.21.201

      netmask 255.255.252.0

      gateway 172.29.20.1

      dns-nameservers 8.8.8.8 1.1.1.1


auto eth2

iface eth2 inet manual


auto eth3

iface eth3 inet manual


# Physical interface

auto eth4

iface eth4 inet manual


# Container/Host management VLAN interface

auto eth4.10

iface eth4.10 inet manual

   vlan-raw-device eth4
```

```
# OpenStack Networking VXLAN (tunnel/overlay) VLAN interface
auto eth4.30
iface eth4.30 inet manual
    vlan-raw-device eth4

# Container/Host management bridge
auto br-mgmt
iface br-mgmt inet static
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    bridge_ports eth4.10
    address 10.0.123.11
    netmask 255.255.252.0

# Bind the External VIP
auto br-mgmt:0
iface br-mgmt:0 inet static
    address 10.0.123.10
    netmask 255.255.252.0

# OpenStack Networking VXLAN (tunnel/overlay) bridge
#
# The COMPUTE, NETWORK and INFRA nodes must have an IP address
# on this bridge.

auto br-vxlan
iface br-vxlan inet static
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
```

```
    bridge_ports eth4.30
    address 10.0.125.11
    netmask 255.255.252.0

# OpenStack Networking VLAN bridge
auto br-vlan
iface br-vlan inet manual
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    # Create veth pair, do not abort if already exists
    pre-up ip link add br-vlan-veth type veth peer name eth12 || true
# Set both ends UP
    pre-up ip link set br-vlan-veth up
    pre-up ip link set eth12 up
# Delete veth pair on DOWN
    post-down ip link del br-vlan-veth || true
    bridge_ports eth4 br-vlan-veth

source /etc/network/interfaces.d/*.cfg
```

11.2. Fichero /etc/network/interfaces del host de cómputo

```
#iface eth4 inet manual

auto eth1
iface eth1 inet static
        address 172.29.21.200
        netmask 255.255.252.0
        gateway 172.29.20.1
        dns-nameservers 8.8.8.8 1.1.1.1
```

```
auto eth4
iface eth4 inet manual

auto eth4.10
iface eth4.10 inet manual
   vlan-raw-device eth4

auto eth4.30
iface eth4.30 inet manual
   vlan-raw-device eth4

auto br-mgmt
iface br-mgmt inet static
   bridge_stp off
   bridge_waitport 0
   bridge_fd 0
   bridge_ports eth4.10
   address 10.0.123.12
   netmask 255.255.252.0

auto br-vxlan
iface br-vxlan inet static
   bridge_stp off
   bridge_waitport 0
   bridge_fd 0
   bridge_ports eth4.30
   address 10.0.125.12
   netmask 255.255.252.0

# compute1 Network VLAN bridge
auto br-vlan
```

```
iface br-vlan inet manual
   bridge_stp off
   bridge_waitport 0
   bridge_fd 0
# Create veth pair, do not abort if already exists
   pre-up ip link add br-vlan-veth type veth peer name eth12 || true
# Set both ends UP
   pre-up ip link set br-vlan-veth up
   pre-up ip link set eth12 up
# Delete veth pair on DOWN
   post-down ip link del br-vlan-veth || true
   bridge_ports eth4 br-vlan-veth
```

11.3. Archivo openstack_user_config.yml

```
---
cidr_networks:
 container: 10.0.120.0/22
 tunnel: 10.0.124.0/22

used_ips:
 - "10.0.123.10,10.0.123.12"
 - "10.0.125.11,10.0.125.12"

global_overrides:
 # The internal and external VIP should be different IPs, however they
 # do not need to be on separate networks.
 external_lb_vip_address: 10.0.123.10
 internal_lb_vip_address: 10.0.123.11
 management_bridge: "br-mgmt"
 provider_networks:
  - network:
      container_bridge: "br-mgmt"
```

```
    container_type: "veth"
    container_interface: "eth1"
    ip_from_q: "container"
    type: "raw"
    group_binds:
      - all_containers
      - hosts
    is_container_address: true
  - network:
    container_bridge: "br-vxlan"
    container_type: "veth"
    container_interface: "eth10"
    ip_from_q: "tunnel"
    type: "vxlan"
    range: "1:1000"
    net_name: "vxlan"
    group_binds:
      - neutron_linuxbridge_agent
  - network:
    container_bridge: "br-vlan"
    container_type: "veth"
    container_interface: "eth12"
    host_bind_override: "eth12"
    type: "flat"
    net_name: "flat"
    group_binds:
      - neutron_linuxbridge_agent
  - network:
    container_bridge: "br-vlan"
    container_type: "veth"
    container_interface: "eth11"
```

```
      type: "vlan"
      range: "101:200,301:400"
      net_name: "vlan"
      group_binds:
        - neutron_linuxbridge_agent

###
### Infrastructure
###

# galera, memcache, rabbitmq, utility
shared-infra_hosts:
  infra1:
    ip: 10.0.123.11

# repository (apt cache, python packages, etc)
repo-infra_hosts:
  infra1:
    ip: 10.0.123.11

# load balancer
haproxy_hosts:
  infra1:
    ip: 10.0.123.11

###
### OpenStack
###

# keystone
identity_hosts:
```

```
  infra1:
    ip: 10.0.123.11

# cinder api services
storage-infra_hosts:
  infra1:
    ip: 10.0.123.11

# glance
image_hosts:
  infra1:
    ip: 10.0.123.11

# placement
placement-infra_hosts:
  infra1:
    ip: 10.0.123.11

# nova api, conductor, etc services
compute-infra_hosts:
  infra1:
    ip: 10.0.123.11

# heat
orchestration_hosts:
  infra1:
    ip: 10.0.123.11

# horizon
dashboard_hosts:
  infra1:
```

```
  ip: 10.0.123.11


# neutron server, agents (L3, etc)
network_hosts:
 infra1:
  ip: 10.0.123.11


# nova hypervisors
compute_hosts:
 compute1:
  ip: 10.0.123.12
```

11.4. Archivo user_variables.yml

```
###
### This file contains commonly used overrides for convenience. Please inspect
### the defaults for each role to find additional override options.
###


## Debug and Verbose options.
debug: true


## Installation method for OpenStack services
# Default option (source) is to install the OpenStack services using PIP
# packages. An alternative method (distro) is to use the distribution cloud
# repositories to install OpenStack using distribution packages
install_method: source
```

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR