Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Diseño y gestión avanzada de nubes de computación

Autor: Domínguez Fernández, Ángel

Tutor/es: Arco Rodríguez, José Manuel

2022

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería en Tecnologías de Telecomunicación

Trabajo de Fin de Grado

Diseño y gestión avanzada de nubes de computación

Autor: Domínguez Fernández, Ángel

Tutor: Arco Rodríguez, José Manuel

TRIBUNAL:

Presidente: Carral Pelayo, Juan Antonio

Vocal 1º: García Reinoso, Jaime José

Vocal 2º: Arco Rodríguez, José Manuel

FECHA: Julio, 2022

Índice

Índice de figuras

1. Resumen

Si se observa el nombre del trabajo de fin de grado y se analiza detalladamente, se puede realizar un resumen perfecto de lo que es este trabajo:

"***DISEÑO Y GESTIÓN AVANZADA DE NUBES DE COMPUTACIÓN***"

Diseño, se ve como se puede automatizar el despliegue de las aplicaciones que nos van a servir para el entorno de virtualización y elegir dónde van estar alojadas dichas aplicaciones.

Gestión, se ve como se puede explotar el ecosistema que se diseña para sacarle el mayor rendimiento posible.

Avanzado, se parte de un escenario inicial antiguo, al cual se le actualizarán las versiones y se le corregirán algunos puntos de fallo.

Nubes de computación, este es el punto sobre el cual gira el proyecto que se aborda: El paradigma del *Cloud Computing*

Con esto se abordará un proyecto que pone en contexto cómo funciona una nube de computación y cómo sacar el máximo rendimiento de la misma.

2. Abstract

If we observe the title of the thesis project and analyze it in detail, we can do a perfect summary about the purpose of this work.

"***DESIGN AND ADVANCE MANAGEMENT OF COMPUTING CLOUDS***"

Design because we will see how to automatize the array of applications to be used in the virtualization environment, as well as where these applications will be hosted.

Management because we will see how to exploit the designed ecosystem to maximize its performance.

Advanced because the starting point will be an initial old platform which will get updated through new versions, and which bugs will be fixed.

Computing clouds is the key point around which the project develops: "The cloud computing paradigm."

With this in mind, we will develop a project that will try to contextualize the functioning of a computing cloud and how to maximize its performance.

3. Resumen extendido

En este trabajo de fin de grado, se propone una guía documentada sobre la plataforma de *Software* libre de virtualización, *OpenStack*. Aunque la tecnología *Cloud* tenga un tiempo de vida relativamente largo (entre 15 y 20 años), se ha requerido de la realización de un sistema que introduzca las principales características de los sistemas *Cloud*, así como documentación de los principales agentes y abstracciones que entran en juego en un servicio de *Cloud Computing*.

Cuando se menciona la tecnología *cloud*, se puede pensar en los tres grandes proveedores de este tipo de servicio: *Amazon, Microsoft Azure o Google Cloud*. Estas empresas tienen a su vez, sus propios subsistemas que brindan los servicios que se conocen sobre virtualización en la nube. Este tipo de servicios son máquinas virtuales, virtualización de red, contenedores o almacenamiento entre otros...

Con este proyecto se pretende dar a conocer uno de los subsistemas de código abierto que permite ofrecer las principales características de cualquier sistema *Cloud*. Se desplegarán las aplicaciones de *OpenStack* en varios servidores y estos servirán como *IaaS* o infraestructura como servicio.

Además, este despliegue constará de herramientas *Software* de gestión de máquinas de cómputo como *MAAS* y herramientas de administración de aplicaciones como *Juju*. Entonces, la lógica de *Openstack* estará orquestada por estas dos entidades descritas anteriormente.

Para finalizar, se comprobará el buen funcionamiento del sistema con algunas pruebas de concepto. Las pruebas consistirán en la creación de un escenario para dar servicio a usuarios, escalabilidad de la nube o monitorización de los recursos de los que se disponen entre otros.

4. Introducción

Se aborda este proyecto con un escenario inicial, en el cual se presenta una nube de computación que consta de 6 servidores. En estos servidores, se despliegan los componentes *Software* de gestión de nubes *Openstack*.

Como objetivo de este trabajo, se va a estudiar, cómo una herramienta de software libre, como es *Openstack*, puede ser explotada para que se desplieguen instancias en ella y además se pueda automatizar el despliegue de todas sus aplicaciones.

Para la consecución de los objetivos marcados en la realización del proyecto, es indispensable tener clara la estructura de la red que se va a utilizar. La arquitectura de la red que se va a utilizar en primera instancia se expone en la siguiente ilustración:

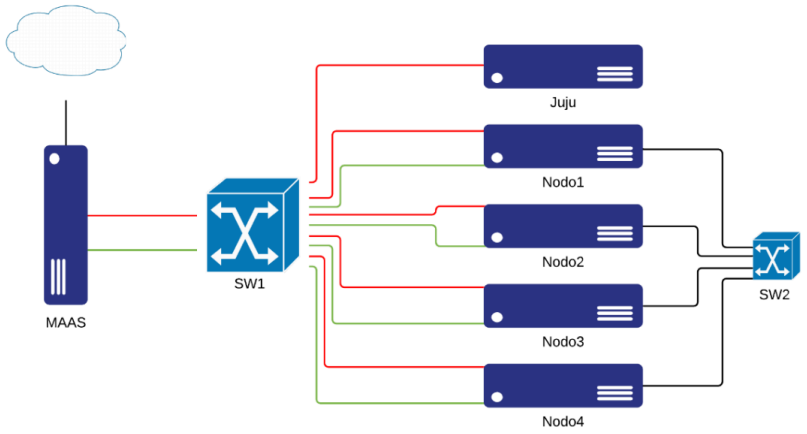


Figura 1: Arquitectura de red [9]

Como se puede observar en la figura anterior [1], se dispone de un conmutador *Netgear* que realiza la conexión entre el servidor que va a orquestar todo el escenario (*MaaS*) y los demás nodos. Los demás servidores a la derecha del conmutador son el controlador *JuJu*, que orquestará todas las aplicaciones de la nube, y los nodos de cómputo donde se alojarán las aplicaciones de *Openstack* necesarias para el funcionamiento de la nube.

Para la orquestación de todo el sistema se utiliza el *Software* de *MaaS*. Este *software* nos proporciona información en un *Dashboard* sobre los recursos *hardware* de los que dispone la nube y el estado de los mismos. También permite configurar los aspectos de la red mediante su *API* web.

Por otro lado, complementando al servidor *MaaS*, se tiene el agente *JuJu*. Este controlador irá instalado en otro nodo de cómputo presente en la red. El *software* de *JuJu*, se encarga de controlar las configuraciones de las aplicaciones *Openstack* y el despliegue de las mismas en los nodos cómputo.

Openstack es una herramienta de software libre que se puede desplegar mediante diferentes métodos. En concreto en este TFG tras un estudio de los diferentes métodos que se presentan, se decide usar: *OpenStack Charms Deployment*. Este método, se basa en la creación de contenedores en los nodos de cómputo, para desplegar todo el conjunto de operaciones que hacen posible el funcionamiento de la nube. Estas operaciones, estarán siempre controladas por el agente *JuJu*. Este despliegue da opción a elegir el número de nodos de cómputo que se quieran utilizar. El caso común que viene en la documentación de *OpenStack* y óptimo para un buen uso del sistema es con cuatro nodos de cómputo, aunque también se han hecho pruebas que se presentarán en siguientes apartados, con diferente número de nodos de cómputo.

Cuando las aplicaciones estén funcionando se utilizará el *Dashboard de Openstack* para lanzar un escenario donde varios usuarios de la nube puedan lanzar instancias. Estas instancias deben ser accesibles por *SSH* desde cualquier punto de internet o al menos desde la *VPN* de la universidad. Se dotará a dichas instancias de salida a internet para que se puedan descargar recursos y se instalará un servidor de Apache, para mandarle peticiones y así monitorizar la red.

Se realizarán pruebas de escalabilidad de los nodos de cómputo. Estas pruebas serán orquestadas por el *Software* de *MaaS*. Añadir recursos a la nube es primordial porque esta tecnología está en auge y cada vez

más gente utiliza este tipo de servicios. También es posible que se quieran eliminar nodos en la nube, se estudiará un método relacionado con el agente *JuJu* para resolver esto.

Se pondrá en contexto un escenario en el que se reinician todos los servidores de la nube y se procede a recuperar todo el sistema. Además se protegerá a los servidores de posibles cortes de luz.

Para finalizar, una vez el escenario esté disponible para pruebas, se estresará a los servidores y se hará una monitorización de dicha prueba con alguna aplicación como lo es *Grafana* o *Prometheus*.

En las conclusiones, se hará un estudio detallado de los posibles puntos de fallo para un posible trabajo futuro.

5. El paradigma del *Cloud Computing*

Como ya se sabe, el mundo del *software* está en constante evolución. La computación en la nube (*Cloud computing*) es la utilización de dicho software para ofrecer servicios a los usuarios mediante la red a gran escala.

La computación en la nube ofrece a los usuarios una variedad de recursos que son seguros (hay empresas que se dedican específicamente a la seguridad en la nube), tienen un buen mantenimiento y son de fácil acceso. Se proporciona a los usuarios unos servidores donde alojar sus datos y hacer pública información relevante para la población que consume de internet. Solo es necesario una conexión a internet para consumir de este recurso.

Si se piensa en el entorno de una empresa, puede ser que dicha organización requiera de un *software* especifico para desarrollar el trabajo diario. Los servidores que se utilizan para desplegar este *software*, podrían tenerlos físicamente o realizar este trabajo en servidores en la nube y pagar por los recursos. En este caso, habría que poner en una balanza si es más barato contratar técnicos para el mantenimiento de estos servidores o delegar todo esto en unos servidores que puedes diseñar a tu medida de manera virtual.

Además, las empresas que ofrecen este tipo de servicios, proporcionan una seguridad que en muchos caso supera a la seguridad que tendrían unos servidores locales.

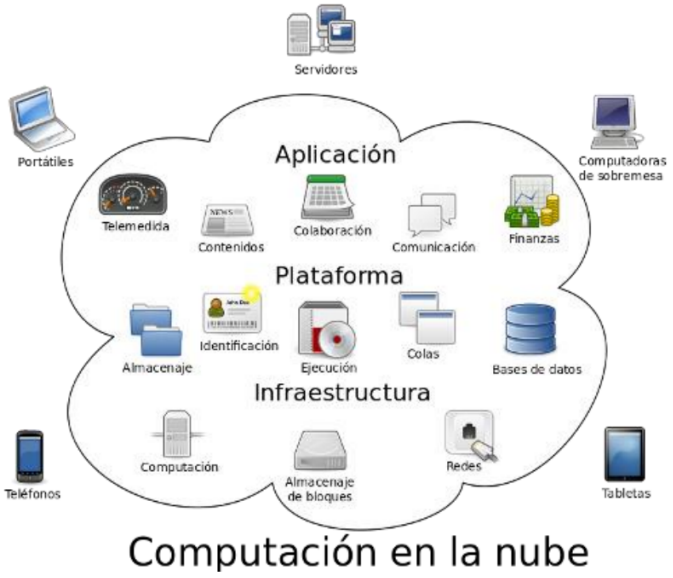


Figura 2: Cloud Computing

5.1. Caracteristicas del *Cloud Computing*

Las principales características del *Cloud Computing* son:

- Recursos bajos demanda: No es necesario un técnico que realice la provisión del servidor, el propio usuario cuenta con la capacidad de computación necesaria.
- Software multiplataforma: Basta una conexión a internet para tener acceso a tu servidor en la nube.
- Flexibilidad rápida: El usuario puede cambiar los recursos hardware a medida cuando lo desee.
- Servicio medido: Los recursos son monitorizados de manera constante.

5.2. Modelos para contratar el servicio

Algunos de los modelos de servicio que ofrece el *Cloud Computing*[10] son los siguientes:

- *Software as a service (SaaS)*: Los cliente que utilizan este servicio, contratan el uso de las aplicaciones que corren en la nube en cuestión. Un ejemplo de esto, es la empresa *SalesForce*. Las aplicaciones que estas empresas ofrecen están controladas por técnicos de la misma. Una de las ventajas de usar estas aplicaciones, es que todos los usuarios usan la misma versión de las.
- *Platform as a service (PaaS)*: Los clientes que consumen este tipo de modelo de servicio, se aprovechan de la plataforma en la que corren ciertas aplicaciones. Un ejemplo de empresa que usa este modelo es *Google App Engine*. Los clientes como en *SaaS* tienen el control de las aplicaciones que consumen. El mantenimiento del servidor recae sobre los técnicos de la empresa que ofrece el servicio.
- *Infrasture as a service (IaaS)*: Los cliente que usan este tipo de servicio demandan recursos *Hardware* como una CPU, capacidad de almacenamiento en disco o algún recurso en línea. Estos recursos son accesibles gracias a la virtualización y son accesibles mediante internet. Los clientes son responsables y tienen el control de los recursos que utilizan.
- *Storage as a service (STaaS)*: Este modelo de negocio es usado por los clientes que demandan espacio para almacenar datos relevantes para ellos. Este servicio es interesante debido a que los propios datos es mejor tenerlos en un servicio como este, que en servidores propios del cliente. El único inconveniente de este modelo, es que los datos son accesibles por medio de internet, es decir, es necesario una conexión a la red para hacer accesibles los datos.
- *Security as a service (SECaaS)*: Este modelo consiste en integrar la seguridad en la infraestructura que presente el cliente. Estos servicios suelen incluir autenticación, servicio de anti-virus, protección frente a ataques a los servidores, entre otros...
- *Data as a Service (DaaS)*: Este modelo ofrece a los clientes tener sus datos siempre accesibles desde cualquier lugar del mundo. Se está utilizando cada vez más en organizaciones como la ONU, por ejemplo.
- *Test enviroment as a Service(TEaaS)*: Los clientes que utilizan un *Software* específico realizan pruebas de él, mediante este tipo de herramientas. Normalmente este tipo de herramientas son accesibles mediante cualquier navegador de internet ya que están corriendo en una nube.
- *Backend as a Service (BaaS)*: También conocido como *Mobile Backend as a Service* es un modelo de nube en el cual los desarrolladores se apoyan para realizar páginas web y aplicaciones para móviles. Este servicio, une el almacenamiento de tipo *Backend* de la nube con la aplicación en cuestión. Este servicio es relativamente nuevo en *Cloud Computing* y tiene un valor de mercado de 216.5 millones de dólares en el año 2012.

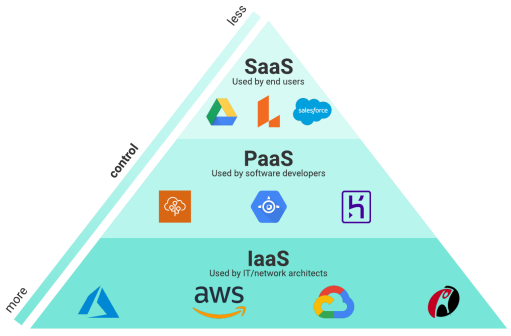


Figura 3: Esquema principales proveedores de *Cloud Computing*

5.3. Tipos de nubes

Las nubes de computación [8] pueden ser clasificadas según quién las maneje y a quién pertenezcan. Una clasificación común es la siguiente:

5.3.1. Nube pública

Las nubes públicas son las más comunes que se pueden encontrar. Este tipo de nubes son accesibles para el público que paga dicho servicio. Los clientes acceden al servicio mediante una conexión de internet. Algunas empresas que ofrecen este servicio son *Amazon, Microsoft o Google* y éstas poseen centros de datos donde alojan los servidores que proveen el servicio.

5.3.2. Nube privada

Este tipo de nubes son utilizadas por empresas que quieren una nube que solo sea accesible por ellos y por lo tanto los servidores (*hardware*) de cómputo están situados en la propia entidad. Un claro ejemplo de esto es la nube que se plantea en este TFG, esta nube solo será accesible a través de la VPN de la universidad. Esta modalidad suele ser la más segura al no estar abierta al público.

5.3.3. Nube híbrida

Este tipo de nubes son una mezcla de nubes públicas y privadas. Una nube privada puede conseguir un mantenimiento de la misma haciendo públicos algunos recursos de ella. De esta manera, la recuperación ante errores o caídas del escenario en cuestión se podrían subsanar de manera más ágil.

5.3.4. Nube comunitaria

Este tipo de nubes suelen ser compartidas y mantenidas por varias empresas a la vez. En estos casos, varias empresas que desean los mismos objetivos comparten la infraestructura. Otra forma de nube comunitaria, es crear un escenario virtual con instancias de la propia nube que no están siendo utilizadas.

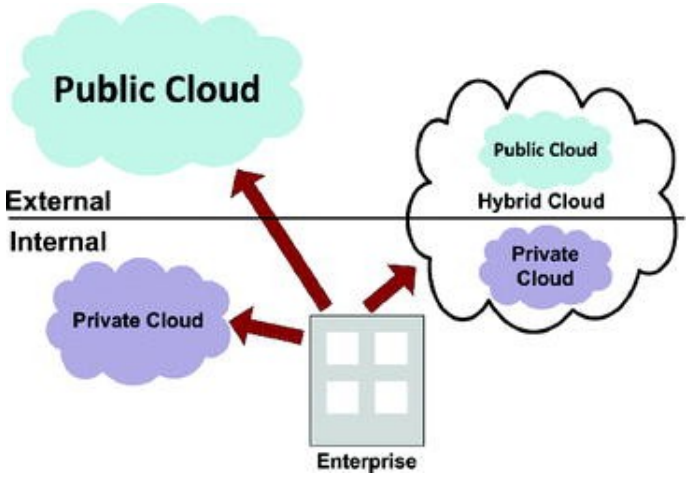


Figura 4: Tipo de nubes de computación

5.4. Ventajas del *Cloud Computing*

- Acceso a la información y servicios desde cualquier lugar.
- El servicio esta siempre disponible mediante una aplicación web.
- Los recursos están disponibles desde cualquier dispositivo tales como PDAs, teléfonos móviles, ordenadores, portátiles, etc...

- No se saturan los recursos de los que dispone el cliente físicamente ya que los servidores están en grandes *data-centers*, solo se necesita una conexión a internet
- Capacidad de procesamiento y almacenamiento sin instalar máquinas localmente

5.5. Desventajas del *Cloud Computing.*

- La información está accesible a terceras empresas.
- Dependes de los servicios en línea de la empresa que ofrece el servicio. Puede ser que el servicio se caiga por alguna circunstancia
- Dependes de una conexión a la red para acceder al servicio. Entonces, también entran en juego, el tipo de conexión que tenga o la velocidad de la misma.
- El servicio es susceptible a ataques que nos tiren nuestros servidores en línea. Aunque el servicio que dan las empresas suele ser bastante seguro.

6. MaaS (Metal as a Service)

6.1. ¿Qué es MaaS?

MaaS (Metal as a Service) es una herramienta de software libre que permite crear un centro de datos mediante servidores *Bare-Metal*. Esta herramienta te permite desplegar y comisionar los nodos de computación, además de configurar las redes de un entorno como el que se quiere presentar. Esta herramienta es útil porque permite agregar/configurar servidores de nuestra nube de computación de manera coherente, ordenada y con un tiempo y esfuerzo mínimo.

La gran ventaja de esta herramienta de la empresa *Canonical*, es que no se necesita tratar a cada nodo de computación de manera individual. Desde la interfaz de *MaaS* es posible controlar toda la funcionalidad necesaria para crear un ecosistema *Cloud*.

En nuestro caso, se combinará con otra herramienta de la cual se hablará en el siguiente apartado, llamada *JuJu*. Este controlador nos hará posible el despliegue de todas las aplicaciones necesarias para el buen funcionamiento de la nube.

Además, estos servidores que conforman el escenario, no es necesario instalarles el sistema operativo de manera convencional mediante una unidad *USB* por ejemplo, el propio software de *MaaS* nos proporciona el sistema operativo que va a correr en los nodos de computación. Esto es gracias al *arranque PXE*. Es necesario que todos nuestros servidores dispongan de esta tecnología (*PXE*). Esta forma de desplegar el sistema operativo no es más que un arranque por red que se puede configurar en la *BIOS* de los servidores.

Figura 5: Logo de *MaaS*

6.2. Instalación de MaaS

6.2.1. Preinstalación del servidor MaaS

Se necesitan unos pasos previos antes de instalar el *Software MaaS* para preparar el escenario que se ha explicado en apartados anteriores. Los requisitos necesarios que tiene que tener la máquina en la que se va a instalar el *software* son:

- El sistema operativo que debe tener el servidor *MaaS* es Ubuntu Server 20.04 LTS (Focal).
- La versión del *Software MaaS* es la 2.9.2
- La versión del agente *JuJu* es la 2.9.0
- La versión de *OpenStack* que se instalará en los nodos es la *OpenStack Wallaby*.

Además, esta máquina, va a ser el router por el cual todos los servidores de cómputo e instancias que se levantan en nuestro escenario van a salir a Internet, por lo tanto se necesita configurar sus interfaces de red mediante *Netplan*. Se entra en el fichero de configuración que se encuentra en la ruta: /etc/netplan/. Se edita el fichero de la siguiente manera:

```
# This file describes the network interfaces available on your
system
# For more information, see netplan(5).
network:
 version: 2
 renderer: networkd
 ethernets:
  eno1:
   match:
    macaddress: e0:69:95:c8:f9:22
   set-name: eth0
   dhcp4: no
   addresses:
    - 172.29.21.202/22
   gateway4: 172.29.20.1
   nameservers:
    addresses:
     - 192.168.153.141
     - 192.168.153.144
    search: []
  enp3s0:
   match:
    macaddress: 64:66:b3:05:15:b6
   set-name: eth1
   dhcp4: no
   addresses:
    - 10.0.1.1/24
   nameservers:
    addresses:
     - 192.168.153.141
     - 192.168.153.144
    search: [maas]
  enp2s0:
   match:
    macaddress: 00:00:e8:74:b8:ea
   set-name: eth2
   dhcp4: no
   addresses:
    - 10.0.3.1/24
   nameservers:
    addresses:
     - 192.168.153.141
     - 192.168.153.144
    search: [maas]
 vlans:
  eth1.2:
   id: 2
   link: enp3s0
   addresses: [ "10.0.2.1/24" ]
   nameservers:
    addresses:
     - 192.168.153.141
     - 192.168.153.144
    search: [maas]
```

```
  eth2.4:
   id: 4
   link: enp2s0
   addresses: [ "10.0.4.1/24" ]
   nameservers:
    addresses:
     - 192.168.153.141
     - 192.168.153.144
    search: [maas]
```

Como se puede observar, se usan 3 interfaces de red:

- La primera se ha llamado *eth0*, esta interfaz es la que nos dará acceso a internet. La IP que tiene asignada es: 172.29.21.202/22. Esta IP es una disponible del laboratorio. El *gateway* por el cual se sale a internet es 172.29.20.1 según la configuración.
- La segunda y tercera interfaz que se van a configurar, se han llamado *eth1 y eth2*, respectivamente. Son interfaces de red privadas que se usarán para el tráfico interno de la nube. Las IP´s que se le han asignado son 10.0.1.0/24 y 10.0.3.0/24 con puertas de enlace 10.0.1.1 y 10.0.3.1, respectivamente. Como se puede observar en la configuración de estas dos interfaces cuelgan dos vlans: eth1.2 y eth2.4. Estas dos vlans (Virtual LAN´s), llevarán en ellas el tráfico de red de la nube.

Una vez editado el fichero, hay que asegurarse que no hay mas fichero .yaml en el mismo y se ejecutan los siguientes comandos para aplicar la configuración:

```
sudo netplan --debug generate
sudo netplan --debug apply
```

Si existiera algún error en la configuración, con la etiqueta *–debug* se puede ver qué es lo que está pasando. Para que este servidor haga las funciones de router que se requieren, hay que ejecutar la siguiente orden:

```
sudo ufw enable
```

Ahora para que el sistema permita reenviar paquetes se edita el archivo /etc/ufw/sysctl.conf de la siguiente manera:

```
net/ipv4/ip_forward=1
```

En algunos sistemas operativos simplemente es descomentar la línea. Ahora se tiene que ejecutar el siguiente *script* para que se acepte el tráfico de paquetes entre las interfaces del servidor. El *script* se ejecuta con:

```
sudo sh inicio-script.sh
```

Después de ejecutar el script, si vemos la tabla de rutas NAT con el comando, *sudo iptables -t nat -L* se observa en la tabla de *POSTROUTING* lo siguiente:

```
usermaas@maas:~$ sudo iptables -t nat -L
[sudo] password for usermaas:
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
[

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  anywhere             anywhere
```

Figura 6: Salida del comando iptables anterior

El *script* de nombre inicio-script.sh contiene una serie de órdenes para el reenvío de paquetes entre interfaces. Se utiliza la tecnología *Software iptables* para que el servidor haga las funciones de router.

```
#!/bin/bash
# /etc/rc.local Default policy to drop all incoming packets.
iptables -P INPUT DROP
iptables -P FORWARD DROP
# Accept incoming packets from localhost and the LAN interface.
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -i eth1 -j ACCEPT
iptables -A INPUT -i eth1.2 -j ACCEPT
iptables -A INPUT -i eth2 -j ACCEPT
iptables -A INPUT -i eth2.4 -j ACCEPT
# Accept incoming packets from the WAN if the router
initiated the connection.
iptables -A INPUT -i eth0 -m conntrack --ctstate ESTABLISHED,
    RELATED -j ACCEPT
# Forward LAN packets to the WAN.
iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
iptables -A FORWARD -i eth1.2 -o eth0 -j ACCEPT
iptables -A FORWARD -i eth2 -o eth0 -j ACCEPT
iptables -A FORWARD -i eth2.4 -o eth0 -j ACCEPT
# Forward WAN packets to the LAN if the LAN initiated
the connection.
iptables -A FORWARD -i eth0 -o eth1 -m conntrack --ctstate
    ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -i eth0 -o eth1.2 -m conntrack --ctstate
    ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -i eth0 -o eth2 -m conntrack --ctstate
    ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -i eth0 -o eth2.4 -m conntrack --ctstate
    ESTABLISHED,RELATED -j ACCEPT
# NAT traffic going out the WAN interface.
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Por último, se ha de habilitar las conexiones *SSH* en el *firewall* y se reinicia el servidor:

```
sudo ufw allow ssh
sudo reboot now
```

Se vuelve a correr el *script, inicio-script.sh*, ya que se ha reiniciado la máquina con el comando anterior.

En caso de que se produjera un apagado forzado por un corte de luz u otra situación indeseada, se enciende el servidor a mano y se correría el *script* anterior *(inicio-script.sh)* . A partir de este punto, todo volvería a funcionar de forma coherente.

6.2.2. Instalación del servidor MaaS

En esta sección se da a conocer la instalación del servidor que va a hacer la función de *MaaS*. Los comandos necesarios son los siguientes:

```
sudo snap install maas-test-db
sudo snap install maas --channel=2.9/stable
sudo maas init region+rack --maas-url http://10.0.1.1:5240/MAAS --database-uri
    maas-test-db:///
sudo maas createadmin --username admin --password admin --email admin@example.com
sudo maas apikey --username admin > ~/admin-api-key
```

Como se puede observar, con el comando *maas init* se coloca en la IP que se muestra, la API web de *MaaS* y con *maas createadmin*, se crea un usuario con el cual nos autenticaremos cuando se acceda al recurso. Cabe resaltar, que el comando que inicia *MaaS* (*maas init*) no puede ser invocado de nuevo porque nos aparecería un mensaje de error.

6.2.3. Post-instalación del servidor MaaS

El siguiente paso es entrar en la interfaz gráfica de *MaaS*. Se accede mediante la IP privada:

```
http://10.0.1.1:5240/MAAS
```

Si se quiere hacer pública mediante la VPN(*Virtual Private Network*) de la universidad la interfaz por la cual se va a controlar todos los nodos de cómputo, se pueden ejecutar los siguientes comandos *iptables* que realizan una traducción *NAT 1-to-1* de la dirección en la que se ha colocado nuestro servidor, con una disponible del laboratorio:

```
sudo ufw allow 5240
sudo iptables -t nat -A PREROUTING -i eth0 -d 172.29.21.202 -j DNAT -p tcp
    --to-destination 10.0.1.1:5240
sudo iptables -A FORWARD -d 10.0.1.1 -j ACCEPT
```

De esta manera, si nuestro portátil personal está dentro de la red del laboratorio o en la VPN de la UAH y se accede con un navegador a la siguiente dirección por el puerto 80, también se vería la interfaz web de *MaaS*:

```
http://172.29.21.202:80/MAAS
```

Esta interfaz gráfica nos da información de los servidores que soportan todas las aplicaciones de *OpenStack*. El escenario inicial y estándar que facilita la página web oficial de *OpenStack* [5] propone que se usen cinco servidores más:

- 1 de ellos dedicado al controlador *JuJu*.
- 4 servidores que harán de nodos de cómputo.

Esto puede estar sujeto a cambios, ya que se harán pruebas con menos nodos de cómputo y se estresará a los servidores para ver cuanto es capaz de soportar la nube. Una vez se accede a la API de *MaaS* se ve lo siguiente:

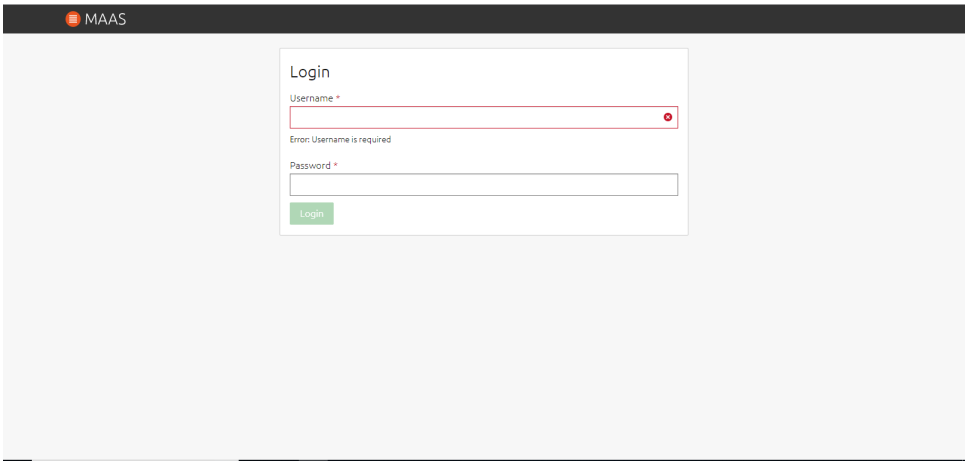


Figura 7: Pantalla inicio de la API de MaaS

Las credenciales de acceso han sido configuradas en la anterior sección del proyecto [6.2.3]. En nuestro caso, el usuario es *admin* y la contraseña es *admin*.

Una vez dentro lo siguiente que se ve es la figura [8]

En esta pantalla lo único que hay que configurar es el *DNS* por defecto, se utilizan algunos generales de internet, el 8.8.8.8, el 1.1.1.1 y 4.4.8.8. Se pulsa en continuar y se pasa a la siguiente pantalla. Es importante verificar que se ha descargado la imagen del sistema operativo que se va a instalar en los nodos de cómputo, en nuestro caso *Ubuntu 20.04*. Esto se sabe, porque en la columna *status* aparecerá en estado *Synced*.

Tras pulsar en continuar, la siguiente pantalla (admin, sshkeys) nos pedirá una clave pública que el sistema utilizará para establecer las claves de acceso a los nodos de cómputo. Esta clave se puede importar de servicios como *GitHub* o *Launchpad* o directamente del mismo usuario *maasuser* que se está utilizando en nuestro servidor *Maas*. Esta clave pública, se encuentra en el directorio home /maasuser /.ssh /idrsa.pub.

Si no existe dicha clave se genera mediante el comando:

```
ssh-keygen -t rsa
```

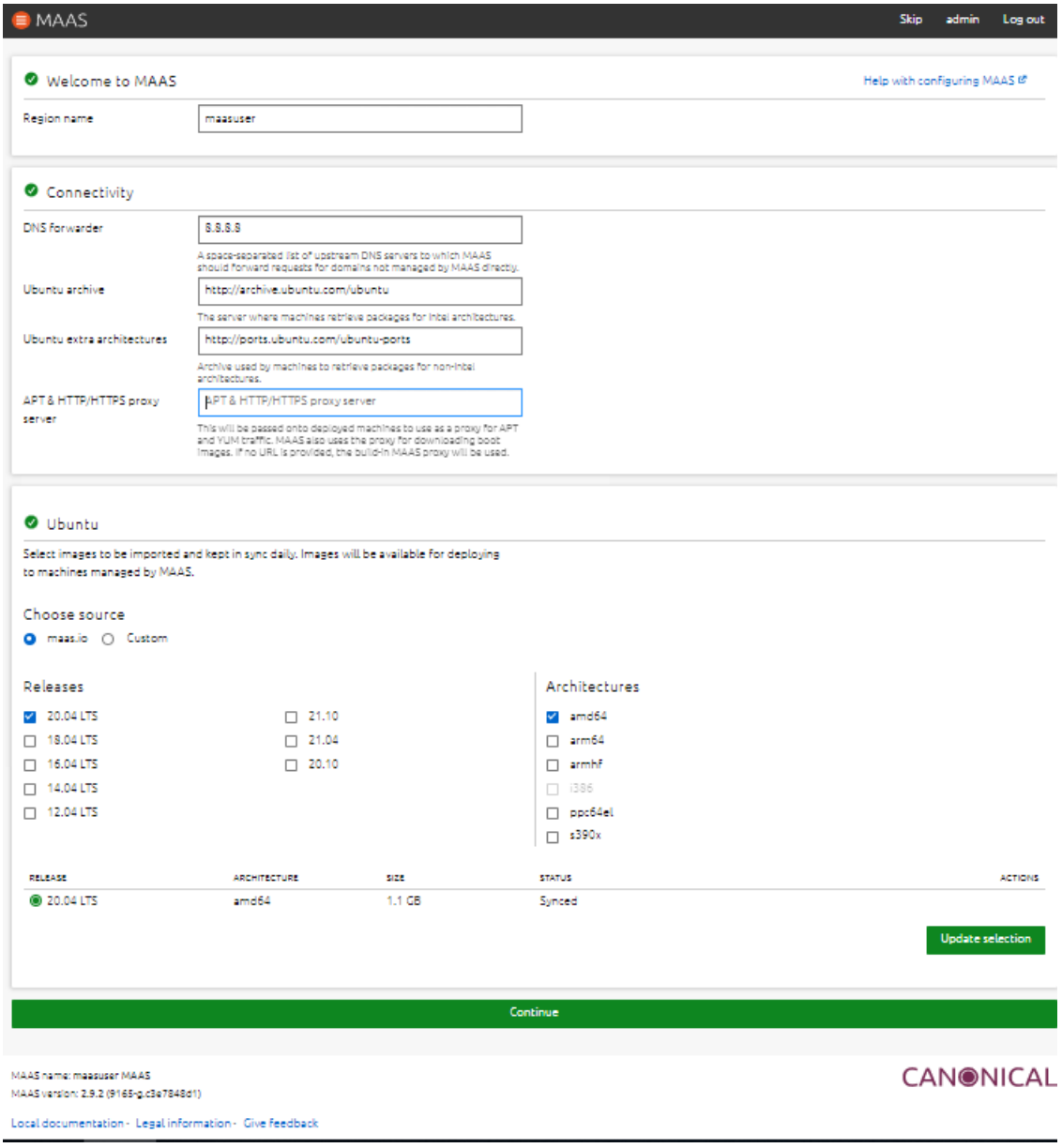


Figura 8: Segunda pantalla de la API de MaaS

Una vez se tenga localizada la clave, se edita con el vim (no con more), se copia y se introduce en la siguiente pantalla, seleccionando la opción *Upload*:

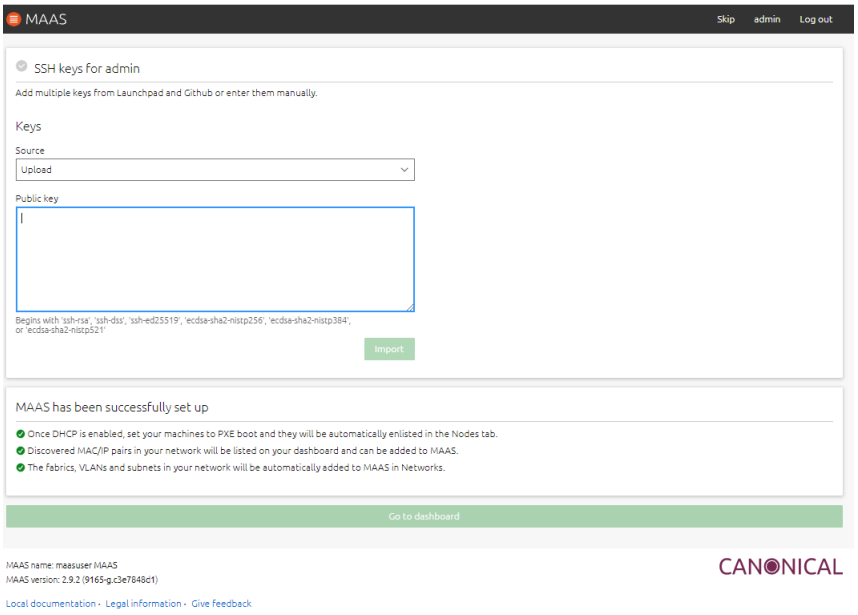


Figura 9: Clave publica del servidor *MaaS*

En este punto, se puede dar un punto de fallo. Si no introducimos bien la clave publica, no se podrá acceder a los nodos de cómputo que se despliegan en nuestro escenario. Como solución a esto, nos situaremos en la siguiente pantalla, se selecciona la opción *Upload* y se mete la clave pública del *MaaS*. Cabe resaltar, que esto se realiza con las máquinas en estado *Deployed*.

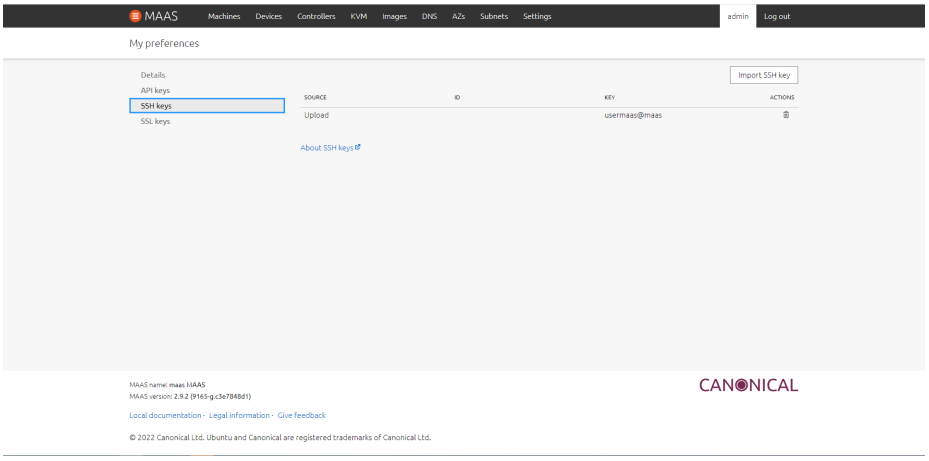


Figura 10: Corrección acceso *SSH* de los nodos

Se pulsa sobre la opción *Import SSH Keys* y se selecciona *Upload*. Se copia la clave pública del *MaaS* (se encuentra en el directorio /home/maasuser/.ssh) que se ha creado con el comando *ssh-keygen* anterior.

En este punto, no se tendría acceso *SSH* a las máquinas por la manera convencional. Sólo tendría acceso el agente *JuJu*. Con el comando siguiente, se accede a la maquina:

```
juju ssh NUMERO_DE_MAQUINA
```

Nos dirigimos al directorio */home/ubuntu/.ssh* del nodo y en el fichero *authorized keys* se copia la clave

pública del servidor *MaaS*.

A partir de aquí, ya se tiene acceso a las máquinas y se corrige el error. Se podrá acceder a las máquinas de cómputo de manera convencional con el siguiente comando:

```
ssh ubuntu@10.0.1.X
```

Se continúa hacia la siguiente pantalla, que nos mostrará la pizarra dónde aparecerán las máquinas que van a formar la nube. El aspecto de esta pantalla es el siguiente:

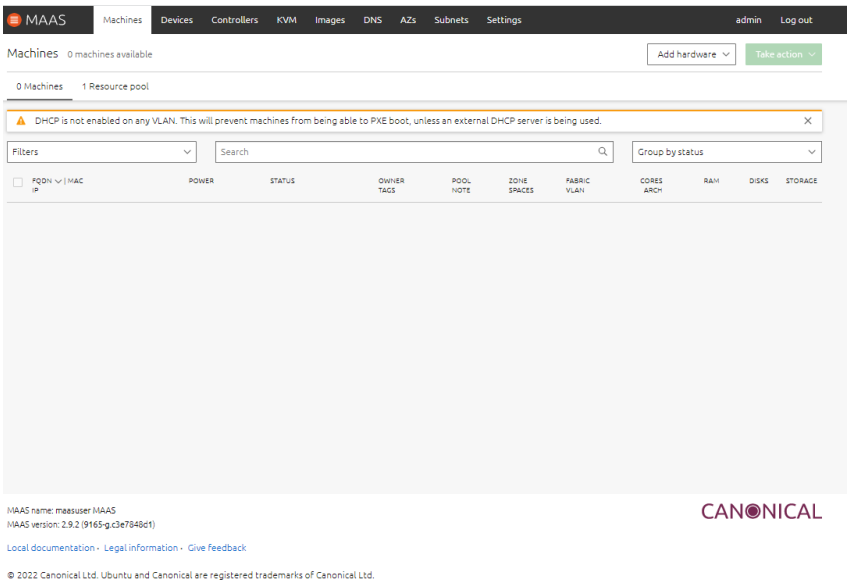


Figura 11: Pizarra máquinas de cómputo

6.2.4. Configuración de red

Si nos fijamos en la siguiente imagen [12], el controlador *MaaS* tiene configuradas las interfaces de red tal y como se ha planeado con *Netplan* en el apartado de preinstalación del servidor [6.2.1]. El apartado *Controllers* de la *API* demuestra esto:

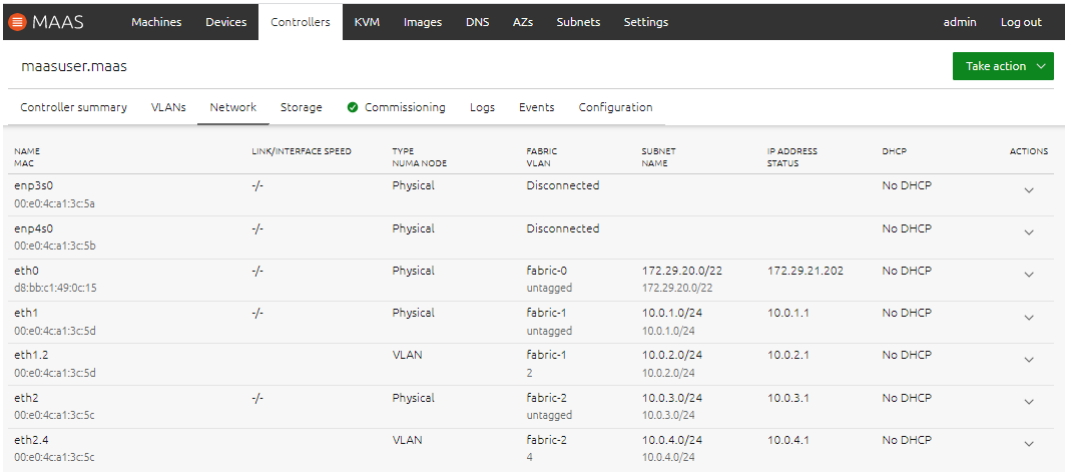


Figura 12: Interfaces de red del servidor *MaaS*

Una vez llegados a este punto, el servidor *MaaS* está casi preparado para hacer su función. Antes de

empezar a desplegar las máquinas que configuran nuestro sistema, se deben configurar las redes que van estar presentes en el mismo.

Nuestro escenario inicial presentará una arquitectura de red que viene reflejada en la siguiente ilustración [13].

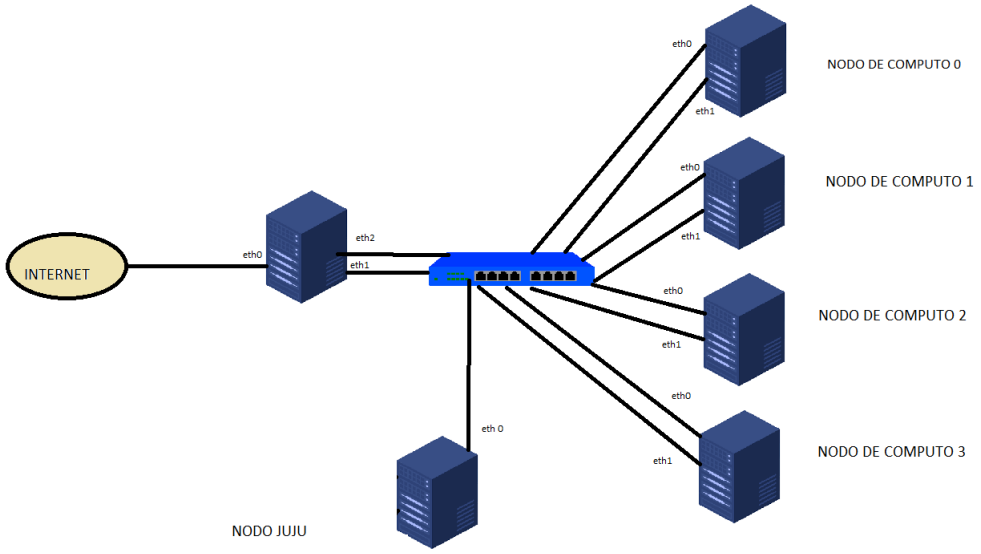


Figura 13: Arquitectura de red general

Siguiendo la distribución del tráfico de red de la nube que se realizó en el escenario inicial citado [9]. Como se puede observar, cada nodo de cómputo tiene conectadas dos interfaces de red al *switch Netgear*. De ahí que se configuren cuatro redes diferentes para el tráfico en cada nodo. De la tarjeta de red física de un nodo, colgará la propia de eth0 (10.0.1.0/24) y una VLAN asociada eth0.2 (10.0.2.0/24) y en la otra interfaz de red estará presente eth1 (10.0.3.0/24) y la VLAN asociada eth1.4 (10.0.4.0/24).

El sistema nos reconoce por defecto algunas redes, el escenario que se encuentra a primera vista es el siguiente:

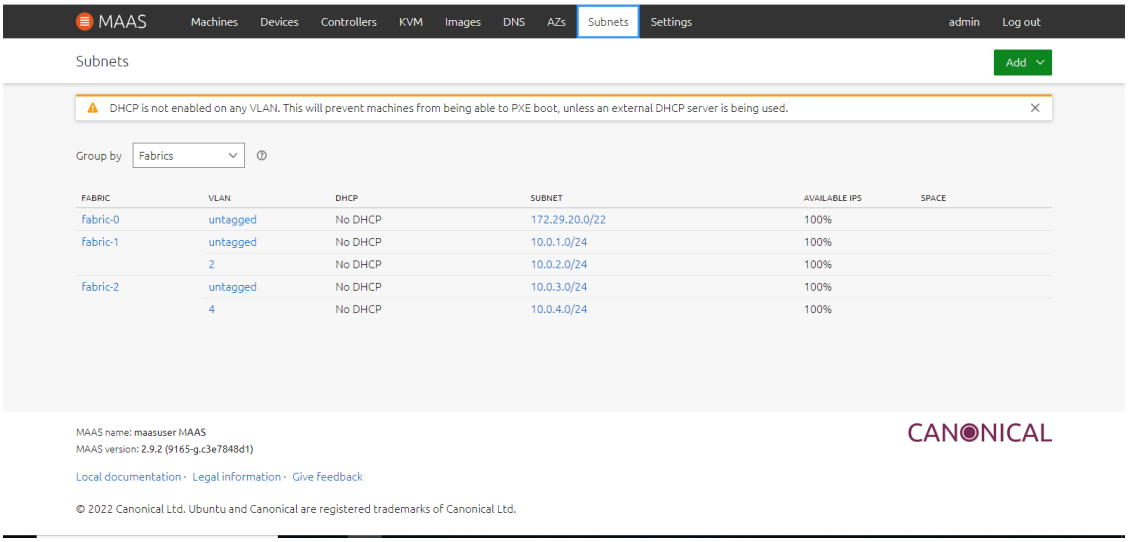


Figura 14: Pestaña *Subnets* de API *MaaS*

Como se observa en la imagen anterior [14] se dispone de tres *fabrics* (por *fabric* se entiende un conjunto

de redes que tienen relación entre ellas), el primero de ellos contiene la red exterior de *MaaS*, se cambia el nombre por *maas-exterior*, el segundo de ellos es el que se usa para definir las redes interiores de nuestra nube, se le pone el nombre de *maas-cloud* y el tercero se desecha, las subredes que están contenidas en el mismo se pasan a *maas-cloud*.

Ahora se pasa a configurar la columna *VLAN (Virtual LAN)*. Se le asigna un *VID (Virtual Identity)* a cada una de ellas. Se pincha sobre las que están en estado *untagged* y se edita el campo en cuestión como se muestra en la figura siguiente [15]. También se asocian las *VLAN* que pertenecen al *fabric-2*, para que aparezcan en el *fabric-1* que se ha llamado *maas-cloud*.

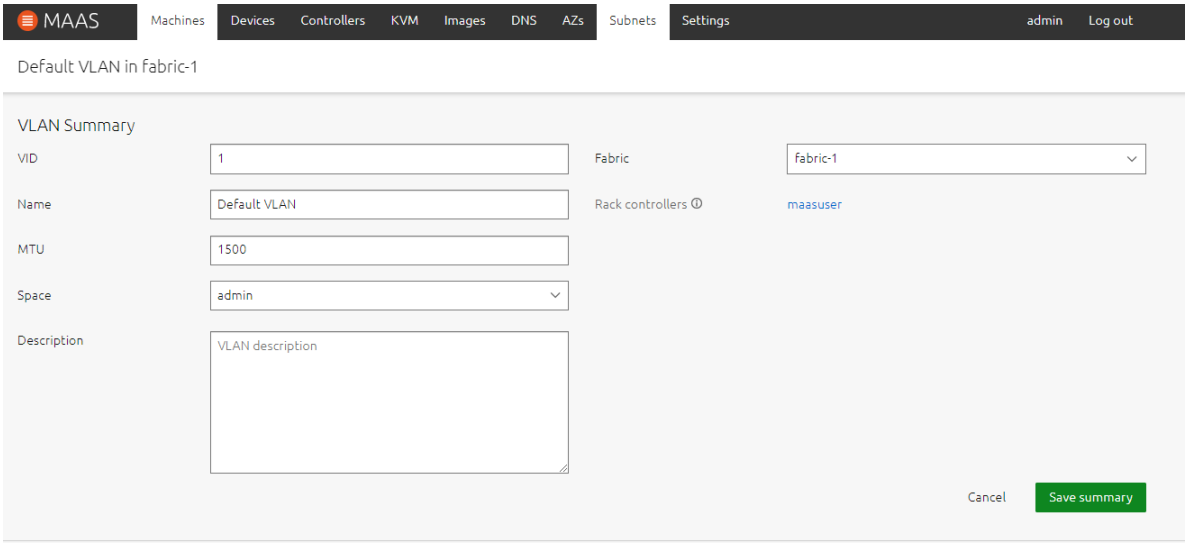


Figura 15: Editando *VLAN* de cada *fabric*

Como se puede observar en la figura anterior [15], a cada *VLAN* se le asociará un espacio (por espacio se entiende espacio de red, cada red tendrá asociado un espacio). Para ello, se selecciona la opción *Add* situado a arriba a la derecha de la ilustración [14] y en el desplegable se elige *Add Space*, seguidamente se crean los siguientes espacios:

VID	Espacio	Red asociada
1	admin	10.0.1.0/24
2	cómputo-ext	10.0.2.0/24
3	público	10.0.3.0/24
4	interno	10.0.4.0/24

Como aclaración indicar que estos VID introducidos como VLANs no se traducen después a la configuración de red en el netplan. También hay que indicar que dentro de un fabric no puede haber VID VLANs repetidos, esto incluye el VID untagget. Otra limitación es que dentro de cada *Space* solo se puede haber un *VID VLAN*. Se asocian estos espacios secuencialmente, según el VID de cada VLAN configurado anteriormente.

El siguiente paso es dotar a las subredes de una puerta de enlace (*gateway*), por el cual tendrán acceso a internet, y de un conjunto de direcciones que puedan usar. Pulsando sobre cualquiera de las direcciones de la columna *Subnet* [14] se hace esta configuración. La siguiente figura muestra la pantalla de configuración de una de las subredes:

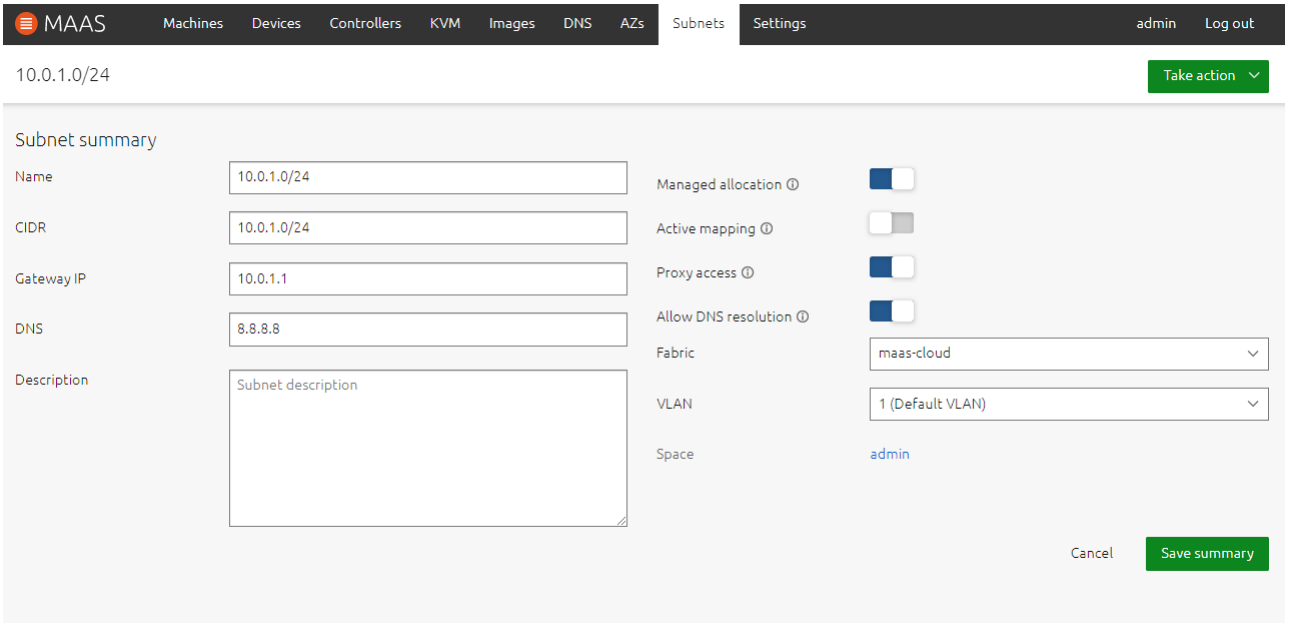


Figura 16: Configuración de las subredes

Ahora se tiene que habilitar el protocolo *DHCP (Dynamic Host Control Protocol)*, para ello en la columna *VLAN* se pincha sobre la subred 10.0.1.0/24 de la figura [14] y dentro de esa pantalla se habilita el protocolo mencionado según muestra la siguiente figura (*Enable DHCP*):

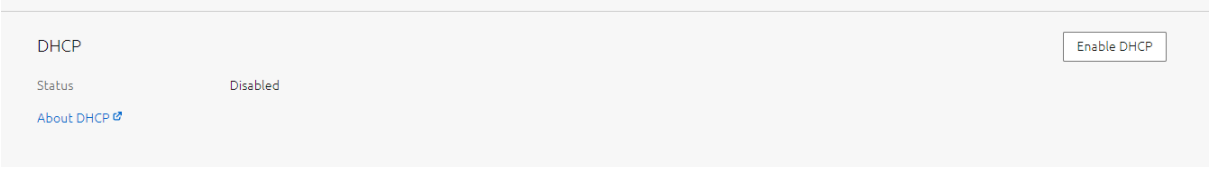


Figura 17: Activando *DHCP* en las subredes

En este apartado, se elige la opción que nos reserva el rango de direcciones dinámicamente. Queda configurado nuestro escenario para utilizar *DHCP*. Es importante este paso, ya que gracias a este protocolo se puede dotar a nuestros nodos de cómputo de un sistema operativo. El servidor *MaaS* será también un servidor *DHCP* a ojos de los nodos que componen la nube. La configuración viene reflejada en la ilustración [18].

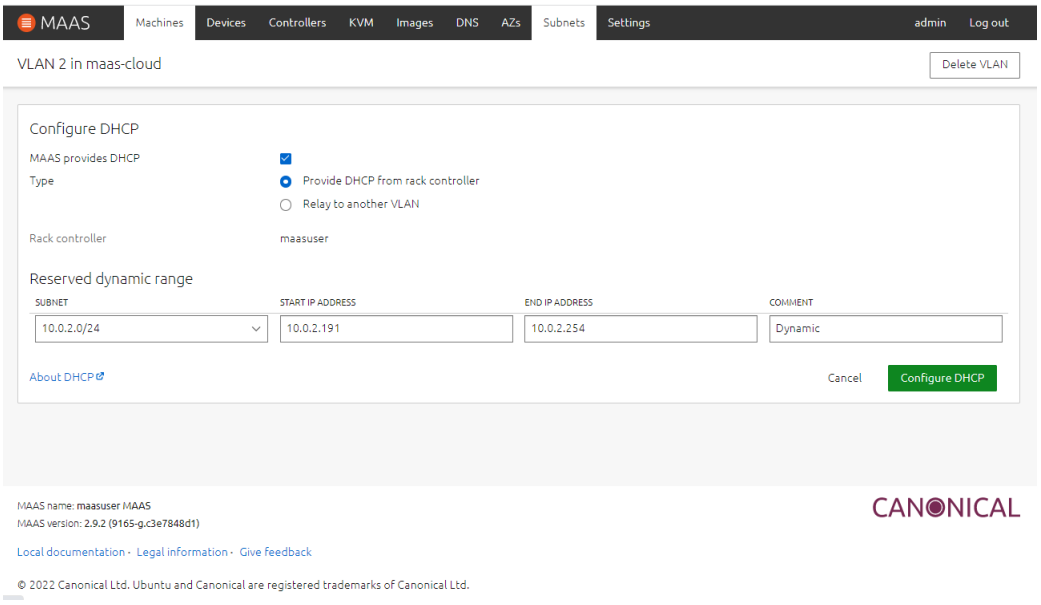


Figura 18: Reserva del rango de direcciones

El aspecto final de la pantalla de configuración de las subredes sería el siguiente:

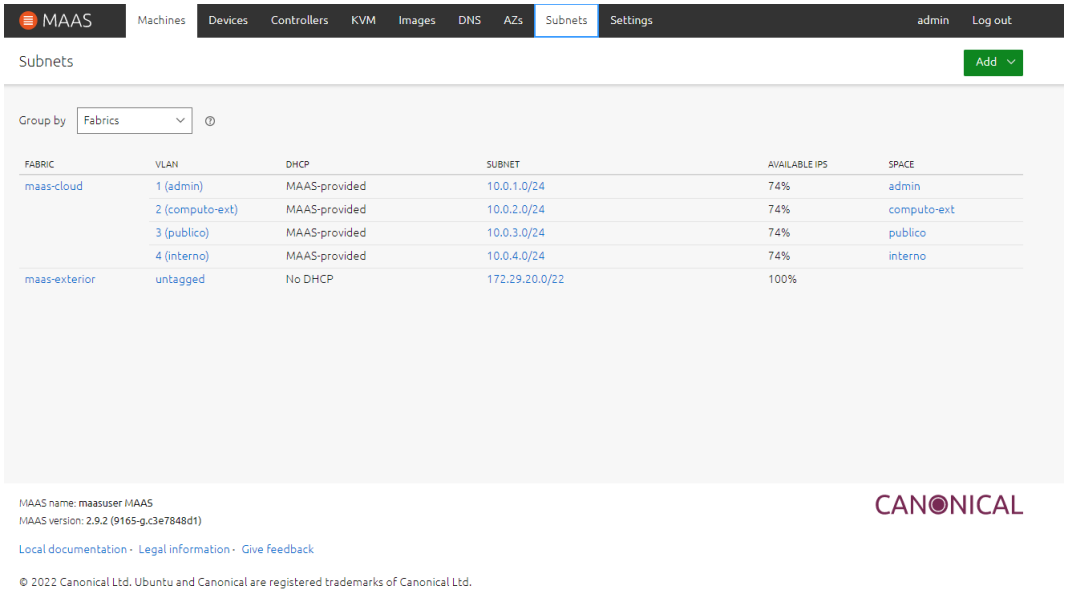


Figura 19: Aspecto final de la pantalla de subredes

6.2.5. Reconocimiento de los nodos

Llegados a este punto, nuestro escenario está listo para que el servidor *MaaS* empiece a reconocer los nodos de cómputo que van a montar nuestro sistema *Cloud*. Lo primero que se debe comprobar es que el arranque de los nodos este configurado de manera que la primera opción sea, el arranque por red (*PXE*).

Figura 20: ASpecto de la BIOS de los servidores

Una vez esté configurado el arranque *PXE* (que puede tardar 1 minuto), se pasa al reconocimiento de las máquinas. En este punto, se encienden los nodos de cómputo uno a uno. La razón de encenderlos de uno en uno es porque *MaaS* por defecto, da un nombre al azar cuando reconoce las máquinas y si se encienden los nodos de manera simultánea, no se sabe que nodo se ha reconocido. La siguiente imagen 21 nos muestra el reconocimiento de un nodo, como se observa, el nombre al azar que pone *MaaS* es *above-yeti.maas*. Se aprovecha para cambiar a un nombre más familiar, a los nodos de cómputo. Para cambiar el nombre de la máquina lo que se hace es pulsar sobre el nombre, aparecerá otra pantalla, en la cual arriba a la izquierda si se pulsa sobre el nombre otra vez, nos dejará modificarlo. Las máquinas quedan apagadas automáticamente.

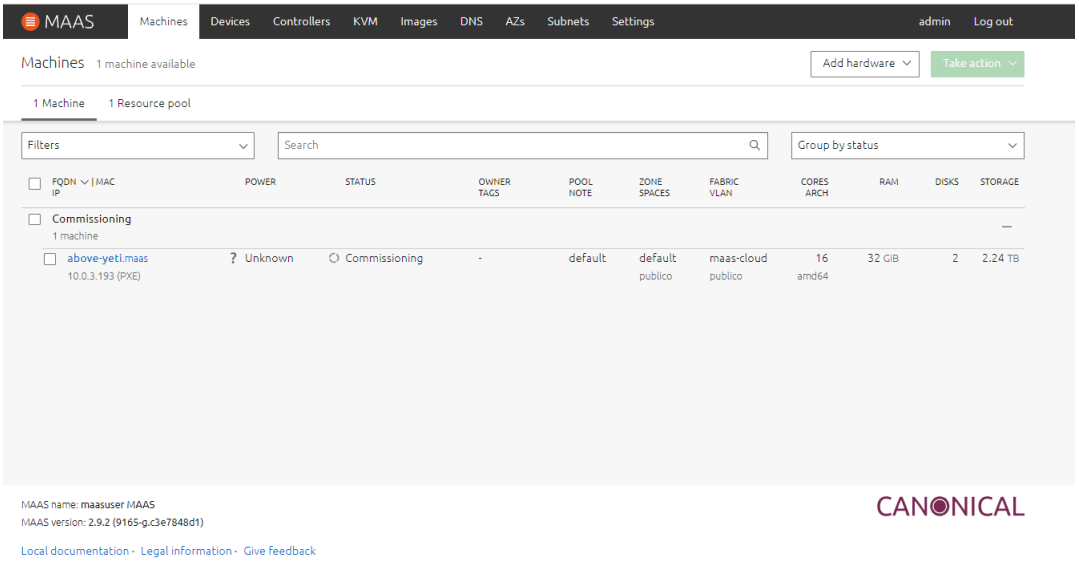


Figura 21: Reconocimiento de una máquina por el servidor *Maas*

Cuando se tengan las cinco máquinas reconocidas del escenario inicial (cuatro nodos de cómputo y un servidor *JuJu*), hay que configurar el modo en el que van a arrancar las máquinas. En nuestro caso se elige la opción de encender las máquinas de manera manual, el encendido a través de la red a veces no iba. Para ello, se pincha en el nombre de la máquina desde la pantalla de la figura [21] opción *Configuration* y se cambia el modo de arranque como muestra la siguiente figura:

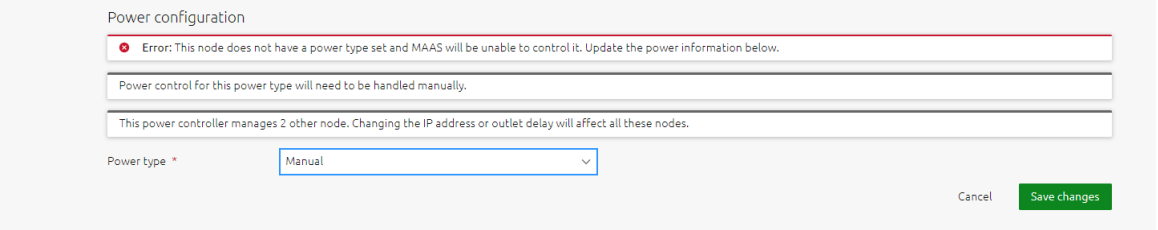


Figura 22: Elección del modo de arranque de los nodos

6.2.6. Comisionado de los nodos

Una vez se ha configurado el modo de arranque de los nodos ya se pueden comisionar (el concepto de comisionar, consiste en integrar el nodo en la red del sistema) nuestros cinco nodos de cómputo. Antes de realizar este comisionado, se tienen que dotar de una etiqueta para diferenciar a la hora de desplegar nuestro modelo, qué papel juega el nodo en cuestión (nodo de cómputo o servidor *JuJu*). Para ello, se seleccionan los nodos y se hace click en el botón *Configuration*, luego en el botón *Edit* [21] y se pulsa sobre *Tag*. Con dos etiquetas será suficiente, una para el controlador *JuJu* y otra para los nodos de cómputo *compute*.

En este momento se comisionan los nodos. Se seleccionan los cinco nodos y se pulsa sobre *Take action*, se desplegará el menú y se selecciona la pestaña *Commision*. Ahora, los nodos cambiaran de estado a *Commisioning*. Como es lógico, al haber configurado que el modo de arranque sea de forma manual, hay que encender el servidor pulsando el botón de encendido. El estado final de cualquiera de los nodos si todo va bien, será *Ready*. Cuando termina el comisionado, los nodos se apagan automáticamente.

6.2.7. Configuración de los interfaces de red

El siguiente paso es la configuración de las interfaces de red de cada nodo (ya sea el nodo *JuJu* o los nodos de cómputo). Debido a un error puede que haya que repetirlo un par de veces para que funcione.

Para configurar las interfaces, se pincha sobre el desplegable de cada interfaz de red (situados en la parte derecha, columna *ACTIONS* de la ilustración [23] o [24]) y se selecciona la opción *Edit physical*. Hay que:

1. Cambiar el nombre de la tarjeta de red, la de arranque (tiene un icono verde activo) a eth0, la otra a eth1.
2. Seleccionar una VLAN de acuerdo con la asignación de las figuras siguientes.
3. Seleccionar una IP viendo las figuras siguientes.
4. En IP ADDRESS STATUS seleccionar *Auto Assign*.

En el caso de *Juju* la *VLAN* 1 y cambiar el nombre de la tarjeta de red a eth0, VLAN 1, debe quedar la siguiente configuración de sus interfaces:

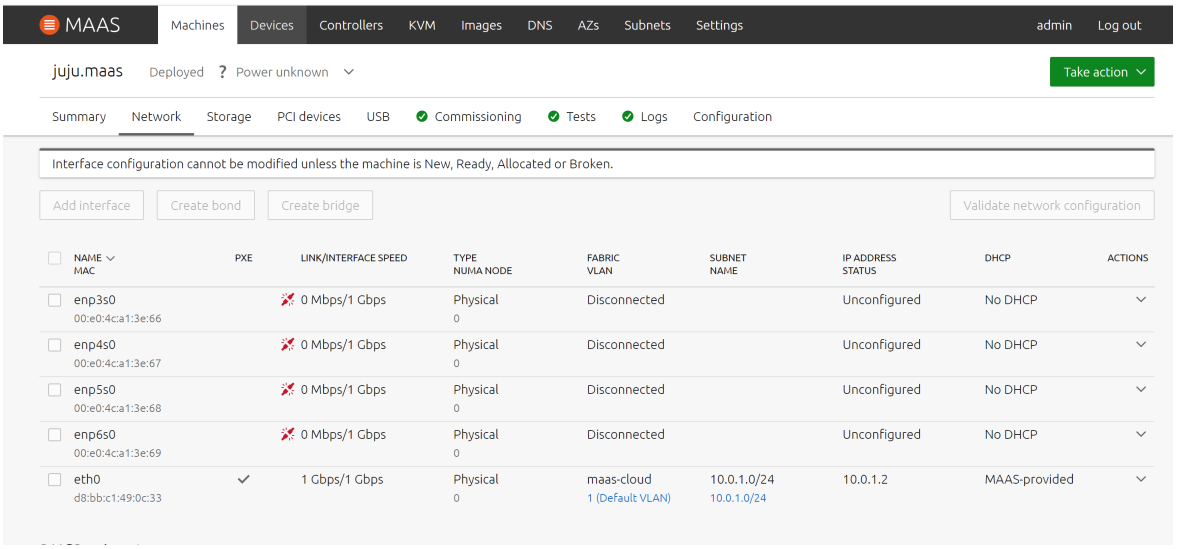


Figura 23: Interfaces de red del servidor *JuJu*

En el caso de los nodos de cómputo, además hay que añadir las VLAN´s asociadas a las interfaces eth0 y eth1 pulsando en el mismo desplegable de la interfaz de red, sobre la opción *Add VLAN or Alias* y en *Type* poner *VLAN*. Las interfaces de red de los nodos de cómputo se configuran de la siguiente manera:

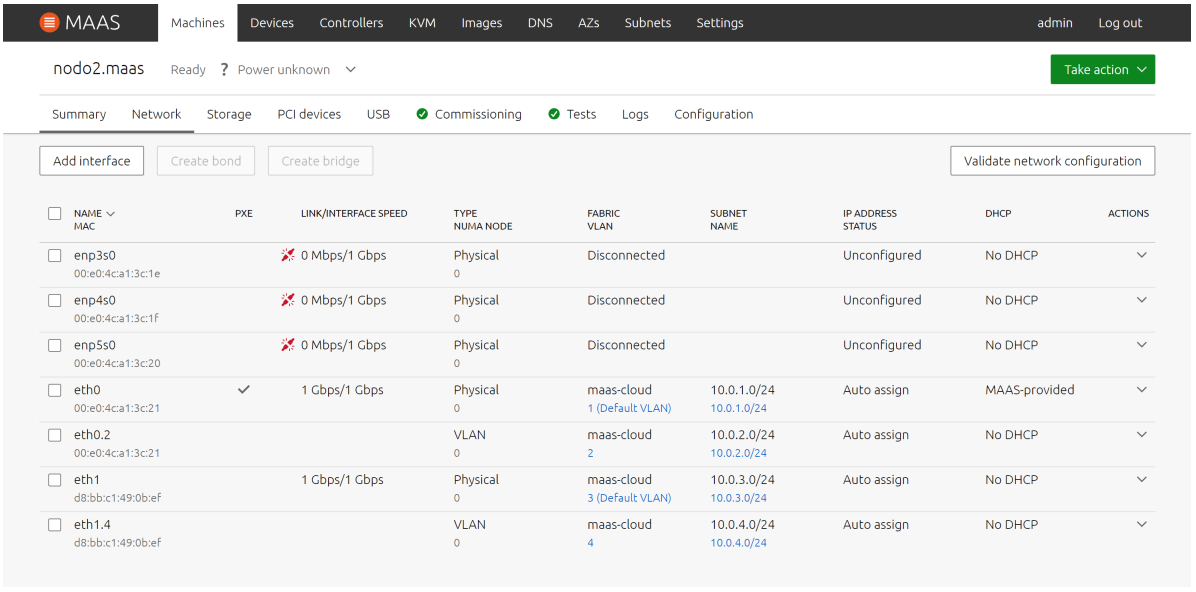


Figura 24: Interfaces de red de los nodos de cómputo

Como se puede observar en la imágenes [23] y [24] la configuración de las interfaces es bastante versátil. En nuestro caso, se tiene el tráfico de los nodos de cómputo separado mediante dos interfaces (eth0 y eth1) y de cada una de ella cuelga una *VLAN*. Esto se hace para separar el tráfico de datos evitando que éste dependa de una sola tarjeta de red. El controlador *JuJu*, solo dispone de una interfaz de red conectada al conmutador *Netgear*. Se considera que el tráfico que va a soportar es menos que el de los nodos de cómputo.

Las máquinas estarán en el siguiente estado según la pizarra de *MaaS*:

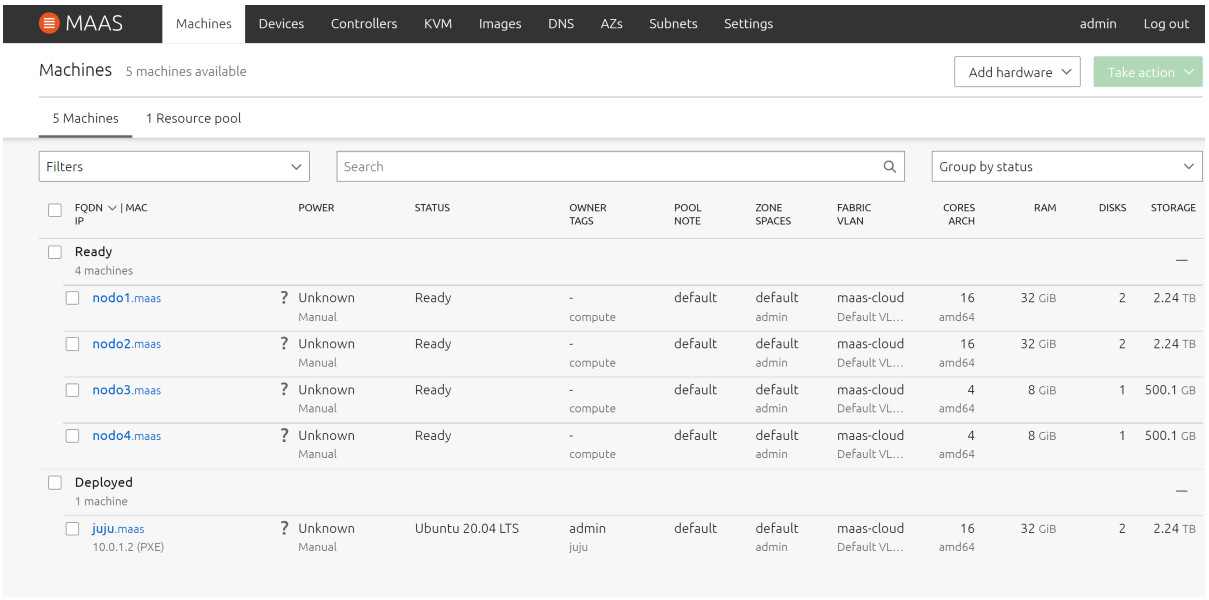


Figura 25: Estados de los nodos

7. JuJu

7.1. ¿Qué es JuJu?

JuJu es una herramienta de código abierto que está patentada por la empresa *Canonical*. Mediante esta herramienta se va a controlar el despliegue, configuración y operación en todo momento de las aplicaciones que van a estar presentes en nuestro entorno *Cloud*.

Esta herramienta, permite la orquestación de estructuras de alto nivel, las cuales están compuestas por un conjunto de aplicaciones, así como el establecimiento de las relaciones que haya entre ellas y la configuración de las mismas. Estas aplicaciones pueden ser configuradas mediante ficheros *.yaml* o directamente con órdenes del controlador *JuJu* de manera sencilla y ágil. Se utiliza el agente *JuJu* para desplegar todas las aplicaciones que van componer la nube pública que se crea.

JuJu es un agente con un ecosistema y conceptos más abstractos, amplios y, en ocasiones, difíciles que *MaaS*.

Una aplicación para *JuJu*, esta concebida de la misma forma y significado que en cualquier entorno ordinario. Un paquete, que dispone de algún recurso *software*, que está accesible a través de internet.

Se amplia un poco más la definición de esta herramienta gracias a que hace que las aplicaciones que orquesta sean receptivas y dinámicas:

- Dinámicas, porque gracias a este agente se puede configurar cualquier aspecto de la configuración de las aplicaciones, con el comando *JuJu* indicado para la situación.
- Receptivas, porque el agente *JuJu* es capaz realizar acciones que afectan a las aplicaciones que controla.

Al desplegar una aplicación en una máquina, *JuJu* despliega un agente de máquina, un *software* que funciona a nivel del hardware del nodo de cómputo y un agente de unidad de aplicación que opera a nivel de la aplicación que se ha instalado en dicha máquina. El agente de máquina administra, gestiona y crea los diferentes agentes de unidad que se necesiten en la máquina de cómputo, así como los contenedores que se crean en el despliegue de las aplicaciones (*Charms*).

Figura 26: Logo de *Juju*

7.1.1. Acciones y configuraciones que puede realizar el agente *JuJu*

- Si la configuración interna de alguna aplicación desplegada necesita ser cambiada, *Juju* proporciona el comando "Juju config" que permite cambiar ciertos parámetros como, por ejemplo, se puede configurar el protocolo de acceso a la consola virtual que ofrece la aplicación *extnova-cloud-controller* pudiendo elegir entre *VNC, noVNC, XVPVNC, None, Spice*:

  ```
  juju config nova-cloud-controller console-access-protocol=vnc
  ```

- Si por necesidades del servicio es necesario, por ejemplo, crear una base de datos *MySQL*, se puede escalar a dos los servidores *MySQL* activos con el comando:

  ```
  juju add-unit --num-units 2 mysql
  ```

  Además, el agente de *Juju* desplegado junto a la aplicación detecta automáticamente el escalado de esta aplicación y habilita un clúster de alta disponibilidad.

- Se pueden añadir volúmenes de almacenamiento a las aplicaciones que estén corriendo como, por ejemplo:

```
juju add-storage etcd/0 data=ebs-ssd,20G
```

7.1.2. ¿Qué es una relación?

Algunas aplicaciones ofrecen servicios que pueden trabajar de manera independiente. Sin embargo, existen otras aplicaciones que necesitan de relaciones con otras para su buen funcionamiento. Si una aplicación necesita de una base de datos, existe, por tanto, otra aplicación de base de datos, que puede cumplir con esta necesidad de una o varias aplicaciones. Cuando dos aplicaciones (*Charms*) se han unido de forma lógica, se dice que ha formado una relación(*relation*) entre ellas.

Las relaciones son protocolos que permiten que las aplicaciones se configuren automáticamente entre sí. No son una estructura de red, como un túnel entre dos aplicaciones. Sin embargo, son un protocolo de intercambio de datos definido para la posible negociación entre dos aplicaciones.

7.1.3. ¿Qué es un modelo?

Un modelo es un espacio de trabajo. En el entorno de *Juju* se hablará de modelo para referirnos a un espacio virtual donde están situadas, en este caso, las diferentes aplicaciones de *OpenStack* y las relaciones de las que se ha hablado, existentes entre las diferentes aplicaciones. Los modelos proporcionan una visión abstracta de la infraestructura que aloja su servicio.

7.2. Instalación del servidor JuJu

Una vez realizada la configuración del servidor *MaaS* en el anterior punto del proyecto, se procede a instalar el nodo que hará la función de agente *JuJu*. Para instalar el agente se ejecutan los siguientes comandos en el servidor *MaaS*:

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

Ahora se tiene que hacer que *JuJu* y *MaaS* tengan relación entre ellos, para ello, se usa el siguiente archivo (*maas-clouds.yaml*) de configuración:

```
clouds:
  mymaas:
    type: maas
    auth-types: [oauth1]
    endpoint: http://10.0.1.1:5240/MAAS
```

Se añade la nube con la configuración anterior de la siguiente manera:

```
juju add-cloud --client -f maas-cloud.yaml mymaas
```

Si se teclea en el terminal *MaaS*: *juju clouds –client* se ve que la configuración es correcta.

El siguiente paso es añadir las credenciales de *MaaS* para que el servidor *JuJu* pueda interactuar con él. Para ello se usa el siguiente archivo de configuración (*maas-creds.yaml*)

```
credentials:
  mymaas:
    anyuser:
      auth-type: oauth1
      maas-oauth: LGJ8svffZZ5kSdeA8E:9kVM7jJpHGG6J9apk3:KE65tLnjpPuqVHZ6vb97T8VWfVB9tM3j
```

El campo *maas-oauth* se tiene que mirar en la API de *MaaS*, accesible en *admin API key* o en la carpeta */admin-api-key* dentro del servidor que contiene la lógica de *MaaS*.

Se añaden las credenciales con el siguiente comando:

```
juju add-credential --client -f maas-creds.yaml mymaas
```

Para ver la lista de credenciales se usa el comando *juju credentials –client –show-secrets –format yaml* y se podrá comprobar que las credenciales se han añadido.

Se pasa a la creación de lo que es el controlador, para ello, se usará la etiqueta que se ha puesto puesto en el apartado anterior cuando se reconocen las máquinas mediante el API de *MaaS*. Cada vez que se ejecuta el siguiente comando se tiene que reiniciar el servidor donde se despliega *JuJu*:

```
juju bootstrap --bootstrap-series=focal --constraints tags=juju --agent-version=2.9.28 mymaas maas-controller
```

En el momento que se ejecuta este comando la máquina etiquetada con el nombre *JuJu* pasa del estado *Ready* a *Deploying* automáticamente. Como es lógico para que avance el proceso, hay que encender la máquina y configurar en la BIOS, como ya se ha explicado, que el arranque sea PXE.

El arranque *PXE* es una forma de decirle a cualquier servidor que instale el sistema operativo, en nuestro caso *Ubuntu Server 20.04*, mediante una interfaz de red. Es una configuración sencilla en la *BIOS* de los servidores, simplemente se cambia el orden de arranque del servidor poniendo como primera opción *PXE*.

En caso de tener que cambiar el PC del nodo *JuJu* a otro PC habría que hacer la fase anterior de reconocimiento del nodo, configuración de la red, comisionado y seguir a partir del comando *juju bootstrap* anterior.

Si por cualquier problema se tuviera que tirar el controlador *JuJu*, se ejecuta el siguiente comando:

```
juju destroy-controller <nombre controlador>
o
juju kill-controller --destroy-all-models maas-controller --force
```

Después de esto, si se ejecuta el comando *juju controllers*, ya no aparece en la lista de controladores. Si se quiere volver a desplegar el controlador se vuelve al paso *juju bootstrap*. En caso de querer eliminar la máquina, habría que eliminarla en el interfaz web de *MaaS*. Se pulsa sobre *Take action* en la pestaña *Machines* y se selecciona la opción *Delete*. De este modo, la máquina desaparecería de nuestra pizarra. Si se quiere volver a reconocer y comisionarla en nuestro sistema, habría que seguir los pasos que se han descrito en el apartado [6.2.5].

Si falla el comando *juju bootstrap* con un mensaje de error como el siguiente, se repite el paso anterior.

```
ERROR failed to bootstrap model: bootstrap instance started but did not
    change to Deployed state: instance "fdqxs6" failed to deploy
```

Para finalizar la instalación del controlador, se tiene que crear un modelo donde se alojarán las aplicaciones *OpenStack*, el comando que crea dicho modelo es el siguiente:

```
juju add-model --config default-series=focal openstack
```

Para comprobar que se ha creado el modelo de manera correcta la salida del comando *juju status* debe ser la siguiente:

```
Model      Controller       Cloud/Region    Version  SLA          Timestamp
openstack  maas-controller  mymaas/default  2.9.0    unsupported  01:51:00Z

Model "admin/openstack" is empty
```

En caso de querer redesplegar de nuevo *Openstack* habría que empezar desde este punto borrando el modelo desplegado con el comando *juju destroy-model*.

8. Openstack

8.1. ¿Qué es Openstack?

Según la siguiente referencia [1], en la actualidad, se dispone de una gran cantidad de herramientas *software* que dan servicios relacionados con el *Cloud Computing*. Muchas de estas herramientas, hay que pagar por ellas, pero en este caso, la herramienta en cuestión es de licencia libre. En un inicio, *OpenStack* fue un proyecto conjunto de dos empresas, la *NASA y Rackspace Hosting*. El objetivo de ambas, fue crear una herramienta que fuera capaz de implementar tanto nubes privadas, como públicas.

Figura 27: Logo del *Software OpenStack*

OpenStack es una herramienta que es capaz de proveer imágenes, contenedores, volúmenes, máquinas virtuales, *snapshots* entre otros recursos... La popularidad de la herramienta fue creciendo gracias a que algunas universidades la usan para dar servicios en la nube. Esto ha incrementado con la pandemia mundial que se ha vivido, ya que muchas universidades necesitaban el servicio para la enseñanza en la modalidad *on-line*.

La integración de este *software* en la actualidad es baja aunque cada vez más clientes adoptan esta tecnología para crear sus nubes públicas o privadas. Se prevé que más empresas empiecen a utilizar esta herramienta ya que está en continuo desarrollo y cada vez se vuelve más atractiva para un ecosistema *Cloud Computing*

8.2. Opciones de despliegue según la arquitectura

En el siguiente proyecto de fin grado [9] se propone una determinada estructura de red pero se tienen un sin fin de opciones para desplegar nuestro modelo *OpenStack*. Como punto de partida, se va a elegir un modelo con cuatros nodos de cómputo y un controlador *JuJu*.

Se ha estudiado y se ha podido comprobar, que se puede reducir el número de nodos de cómputo. Se puede proponer un modelo que disponga de tres, dos o incluso un único nodo de cómputo.

Como es lógico, reducir el número de nodos conlleva con ello, una pérdida de recursos. La nube estará más saturada. Se dispondrá de menos recursos y no se podrá dar servicio a una gran cantidad de clientes.

8.3. Instalación del modelo Openstack

Para la instalación de la herramienta de software libre *Openstack (versión Wallaby)* se ha seguido la guía oficial de la página de *Openstack* [5].

Otro aspecto a resaltar en este punto, es la organización y distribución de las aplicaciones en los nodos de cómputo. Se realiza una repartición de las aplicaciones de forma que todos los nodos de cómputo poseen el mismo número de unidades de aplicaciones, es decir, si se tiene una aplicación que está dividida en tres unidades y se hace un despliegue con tres nodos de cómputo, cada unidad de dicha aplicación se situaría en un contenedor de cada nodo de cómputo.

En el apartado anterior se instaló un controlador *JuJu* que se va a usar para proceder con la instalación de las aplicaciones. Se estudian dos maneras de instalar la herramienta:

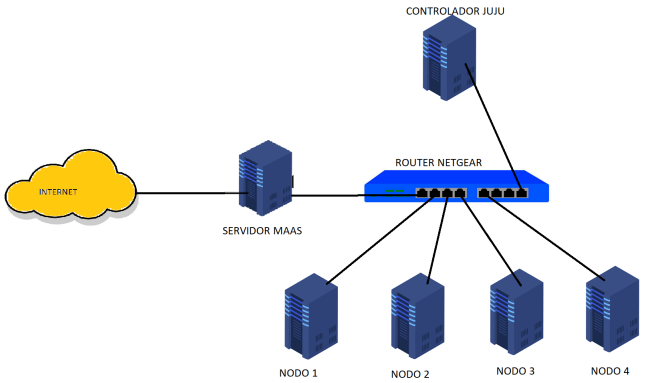


Figura 28: Arquitectura por defecto

- *By individual Charm*: Este método nos proporciona una visión de la instalación paso a paso. Nos da una visión clara de como funciona el controlador *JuJu* y de como se integran las aplicaciones en los nodos de cómputo.
- *By charm bundle*: Este método es completamente automático. Consiste en la creación de un fichero *yaml* que despliega las aplicaciones de manera automática.

En nuestro trabajo se ha elegido la primera opción. Se elige esta opción porque la opción de despliegue automático falla en el despliegue de las aplicaciones. Llega un momento en el proceso de instalación que algunas de las aplicaciones quedan bloqueadas. Es por eso que se elige la primera opción, que tiene la ventaja de controlar más el despliegue de todas las aplicaciones.

Se monitoriza el progreso de la instalación en un terminal a parte de la máquina *MaaS* con el comando:

```
watch -n 5 -c juju status --color
```

Este comando nos devuelve el progreso de la instalación cada 5 segundos. En el apartado anterior, se han instalado un controlador de *JuJu* llamado *maas-controller* y un modelo llamado *Openstack*. Para cambiar a este contexto creado anteriormente se ejecuta en la máquina *MaaS*:

```
juju switch maas-controller:openstack
```

Una vez se está en el contexto creado en el apartado anterior se procede a instalar la primera aplicación:

8.3.1. Ceph OSD:

Ceph es una aplicación que gestiona ficheros de almacenamiento distribuido en red. OSD son las siglas de *Object Storage Daemon*, traducido al castellano demonio de almacenaje de objetos. *Ceph* es reponsable de almacenar objetos para el sistema de archivos distribuido de la misma aplicación. Una vez que el servicio *Ceph-OSD* haya arrancado el clúster buscará los dispositivos de almacenamiento configurados y los agregará al grupo de almacenamiento disponible. Esta aplicación se despliega en los cuatro nodos de cómputo. El archivo "osd-ceph.yaml" contiene la configuración de esta aplicación:

```
ceph-osd:
  osd-devices: /dev/sdb
  source: cloud:focal-wallaby
```

Este fichero de configuración, nos indica el directorio donde se va a desplegar la aplicación y la fuente dónde están los ficheros de despliegue de la aplicación. Cabe resaltar, que se ejecutará el siguiente comando *juju deploy* desde el mismo directorio que se guarda el archivo *osd-ceph.yaml* (para todas las aplicaciones es igual).

Para que reconozca los nodos de cómputo donde se tiene que desplegar la aplicación, se utiliza la etiqueta "compute". Esta etiqueta ha sido creada en el momento que se han comisionado los nodos de cómputo. El comando que se utiliza es:

```
juju deploy -n 4 --config ceph-osd.yaml --constraints tags=compute ceph-osd
```

Si se observa el comando anterior se puede explicar algunas etiquetas que se usarán en comandos posteriores. La etiqueta *-n* indica el número de máquinas en las que se va a desplegar la aplicación, *–config* nos indica el fichero de configuración de dicha aplicación y *–constrains tags=compute* nos indica que las máquinas con esa etiqueta serán las elegidas para el despliegue de esta aplicación.

Figura 29: Logo de *Ceph*

En este momento las máquinas etiquetadas con el nombre *compute*, pasan del estado *Ready* a *Deploying* y como en el caso del controlador *JuJu*, se tienen que encender las máquinas, que arrancarán por red (*PXE*).

8.3.2. Nova Compute:

Esta aplicación se encarga del cómputo de la nube (gestiona las máquinas vituales). Es la parte fundamental de un sistema *IaaS*. Esta diseñado para automatizar y gestionar los *pools* de los equipos que se utilizan como nodos de cómputo. Contiene en su lógica el servicio de hipervisión y se implementa directamente en los nodos físicos de cómputo. Esta aplicación se despliega en tres máquinas. La configuración de la misma queda recogida en el archivo "nova-compute.yaml" cuyo contenido es el siguiente:

```
nova-compute:
  config-flags: default_ephemeral_format=ext4
  enable-live-migration: true
  enable-resize: true
  migration-auth-type: ssh
  openstack-origin: cloud:focal-wallaby
```

Cabe resaltar en el archivo de configuración, que se pone en *true* el flag *enable-resize* por si en el futuro hay que añadir una unidad de cómputo más. El comando que despliega esta aplicación es el siguiente:

```
juju deploy -n 3 --to 1,2,3 --config nova-compute.yaml nova-compute
```

Con la etiqueta *–to* se indica donde se va a desplegar la aplicación.

8.3.3. MySql Innodb cluster:

MySql Innodb cluster es una solución de alta fiabilidad proporcionada por *MySql* que dispone de varios agentes (en este caso 3) que se encargan de la detección de errores de conmutación. Se trata de varios servidores que controlan todas las bases de datos del sistema. Esta aplicación requiere de al menos 3 unidades

Figura 30: Logo de *Nova*

para que funcione de forma correcta. En nuestro caso se han elegido contenedores de tres nodos de cómputo diferentes.

```
juju deploy -n 3 --to lxd:0,lxd:1,lxd:2 mysql-innodb-cluster
```

Figura 31: Logo de *MySQL*

8.3.4. Vault

Vault es necesario para administrar los certificados TLS que permitirán la comunicación cifrada entre aplicaciones en la nube. También se encarga de proporcionar un almacenamiento de los datos de la nube. Se colocará en un contenedor que esta situado en el tercer nodo de cómputo.

```
juju deploy --to lxd:3 vault
```

Como se puede observar la etiqueta *–to* indica donde va a correr la aplicación, en este caso será en un contenedor del tercer nodo de cómputo.

Esta aplicación es la primera que se unirá a la base de datos que se desplegó en el apartado [8.3.3] (MySQL Innodb Cluster). Para ello añadimos las siguientes relaciones:

```
juju deploy mysql-router vault-mysql-router
juju add-relation vault-mysql-router:db-router mysql-innodb-cluster:db-router
juju add-relation vault-mysql-router:shared-db vault:shared-db
juju add-relation mysql-innodb-cluster:certificates vault:certificates
```

Figura 32: Logo del *Software OpenStack*

Una vez se han añadido estas relaciones, toca comprobar la salida del comando que nos está monitorizando el despliegue. La salida debe ser similar a esta:

```
Unit                     Workload  Agent  Machine  Public address  Ports     Message
ceph-osd/0*              blocked   idle   0        10.0.0.150                Missing relation: monitor
ceph-osd/1               blocked   idle   1        10.0.0.151                Missing relation: monitor
ceph-osd/2               blocked   idle   2        10.0.0.152                Missing relation: monitor
ceph-osd/3               blocked   idle   3        10.0.0.153                Missing relation: monitor
mysql-innodb-cluster/0*  active    idle   0/lxd/0  10.0.0.154                Unit is ready: Mode: R/W, Cluster is ONLINE and can tolerate up to ONE failure.
mysql-innodb-cluster/1   active    idle   1/lxd/0  10.0.0.155                Unit is ready: Mode: R/O, Cluster is ONLINE and can tolerate up to ONE failure.
mysql-innodb-cluster/2   active    idle   2/lxd/0  10.0.0.156                Unit is ready: Mode: R/O, Cluster is ONLINE and can tolerate up to ONE failure.
nova-compute/0*          blocked   idle   1        10.0.0.151                Missing relations: messaging, image
nova-compute/1           blocked   idle   2        10.0.0.152                Missing relations: messaging, image
nova-compute/2           blocked   idle   3        10.0.0.153                Missing relations: messaging, image
vault/0*                 active    idle   3/lxd/0  10.0.0.157      8200/tcp  Unit is ready (active: true, mlock: disabled)
  vault-mysql-router/0*  active    idle            10.0.0.157                Unit is ready
```

Como se puede observar, hay varias columnas importantes en la salida del comando *juju status*. La primera de ellas nos da información sobre el nombre de la aplicación, la segunda es el estado en el que se encuentra la aplicación (*active, blocked o error*), la tercera nos proporciona nociones de si está ejecutandose o inactiva (estado *executing o idle*), la cuarta es el nodo o contenedor en el que está corriendo, la quinta es la IP en la que está accesible (coincide con la IP del nodo/contenedor en el que se ejecute la aplicación), la sexta es el puerto y la última es un mensaje descripción de estado (esta última muy importante para detectar errores). El objetivo es que todas las apliaciones queden en un estado activo.

8.3.5. Neutron Networking:

La red de Neutron se despliega mediante 4 aplicaciones:

- Neutron-api: Esta aplicación da un servicio de redes virtuales para *OpenStack*. *Neutron* da una interfaz para solicitar y configurar dinámicamente redes virtuales. Estas redes llamadas SDN, son redes definidas por *Software*. Son las encargadas de conectar las interfaces virtuales de las instancias de *Nova*. La API de *Neutron* admite extensiones para proporcionar en sus redes calidad de servicio (QoS) o monitereo de las mismas.
- Neutron-api-plugin-ovn (subordinate): Es la aplicación que se encarga de que *OVN (Open Virtual Network)* y *Neutron* se comuniquen entre ellas.
- Ovn-central(*Open Virtual Network Central*): Se encarga de orquestar las bases de datos de *ovn-chassis*.
- Ovn-chassis (subordinate): Esta aplicación es la encargada de asignar los puentes de red y puertos de datos. Con esta aplicación se gestionará la puerta de enlace de acceso a la instancias virtuales.

El fichero *neutron.yaml* contiene la configuración para 3 de ellas:

```
ovn-chassis:
  bridge-interface-mappings: br-ex:eth0.2
  ovn-bridge-mappings: physnet1:br-ex
neutron-api:
  neutron-security-groups: true
  flat-network-providers: physnet1
  worker-multiplier: 0.25
  openstack-origin: cloud:focal-wallaby
ovn-central:
  source: cloud:focal-wallaby
```

Este fichero de configuración es crítico, ya que en los parámetros: *bridge-interface-mappings y ovn-bridge-mappings* viene especificado a través de que red saldrán a internet las instancias. Para ello, hay que crear una red IP que se llama *provider*, más adelante se explica el comando *openstack network create provider*. En el comando anterior también se usa el parámetro *–provider-physical-network physnet1*, que quiere decir que la red física que se usa es la *physnet1*. En el fichero anterior se relaciona *physnet1* con el puente que usará, el br-ex (*ovn-bridge-mappings: physnet1:br-ex*). Por último, hay que indicar el interfaz de red que usará br-ex con la línea *bridge-interface-mappings: br-ex:eth0.2*. Resumiendo, la red IP *provider* usa *physnet1* (especificado en el comando *openstack network create*) que se implementa sobre el puente *br-ex* que usa un interfaz de red *ethx* (especificado en el neutron.yaml).

Como se puede observar, se utiliza un *bridge* que colgará de la *VLAN* eth0.2. Todas las instancias compartirán esta red para descargar cualquier archivo disponible en la red.

La aplicación *ovn-central* requiere de al menos 3 unidades según la documentación de *OpenStack* [5]. Se colocará en contenedores de las máquinas de cómputo 0, 1, 2. El comando es el siguiente:

```
juju deploy -n 3 --to lxd:0,lxd:1,lxd:2 --config neutron.yaml ovn-central
```

La aplicación de *neutron-api* se colocará en un contenedor de la máquina 1:

```
juju deploy --to lxd:1 --config neutron.yaml neutron-api
```

Ya solo nos quedan las otras dos aplicaciones:

```
juju deploy neutron-api-plugin-ovn
juju deploy --config neutron.yaml ovn-chassis
```

Añadimos las relaciones necesarias:

```
juju add-relation neutron-api-plugin-ovn:neutron-plugin
    neutron-api:neutron-plugin-api-subordinate
juju add-relation neutron-api-plugin-ovn:ovsdb-cms ovn-central:ovsdb-cms
juju add-relation ovn-chassis:ovsdb ovn-central:ovsdb
juju add-relation ovn-chassis:nova-compute nova-compute:neutron-plugin
juju add-relation neutron-api:certificates vault:certificates
juju add-relation neutron-api-plugin-ovn:certificates vault:certificates
juju add-relation ovn-central:certificates vault:certificates
juju add-relation ovn-chassis:certificates vault:certificates
```

Ahora se despliega la base de datos correspondiente a este apartado y se añaden las correspondientes relaciones:

```
juju deploy mysql-router neutron-api-mysql-router
juju add-relation neutron-api-mysql-router:db-router mysql-innodb-cluster:db-router
juju add-relation neutron-api-mysql-router:shared-db neutron-api:shared-db
```

Figura 33: Logo de *Neutron*

8.3.6. Keystone

Esta aplicación es la que proporciona servicio de autenticación de clientes que consumen recursos de la nube. Implementa la API de identidad de *OpenStack*. Esta aplicación se almacenará en un contenedor de la máquina 0. El archivo *keystone.yaml* tiene la siguiente configuración:

```
keystone:
  worker-multiplier: 0.25
  openstack-origin: cloud:focal-wallaby
```

Cabe resaltar, que el apartado de configuración *worker-multiplier* indica el numero de *trabajadores* que se van utilizar cuando esta aplicación demande CPU al nodo en el que se despliega. El comando para desplegar es:

```
juju deploy --to lxd:0 --config keystone.yaml keystone
```

Añadimos la base de datos necesaria para esta aplicación:

```
juju deploy mysql-router keystone-mysql-router
juju add-relation keystone-mysql-router:db-router mysql-innodb-cluster:db-router
juju add-relation keystone-mysql-router:shared-db keystone:shared-db
```

Se tienen que añadir las siguientes relaciones:

```
juju add-relation keystone:identity-service neutron-api:identity-service
juju add-relation keystone:certificates vault:certificates
```

Figura 34: Logo del *Keystone*

8.3.7. RabbitMQ

Esta aplicación es un servicio AMQP (*Advanced Message Queuing Protocol*). Esta aplicación es un servicio de mensajería entre aplicaciones del ecosistema *Cloud*. En un contenedor de la máquina 2 estará presente esta aplicación. El comando que despliega es:

```
juju deploy --to lxd:2 rabbitmq-server
```

Se agregan las siguientes relaciones:

```
juju add-relation rabbitmq-server:amqp neutron-api:amqp
juju add-relation rabbitmq-server:amqp nova-compute:amqp
```

Figura 35: Logo de *RabbitMQ*

En este momento la salida que se ve mediante “juju status” es la siguiente:

```
Unit                          Workload  Agent  Machine  Public address  Ports             Message
ceph-osd/0*                   blocked   idle   0        10.0.0.150                        Missing relation: monitor
ceph-osd/1                    blocked   idle   1        10.0.0.151                        Missing relation: monitor
ceph-osd/2                    blocked   idle   2        10.0.0.152                        Missing relation: monitor
ceph-osd/3                    blocked   idle   3        10.0.0.153                        Missing relation: monitor
keystone/0*                   active    idle   0/lxd/2  10.0.0.162      5000/tcp          Unit is ready
  keystone-mysql-router/0*    active    idle            10.0.0.162                        Unit is ready
mysql-innodb-cluster/0*       active    idle   0/lxd/0  10.0.0.154                        Unit is ready: Mode: R/W, Cluster is ONLINE and can tolerate up to ONE failure.
mysql-innodb-cluster/1        active    idle   1/lxd/0  10.0.0.155                        Unit is ready: Mode: R/O, Cluster is ONLINE and can tolerate up to ONE failure.
mysql-innodb-cluster/2        active    idle   2/lxd/0  10.0.0.156                        Unit is ready: Mode: R/O, Cluster is ONLINE and can tolerate up to ONE failure.
neutron-api/0*                active    idle   1/lxd/2  10.0.0.161      9696/tcp          Unit is ready
  neutron-api-mysql-router/0* active    idle            10.0.0.161                        Unit is ready
  neutron-api-plugin-ovn/0*   active    idle            10.0.0.161                        Unit is ready
nova-compute/0*               blocked   idle   1        10.0.0.151                        Missing relations: image
  ovn-chassis/2               active    idle            10.0.0.151                        Unit is ready
nova-compute/1                blocked   idle   2        10.0.0.152                        Missing relations: image
  ovn-chassis/0*              active    idle            10.0.0.152                        Unit is ready
nova-compute/2                blocked   idle   3        10.0.0.153                        Missing relations: image
  ovn-chassis/1               active    idle            10.0.0.153                        Unit is ready
ovn-central/0*                active    idle   0/lxd/1  10.0.0.158      6641/tcp,6642/tcp  Unit is ready (leader: ovnnb_db, ovnsb_db northd: active)
ovn-central/1                 active    idle   1/lxd/1  10.0.0.159      6641/tcp,6642/tcp  Unit is ready
ovn-central/2                 active    idle   2/lxd/1  10.0.0.160      6641/tcp,6642/tcp  Unit is ready
rabbitmq-server/0*            active    idle   2/lxd/2  10.0.0.163      5672/tcp          Unit is ready
vault/0*                      active    idle   3/lxd/0  10.0.0.157      8200/tcp          Unit is ready (active: true, mlock: disabled)
  vault-mysql-router/0*       active    idle            10.0.0.157                        Unit is ready
```

8.3.8. Nova Cloud Controller

Esta aplicación es donde reside el control y acceso de los recursos de virtualización. Es un controlador de computación de *OpenStack*. Es compatible con la API de Amazon además de la que se está tratando (*OpenStack*). Esta aplicacion contiene servicios del tipo nova-scheduler, nova-api y nova-conductor. Estará almacenada en un contenedor de la máquina 0. El archivo *nova-cloud-controller.yaml* contiene lo siguiente:

```
nova-cloud-controller:
  network-manager: Neutron
  worker-multiplier: 0.25
  openstack-origin: cloud:focal-wallaby
```

Para desplegar la aplicación se utiliza:

```
juju deploy --to lxd:3 --config nova-cloud-controller.yaml nova-cloud-controller
```

La base de datos de esta aplicación se despliega con el siguiente comando:

```
juju deploy mysql-router ncc-mysql-router
juju add-relation ncc-mysql-router:db-router mysql-innodb-cluster:db-router
juju add-relation ncc-mysql-router:shared-db nova-cloud-controller:shared-db
```

Se agregan las relaciones necesarias:

```
juju add-relation nova-cloud-controller:identity-service keystone:identity-service
juju add-relation nova-cloud-controller:amqp rabbitmq-server:amqp
juju add-relation nova-cloud-controller:neutron-api neutron-api:neutron-api
juju add-relation nova-cloud-controller:cloud-compute nova-compute:cloud-compute
juju add-relation nova-cloud-controller:certificates vault:certificates
```

8.3.9. Placement:

Proporciona un servicio para administrar recursos de las diferentes aplicaciones como por ejemplo, memoria RAM, almacenamiento o procesadores. Esta aplicación da a conocer los recursos que están libres en una nube. Estará ubicada en un contenedor de la máquina tres. El archivo *placement.yaml* contiene lo siguiente:

```
placement:
  worker-multiplier: 0.25
  openstack-origin: cloud:focal-wallaby
```

Para desplegar se utiliza el siguiente comando:

```
juju deploy --to lxd:3 --config placement.yaml placement
```

Ahora se despliega la base de datos relacionada con esta aplicación y se añaden sus respectivas relaciones:

```
juju deploy mysql-router placement-mysql-router
juju add-relation placement-mysql-router:db-router mysql-innodb-cluster:db-router
juju add-relation placement-mysql-router:shared-db placement:shared-db
```

Se agregan tres relaciones más:

```
juju add-relation placement:identity-service keystone:identity-service
juju add-relation placement:placement nova-cloud-controller:placement
juju add-relation placement:certificates vault:certificates
```

Figura 36: Logo de *placement*

8.3.10. Openstack Dashboard:

OpenStack Dashboard (API *Horizon*) proporciona una interfaz web, basada en Django, con funciones completas para que la usen tanto los administradores como los usuarios de OpenStack y puedan interactuar con instancias (Nova), imágenes (Glance), bloques de almacenamiento (Cinder) y redes (Neutron) dentro de una implementación de un modelo *OpenStack*. Esta aplicación está situada en un contenedor de la máquina 2. El comando de despliegue es:

```
juju deploy --to lxd:2 --config openstack-origin=cloud:focal-wallaby
    openstack-dashboard
```

La base de datos relacionada con esta aplicación y las relaciones necesarias se despliegan con los comandos:

```
juju deploy mysql-router dashboard-mysql-router
juju add-relation dashboard-mysql-router:db-router mysql-innodb-cluster:db-router
juju add-relation dashboard-mysql-router:shared-db openstack-dashboard:shared-db
```

Se requieren dos relaciones más:

```
juju add-relation openstack-dashboard:identity-service keystone:identity-service
juju add-relation openstack-dashboard:certificates vault:certificates
```

8.3.11. Glance

Glance es la aplicación encarga del suministro de las imágenes de sistema operativo para los elementos de virtualización tales como instancias, contenedores... Se aprovecha del servicio de almacenamiento que proporciona *Ceph* para guardar las imágenes. El que más aprovecha el servicio que da esta aplicación es *Nova* ya que es la controla toda la lógica para la creación de las instacias virtuales. Esta aplicación estará contenida en un contenedor situado en la máquina 3. El archivo *glance.yaml* contiene la configuración de la misma:

```
glance:
  worker-multiplier: 0.25
  openstack-origin: cloud:focal-wallaby
```

Para desplegar, se ejecuta en el terminal de la máquina *MaaS*:

```
juju deploy --to lxd:3 --config glance.yaml glance
```

Se le añade la base de datos que es necesaria para el correcto funcionamiento de la misma, con sus pertinentes relaciones:

```
juju deploy mysql-router glance-mysql-router
juju add-relation glance-mysql-router:db-router mysql-innodb-cluster:db-router
juju add-relation glance-mysql-router:shared-db glance:shared-db
```

Se agregan las siguientes relaciones:

```
juju add-relation glance:image-service nova-cloud-controller:image-service
juju add-relation glance:image-service nova-compute:image-service
juju add-relation glance:identity-service keystone:identity-service
juju add-relation glance:certificates vault:certificates
```

Figura 37: Logo de *Glance*

En este momento la salida del comando "juju status" es la siguiente:

```
Unit                          Workload  Agent  Machine  Public address  Ports             Message
ceph-osd/0*                   blocked   idle   0        10.0.0.150                        Missing relation: monitor
ceph-osd/1                    blocked   idle   1        10.0.0.151                        Missing relation: monitor
ceph-osd/2                    blocked   idle   2        10.0.0.152                        Missing relation: monitor
ceph-osd/3                    blocked   idle   3        10.0.0.153                        Missing relation: monitor
glance/0*                     active    idle   3/lxd/3  10.0.0.167      9292/tcp          Unit is ready
  glance-mysql-router/0*      active    idle            10.0.0.167                        Unit is ready
keystone/0*                   active    idle   0/lxd/2  10.0.0.162      5000/tcp          Unit is ready
  keystone-mysql-router/0*    active    idle            10.0.0.162                        Unit is ready
mysql-innodb-cluster/0*       active    idle   0/lxd/0  10.0.0.154                        Unit is ready: Mode: R/W, Cluster is ONLINE and can tolerate up to ONE failure.
mysql-innodb-cluster/1        active    idle   1/lxd/0  10.0.0.155                        Unit is ready: Mode: R/O, Cluster is ONLINE and can tolerate up to ONE failure.
mysql-innodb-cluster/2        active    idle   2/lxd/0  10.0.0.156                        Unit is ready: Mode: R/O, Cluster is ONLINE and can tolerate up to ONE failure.
neutron-api/0*                active    idle   1/lxd/2  10.0.0.161      9696/tcp          Unit is ready
  neutron-api-mysql-router/0* active    idle            10.0.0.161                        Unit is ready
  neutron-api-plugin-ovn/0*   active    idle            10.0.0.161                        Unit is ready
nova-cloud-controller/0*      active    idle   3/lxd/1  10.0.0.164      8774/tcp,8775/tcp  Unit is ready
  ncc-mysql-router/0*         active    idle            10.0.0.164                        Unit is ready
nova-compute/0*               active    idle   1        10.0.0.151                        Unit is ready
  ovn-chassis/2               active    idle            10.0.0.151                        Unit is ready
```

```
nova-compute/1                active    idle   2        10.0.0.152                      Unit is ready
  ovn-chassis/0*              active    idle            10.0.0.152                    Unit is ready
nova-compute/2                active    idle   3        10.0.0.153                      Unit is ready
  ovn-chassis/1               active    idle            10.0.0.153                      Unit is ready
openstack-dashboard/0*        active    idle   2/lxd/3  10.0.0.166      80/tcp,443/tcp  Unit is ready
  dashboard-mysql-router/0*   active    idle            10.0.0.166                      Unit is ready
ovn-central/0*                active    idle   0/lxd/1  10.0.0.158      6641/tcp,6642/tcp  Unit is ready (leader: ovnnb_db, ovnsb_db northd: active)
ovn-central/1                 active    idle   1/lxd/1  10.0.0.159      6641/tcp,6642/tcp  Unit is ready
ovn-central/2                 active    idle   2/lxd/1  10.0.0.160      6641/tcp,6642/tcp  Unit is ready
placement/0*                  active    idle   3/lxd/2  10.0.0.165      8778/tcp        Unit is ready
  placement-mysql-router/0*   active    idle            10.0.0.165                      Unit is ready
rabbitmq-server/0*            active    idle   2/lxd/2  10.0.0.163      5672/tcp        Unit is ready
vault/0*                      active    idle   3/lxd/0  10.0.0.157      8200/tcp        Unit is ready (active: true, mlock: disabled)
  vault-mysql-router/0*       active    idle            10.0.0.157                      Unit is ready
```

8.3.12. Ceph-monitor

Ceph-mon se encarga de monitorizar el clúster del sistema de archivos distribuido. Esta aplicación está presente en contenedores de las máquinas cero, uno y dos. El fichero *ceph-mon.yaml* contiene la configuración de la misma:

```
ceph-mon:
  expected-osd-count: 3
  monitor-count: 3
  source: cloud:focal-wallaby
```

El comando que despliega esta aplicación es:

```
juju deploy -n 3 --to lxd:0,lxd:1,lxd:2 --config ceph-mon.yaml ceph-mon
```

Las relaciones necesarias para una buena práctica deben ser:

```
juju add-relation ceph-mon:osd ceph-osd:mon
juju add-relation ceph-mon:client nova-compute:ceph
juju add-relation ceph-mon:client glance:ceph
```

8.3.13. Cinder

Esta aplicación es la encargada de gestionar el almacenamiento por bloques. Proporciona volúmenes de almacenamiento a las máquinas virtuales y a los contenedores. Como usuario final, estos volúmenes se crean desde el *Dashboard de Horizon*, aunque también se podría utilizar el API REST de *Cinder*. *Cinder* está situado en un contenedor de la máquina uno. El archivo de configuración deberá de disponer de lo siguiente:

```
cinder:
  block-device: None
  glance-api-version: 2
  worker-multiplier: 0.25
  openstack-origin: cloud:focal-wallaby
```

Para desplegar la aplicación habrá que teclear en el terminal:

```
juju deploy --to lxd:1 --config cinder.yaml cinder
```

Añadimos la base de datos necesaria para el buen funcionamiento de la aplicación, con algunas relaciones:

```
juju deploy mysql-router cinder-mysql-router
juju add-relation cinder-mysql-router:db-router mysql-innodb-cluster:db-router
juju add-relation cinder-mysql-router:shared-db cinder:shared-db
```

Se añaden las siguientes relaciones:

```
juju add-relation cinder:cinder-volume-service
    nova-cloud-controller:cinder-volume-service
juju add-relation cinder:identity-service keystone:identity-service
juju add-relation cinder:amqp rabbitmq-server:amqp
juju add-relation cinder:image-service glance:image-service
juju add-relation cinder:certificates vault:certificates
```

Como método de almacenamiento esta aplicación usará "Cinder-Ceph" para su despliegue introducimos en el terminal:

```
juju deploy cinder-ceph
```

Nos falta añadir 3 relaciones más:

```
juju add-relation cinder-ceph:storage-backend cinder:storage-backend
juju add-relation cinder-ceph:ceph ceph-mon:client
juju add-relation cinder-ceph:ceph-access nova-compute:ceph-access
```

Figura 38: Logo del *Software OpenStack*

8.3.14. Ceph Radows Gateway

La aplicación que se utiliza como puerta de enlace *HTTP* y que es compatible con *Swift y S3* se almacenará en un contenedor de la máquina cero. El comando de instalación es el siguiente:

```
juju deploy --to lxd:0 --config source=cloud:focal-wallaby ceph-radosgw
```

La relación necesaria para el correcto funcionamiento es la siguiente:

```
juju add-relation ceph-radosgw:mon ceph-mon:radosgw
```

8.3.15. NTP

La aplicación que se encarga de la sincronización de la hora en cada nodo de la nube, se desplegará en los propios nodos físicos. El comando de despliegue es el siguiente:

```
juju deploy ntp
```

La siguiente relación es necesaria para una buena práctica:

```
juju add-relation ceph-osd:juju-info ntp:juju-info
```

Figura 39: Logo de *NTP*

8.4. Automatización del despliegue de las aplicaciones *Openstack*

Como se puede observar, el proceso seguido para la instalación de la nube no es automático y requiere de alguien que este lanzando uno a uno los comandos expuestos anteriormente. Debido a esto, se propone un despliegue en cuatro fases.

El despliegue se divide en cuatro fases, porque las aplicaciones que componen la nube tienen relaciones unas con otras y sin estas relaciones no avanzaría el proceso de despliegue. Además, la aplicación *vault*, hay que realizar varias configuraciones a mano, por lo tanto requiere de un parón en el despliegue de las aplicaciones.

Como se ha podido observar en la instalación del proceso, hay cuatro *puntos de control* en los cuales las aplicaciones tienen que estar en el estado indicado si no fallaría el proceso de instalación.

8.4.1. Primera fase

Con los nodos cómputo apagados se ejecutan los *scripts* con el intérprete *Shell Script* de linux:

```
sh script1.sh
```

Una vez se haya ejecutado el *script*, se tienen que arrancar las máquinas para que el proceso siga su curso y las aplicaciones se empiecen a instalar de manera correcta.

El *script* que despliega las aplicaciones de esta primera fase es el siguiente:

```
#!/bin/sh
juju add-model --config default-series=focal openstack
juju model-config default-space=admin
juju deploy -n 4 --config ceph-osd.yaml --constraints tags=compute ceph-osd
juju deploy -n 3 --to 1,2,3 --config nova-compute.yaml nova-compute
juju deploy -n 3 --to lxd:0,lxd:1,lxd:2 mysql-innodb-cluster
juju deploy --to lxd:3 vault
juju deploy mysql-router vault-mysql-router
juju add-relation vault-mysql-router:db-router mysql-innodb-cluster:db-router
juju add-relation vault-mysql-router:shared-db vault:shared-db
juju add-relation mysql-innodb-cluster:certificates vault:certificates
```

Se monitoriza el progreso de la instalación en un terminal a parte de la máquina *MaaS* con el comando:

```
watch -n 5 -c juju status --color
```

La aplicación *Vault* no se despliega automáticamente y hay que lanzar unos comandos en el terminal para desbloquearla. Cabe resaltar, que para realizar esta configuración es necesario que el comando *juju status* muestre en la columna *Messages* el mensaje: *Vault needs to be initilized*, puede tardar unas dos horas. En este punto, los comandos que hay que ejecutar son los siguientes:

```
sudo snap install vault
export VAULT_ADDR="http://IP-CONTENEDOR-VAULT:8200"
```

Se sustituye *IP-CONTENDOR-VAULT* por la IP en la que corre la aplicación *Vault*. Se inicia la aplicación con el siguiente comando:

```
vault operator init -key-shares=5 -key-threshold=3 > vault.txt
```

Como se puede observar, se guarda la salida del comando en un archivo de texto ya que la información que devuelve es útil para desbloquear la aplicación. La salida del comando es algo parecido a esto:

```
Unseal Key 1: bgpuJuViLVTiPoRoY8XbYoZ/UQchsIbEx0dUfpOdIHzI
Unseal Key 2: tQebrrFzm6cU8UNmuPU70urGyjlq6PlWnohcfIHRbEDr
Unseal Key 3: mRELp7kEWQa7UgJ4fHCS7vldR1B6gNOdRpVlBF1FozNY

Initial Root Token: s.rnafhsxqcpFNJkq061bpSNbM

Vault initialized with 5 key shares and a key threshold of 3. Please securely
distribute the key shares printed above. When the Vault is re-sealed,
restarted, or stopped, you must supply at least 3 of these keys to unseal it
before it can start servicing requests.

Vault does not store the generated master key. Without at least 3 key to
reconstruct the master key, Vault will remain permanently sealed!

It is possible to generate new unseal keys, provided you have a quorum of
existing unseal keys shares. See "vault operator rekey" for more information.
```

Ahora se utilizan las llaves anteriores para desbloquear la aplicación. El comando utilizado es el siguiente:

```
vault operator unseal bgpuJuViLVTiPoRoY8XbYoZ/UQchsIbEx0dUfpOdIHzI
vault operator unseal tQebrrFzm6cU8UNmuPU70urGyjlq6PlWnohcfIHRbED
vault operator unseal mRELp7kEWQa7UgJ4fHCS7vldR1B6gNOdRpVlBF1FozNY
```

Se cambia la variable de entorno *VAULT TOKEN* con el siguiente *Token* recogido de la salida del comando anterior:

```
export VAULT_TOKEN=s.rnafhsxqcpFNJkq061bpSNbM
```

Se crea un *Token* nuevo con:

```
vault token create -ttl=10m > vault1.txt
```

Al crear el *Token* anterior se guarda la salida en un fichero, ya que es imprescindible para el siguiente comando. La salida es algo parecido a esto:

```
Key                  Value
---                  -----
token                s.QMhaOED3UGQ4MeH3fmGOpNED
token_accessor       nApB972Dp2lnTTIF5VXQqnnb
token_duration       10m
token_renewable      true
token_policies       ["root"]
identity_policies    []
policies             ["root"]
```

Se usa el controlador *JuJu* para iniciar la aplicación:

```
juju run-action --wait vault/leader authorize-charm token=s.QMhaOED3UGQ4MeH3fmGOpNED
```

Por último, se vuelve a utilizar el controlador *JuJu* para poner activa esta aplicación:

```
juju run-action --wait vault/leader generate-root-ca
```

Hasta que las aplicaciones no lleguen al estado que se muestra en el siguiente cuadro, puede tardar una media hora, no será posible ejecutar el siguiente *script*. También puede ocurrir que alguna unidad *mysql-innodb-cluster* esté en estado de *blocked*, aun así seguir con el proceso:

```
Unit                    Workload  Agent  Machine  Public address  Ports     Message
ceph-osd/0*             blocked   idle   0        10.0.0.150                Missing relation: monitor
ceph-osd/1              blocked   idle   1        10.0.0.151                Missing relation: monitor
ceph-osd/2              blocked   idle   2        10.0.0.152                Missing relation: monitor
ceph-osd/3              blocked   idle   3        10.0.0.153                Missing relation: monitor
mysql-innodb-cluster/0* active    idle   0/lxd/0  10.0.0.154                Unit is ready: Mode: R/W, Cluster is ONLINE and can tolerate up to ONE failure.
mysql-innodb-cluster/1  active    idle   1/lxd/0  10.0.0.155                Unit is ready: Mode: R/O, Cluster is ONLINE and can tolerate up to ONE failure.
mysql-innodb-cluster/2  active    idle   2/lxd/0  10.0.0.156                Unit is ready: Mode: R/O, Cluster is ONLINE and can tolerate up to ONE failure.
nova-compute/0*         blocked   idle   1        10.0.0.151                Missing relations: messaging, image
nova-compute/1          blocked   idle   2        10.0.0.152                Missing relations: messaging, image
nova-compute/2          blocked   idle   3        10.0.0.153                Missing relations: messaging, image
vault/0*                active    idle   3/lxd/0  10.0.0.157      8200/tcp  Unit is ready (active: true, mlock: disabled)
  vault-mysql-router/0* active    idle            10.0.0.157                Unit is ready
```

En esta primera fase se puede producir un *bug*. Este punto crítico del despliegue, depende de la versión de *MaaS y JuJu* que se ha instalado. Se suele producir en versiones de *MaaS* superiores a la 3.0.0. Este fallo provoca que la creación de los contenedores no sea exitosa y se produce una fase de bloqueo, ya que se quedarían en el estado *Pending* (con la salida del comando *juju status* se puede observar este suceso). Se produce porque los contenedores del tipo *LXD* tienen la versión 5.0/stable por defecto y el sistema no está preparado para ello. Como solución se propone bajar a la versión 4.24 estable. Con el siguiente comando *JuJu* conseguimos corregir el *bug* [2]:

```
juju model-config lxd-snap-channel=4.24/stable
```

Con las nuevas actualizaciones de la aplicación *mysql-innodb-cluster*, que es la encargada de levantar las bases de datos que acompañan a las aplicaciones, se puede producir un estado *blocked* durante el despliegue. La primera base de datos que se levanta es la de *vault*, por tanto se explicará como solucionar este problema en este momento del despliegue. Con las demás bases de datos siguientes se soluciona de manera análoga. El mensaje de fallo que devuelve el comando *juju status* es *Failed to connect to MySQL*. El *bug* viene recogido en el siguiente enlace [3].

Para resolver el error, hay que entrar por *SSH* a la máquina donde está situada la base de datos y entrar al siguiente fichero de configuración: */var/lib/mysql/vault-mysql-router/mysqlrouter.conf*. Se cambian las siguientes líneas del fichero en cuestión.

```
[metadata_cache:bootstrap] --> [metadata_cache:jujuCluster]
[routing:bootstrap_rw] --> [routing:jujuCluster_rw]
[routing:bootstrap_ro] --> [routing:jujuCluster_ro]
[routing:bootstrap_x_rw] --> [routing:jujuCluster_x_rw]
[routing:bootsrap_x_ro] --> [routing:jujuCluster_x_ro]
borrar la línea --> [metadata_cache:jujuCluster]
```

En este momento se reinicia la máquina donde corre la base de datos con el comando *sudo reboot* y seguirá el despliegue su curso.

Otro problema que puede suceder es que alguna réplica de la aplicación *Ceph-osd* quede bloqueada, no es crítico, en la sección de reparación de la nube ante cortes de luz y fallos se explica como se puede solucionar.

8.4.2. Segunda fase

Para iniciar la segunda fase, se ejecuta:

```
sh script2.sh
```

El *script* que despliega las aplicaciones de esta segunda fase tiene el siguiente aspecto:

```
juju deploy -n 3 --to lxd:0,lxd:1,lxd:2 --config neutron.yaml ovn-central
juju deploy --to lxd:1 --config neutron.yaml neutron-api
juju deploy neutron-api-plugin-ovn
juju deploy --config neutron.yaml ovn-chassis
juju add-relation neutron-api-plugin-ovn:neutron-plugin
    neutron-api:neutron-plugin-api-subordinate
juju add-relation neutron-api-plugin-ovn:ovsdb-cms ovn-central:ovsdb-cms
juju add-relation ovn-chassis:ovsdb ovn-central:ovsdb
juju add-relation ovn-chassis:nova-compute nova-compute:neutron-plugin
juju add-relation neutron-api:certificates vault:certificates
juju add-relation neutron-api-plugin-ovn:certificates vault:certificates
juju add-relation ovn-central:certificates vault:certificates
juju add-relation ovn-chassis:certificates vault:certificates
juju deploy mysql-router neutron-api-mysql-router
juju add-relation neutron-api-mysql-router:db-router mysql-innodb-cluster:db-router
juju add-relation neutron-api-mysql-router:shared-db neutron-api:shared-db
juju deploy --to lxd:0 --config keystone.yaml keystone
juju deploy mysql-router keystone-mysql-router
juju add-relation keystone-mysql-router:db-router mysql-innodb-cluster:db-router
juju add-relation keystone-mysql-router:shared-db keystone:shared-db
juju add-relation keystone:identity-service neutron-api:identity-service
juju add-relation keystone:certificates vault:certificates
juju deploy --to lxd:2 rabbitmq-server
juju add-relation rabbitmq-server:amqp neutron-api:amqp
juju add-relation rabbitmq-server:amqp nova-compute:amqp
```

Para continuar con el despliegue las aplicaciones tienen que estar en el siguiente estado, todo activo menos ceph-osd y nova-compute que están bloqueados. Suele tardar una hora y media:

```
Unit                          Workload  Agent  Machine  Public address  Ports              Message
ceph-osd/0*                   blocked   idle   0        10.0.0.150                         Missing relation: monitor
ceph-osd/1                    blocked   idle   1        10.0.0.151                         Missing relation: monitor
ceph-osd/2                    blocked   idle   2        10.0.0.152                         Missing relation: monitor
ceph-osd/3                    blocked   idle   3        10.0.0.153                         Missing relation: monitor
keystone/0*                   active    idle   0/lxd/2  10.0.0.162      5000/tcp           Unit is ready
  keystone-mysql-router/0*    active    idle            10.0.0.162                         Unit is ready
mysql-innodb-cluster/0*       active    idle   0/lxd/0  10.0.0.154                         Unit is ready: Mode: R/W, Cluster is ONLINE and can tolerate up to ONE failure.
mysql-innodb-cluster/1        active    idle   1/lxd/0  10.0.0.155                         Unit is ready: Mode: R/O, Cluster is ONLINE and can tolerate up to ONE failure.
mysql-innodb-cluster/2        active    idle   2/lxd/0  10.0.0.156                         Unit is ready: Mode: R/O, Cluster is ONLINE and can tolerate up to ONE failure.
neutron-api/0*                active    idle   1/lxd/2  10.0.0.161      9696/tcp           Unit is ready
  neutron-api-mysql-router/0* active    idle            10.0.0.161                         Unit is ready
  neutron-api-plugin-ovn/0*   active    idle            10.0.0.161                         Unit is ready
nova-compute/0*               blocked   idle   1        10.0.0.151                         Missing relations: image
  ovn-chassis/2               active    idle            10.0.0.151                         Unit is ready
nova-compute/1                blocked   idle   2        10.0.0.152                         Missing relations: image
  ovn-chassis/0*              active    idle            10.0.0.152                         Unit is ready
nova-compute/2                blocked   idle   3        10.0.0.153                         Missing relations: image
  ovn-chassis/1               active    idle            10.0.0.153                         Unit is ready
ovn-central/0*                active    idle   0/lxd/1  10.0.0.158      6641/tcp,6642/tcp  Unit is ready (leader: ovnnb_db, ovnsb_db northd: active)
ovn-central/1                 active    idle   1/lxd/1  10.0.0.159      6641/tcp,6642/tcp  Unit is ready
ovn-central/2                 active    idle   2/lxd/1  10.0.0.160      6641/tcp,6642/tcp  Unit is ready
rabbitmq-server/0*            active    idle   2/lxd/2  10.0.0.163      5672/tcp           Unit is ready
vault/0*                      active    idle   3/lxd/0  10.0.0.157      8200/tcp           Unit is ready (active: true, mlock: disabled)
  vault-mysql-router/0*       active    idle            10.0.0.157                         Unit is ready
```

8.4.3. Tercera fase

El comando que inicia esta fase es:

```
sh script3.sh
```

El *script* que despliega esta tercera fase de aplicaciones es el siguiente:

```
#!/bin/sh
juju deploy --to lxd:3 --config nova-cloud-controller.yaml nova-cloud-controller
juju deploy mysql-router ncc-mysql-router
juju add-relation ncc-mysql-router:db-router mysql-innodb-cluster:db-router
juju add-relation ncc-mysql-router:shared-db nova-cloud-controller:shared-db
juju add-relation nova-cloud-controller:identity-service keystone:identity-service
juju add-relation nova-cloud-controller:amqp rabbitmq-server:amqp
juju add-relation nova-cloud-controller:neutron-api neutron-api:neutron-api
juju add-relation nova-cloud-controller:cloud-compute nova-compute:cloud-compute
juju add-relation nova-cloud-controller:certificates vault:certificates
juju deploy --to lxd:3 --config placement.yaml placement
juju deploy mysql-router placement-mysql-router
juju add-relation placement-mysql-router:db-router mysql-innodb-cluster:db-router
juju add-relation placement-mysql-router:shared-db placement:shared-db
juju add-relation placement:identity-service keystone:identity-service
juju add-relation placement:placement nova-cloud-controller:placement
juju add-relation placement:certificates vault:certificates
juju deploy --to lxd:2 --config openstack-origin=cloud:focal-wallaby
    openstack-dashboard
juju deploy mysql-router dashboard-mysql-router
juju add-relation dashboard-mysql-router:db-router mysql-innodb-cluster:db-router
juju add-relation dashboard-mysql-router:shared-db openstack-dashboard:shared-db
juju add-relation openstack-dashboard:identity-service keystone:identity-service
juju add-relation openstack-dashboard:certificates vault:certificates
juju deploy --to lxd:3 --config glance.yaml glance
juju deploy mysql-router glance-mysql-router
juju add-relation glance-mysql-router:db-router mysql-innodb-cluster:db-router
juju add-relation glance-mysql-router:shared-db glance:shared-db
juju add-relation glance:image-service nova-cloud-controller:image-service
juju add-relation glance:image-service nova-compute:image-service
juju add-relation glance:identity-service keystone:identity-service
juju add-relation glance:certificates vault:certificates
```

Para pasar a la siguiente fase se debe tener las aplicaciones en el siguiente estado (puede tardar unos 40 minutos):

```
Unit                          Workload  Agent  Machine  Public address  Ports              Message
ceph-osd/0*                   blocked   idle   0        10.0.0.150                         Missing relation: monitor
ceph-osd/1                    blocked   idle   1        10.0.0.151                         Missing relation: monitor
ceph-osd/2                    blocked   idle   2        10.0.0.152                         Missing relation: monitor
ceph-osd/3                    blocked   idle   3        10.0.0.153                         Missing relation: monitor
glance/0*                     active    idle   3/lxd/3  10.0.0.167      9292/tcp           Unit is ready
  glance-mysql-router/0*      active    idle            10.0.0.167                         Unit is ready
keystone/0*                   active    idle   0/lxd/2  10.0.0.162      5000/tcp           Unit is ready
  keystone-mysql-router/0*    active    idle            10.0.0.162                         Unit is ready
mysql-innodb-cluster/0*       active    idle   0/lxd/0  10.0.0.154                         Unit is ready: Mode: R/W, Cluster is ONLINE and can tolerate up to ONE failure.
mysql-innodb-cluster/1        active    idle   1/lxd/0  10.0.0.155                         Unit is ready: Mode: R/O, Cluster is ONLINE and can tolerate up to ONE failure.
mysql-innodb-cluster/2        active    idle   2/lxd/0  10.0.0.156                         Unit is ready: Mode: R/O, Cluster is ONLINE and can tolerate up to ONE failure.
neutron-api/0*                active    idle   1/lxd/2  10.0.0.161      9696/tcp           Unit is ready
  neutron-api-mysql-router/0* active    idle            10.0.0.161                         Unit is ready
  neutron-api-plugin-ovn/0*   active    idle            10.0.0.161                         Unit is ready
nova-cloud-controller/0*      active    idle   3/lxd/1  10.0.0.164      8774/tcp,8775/tcp  Unit is ready
  ncc-mysql-router/0*         active    idle            10.0.0.164                         Unit is ready
nova-compute/0*               active    idle   1        10.0.0.151                         Unit is ready
  ovn-chassis/2               active    idle            10.0.0.151                         Unit is ready
nova-compute/1                active    idle   2        10.0.0.152                         Unit is ready
  ovn-chassis/0*              active    idle            10.0.0.152                         Unit is ready
nova-compute/2                active    idle   3        10.0.0.153                         Unit is ready
  ovn-chassis/1               active    idle            10.0.0.153                         Unit is ready
openstack-dashboard/0*        active    idle   2/lxd/3  10.0.0.166      80/tcp,443/tcp     Unit is ready
  dashboard-mysql-router/0*   active    idle            10.0.0.166                         Unit is ready
ovn-central/0*                active    idle   0/lxd/1  10.0.0.158      6641/tcp,6642/tcp  Unit is ready (leader: ovnnb_db, ovnsb_db northd: active)
ovn-central/1                 active    idle   1/lxd/1  10.0.0.159      6641/tcp,6642/tcp  Unit is ready
ovn-central/2                 active    idle   2/lxd/1  10.0.0.160      6641/tcp,6642/tcp  Unit is ready
placement/0*                  active    idle   3/lxd/2  10.0.0.165      8778/tcp           Unit is ready
  placement-mysql-router/0*   active    idle            10.0.0.165                         Unit is ready
rabbitmq-server/0*            active    idle   2/lxd/2  10.0.0.163      5672/tcp           Unit is ready
vault/0*                      active    idle   3/lxd/0  10.0.0.157      8200/tcp           Unit is ready (active: true, mlock: disabled)
  vault-mysql-router/0*       active    idle            10.0.0.157                         Unit is ready
```

8.4.4. Cuarta fase

Por último, hay que ejecutar el cuarto *script*:

```
#!/bin/sh
juju deploy -n 3 --to lxd:0,lxd:1,lxd:2 --config ceph-mon.yaml ceph-mon
juju add-relation ceph-mon:osd ceph-osd:mon
juju add-relation ceph-mon:client nova-compute:ceph
juju add-relation ceph-mon:client glance:ceph
juju deploy --to lxd:1 --config cinder.yaml cinder
juju deploy mysql-router cinder-mysql-router
juju add-relation cinder-mysql-router:db-router mysql-innodb-cluster:db-router
juju add-relation cinder-mysql-router:shared-db cinder:shared-db
juju add-relation cinder:cinder-volume-service nova-cloud-controller:cinder-volume-service
juju add-relation cinder:identity-service keystone:identity-service
juju add-relation cinder:amqp rabbitmq-server:amqp
juju add-relation cinder:image-service glance:image-service
juju add-relation cinder:certificates vault:certificates
juju deploy cinder-ceph
juju add-relation cinder-ceph:storage-backend cinder:storage-backend
juju add-relation cinder-ceph:ceph ceph-mon:client
juju add-relation cinder-ceph:ceph-access nova-compute:ceph-access
juju deploy --to lxd:0 --config source=cloud:focal-wallaby ceph-radosgw
juju add-relation ceph-radosgw:mon ceph-mon:radosgw
juju deploy ntp
juju add-relation ceph-osd:juju-info ntp:juju-info
```

Al final deben estar todas los *Workload* en estado *Active* (puede tardar unos 40 minutos). Con esto finaliza el proceso, todas las aplicaciones quedarían activas y se podría realizar pruebas con la nube.

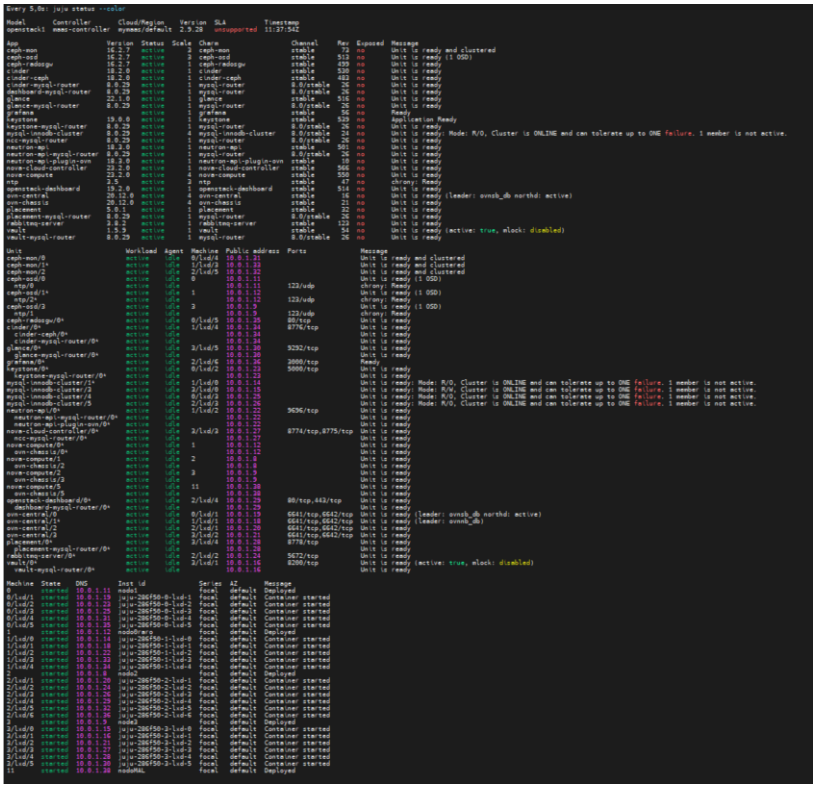


Figura 40: Estado de las aplicaciones tras cuarta fase

9. Pruebas de concepto para nuestro escenario

9.1. Recuperación de la nube ante cortes de luz y fallos

La nube es susceptible ante cortes de luz. Cuando el corte es de menos de media hora aproximadamente, se tiene una *SAI* (Sistema de alimentación ininterrumpida) que se encarga de proteger los equipos de nuestra nube. Este aparato, es un sistema de alimentación ininterrumpida que proporciona respaldo ante situaciones como la que se está comentando, además de prevención de daños a la electrónica de nuestra nube. El modelo de este componente esencial en nuestra nube, se puede consultar en el siguiente enlace [7].

Tras el corte puede que algún nodo no arranque por PXE, probar entonces a arrancar desde el disco duro (si estaba en Deployed ya tenía el sistema operativo instalado) entrando en la BIOS con F10 (en los PC del EL11). Puede tardar un poco en arrancar, ir viendo con *juju status*. Si el corte de luz supone un tiempo superior al comentado, algunas aplicaciones de nuestro modelo pasarían al estado de bloqueado y habría que desbloquearlas de manera manual.

El estado de las aplicaciones cuando los nodos de cómputo caen es el siguiente:

```
Every 5,0s: juju status --color

Model      Controller       Cloud/Region    Version  SLA          Timestamp
openstack  maas-controller  mymaas/default  2.9.28   unsupported  17:29:51Z

App                    Version  Status   Scale  Charm                  Channel  Rev  Exposed  Message
ceph-mon               16.2.7   active       2  ceph-mon               stable    73  no       Unit is ready and clustered
ceph-osd               16.2.7   active       2  ceph-osd               stable   513  no       Unit is ready (1 OSD)
ceph-radosgw           16.2.7   active       1  ceph-radosgw           stable   499  no       Unit is ready
cinder                 19.0.0   active       1  cinder                 stable   530  no       Unit is ready
cinder-ceph            19.0.0   active       1  cinder-ceph            stable   483  no       Unit is ready
cinder-mysql-router    8.0.28   blocked      1  mysql-router           stable    15  no       Failed to connect to MySQL
dashboard-mysql-router 8.0.28   blocked      1  mysql-router           stable    15  no       Failed to connect to MySQL
glance                 23.0.0   active       1  glance                 stable   516  no       Unit is ready
glance-mysql-router    8.0.28   blocked      1  mysql-router           stable    15  no       Failed to connect to MySQL
keystone               20.0.0   active       1  keystone               stable   539  no       Application Ready
keystone-mysql-router  8.0.28   blocked      1  mysql-router           stable    15  no       Failed to connect to MySQL
mysql-innodb-cluster   8.0.28   error        3  mysql-innodb-cluster   stable    15  no       hook failed: "update-status"
ncc-mysql-router       8.0.28   blocked      1  mysql-router           stable    15  no       Failed to connect to MySQL
neutron-api            19.1.0   active       1  neutron-api            stable   501  no       Unit is ready
neutron-api-mysql-router 8.0.28 blocked      1  mysql-router           stable    15  no       Failed to connect to MySQL
neutron-api-plugin-ovn 19.1.0   active       1  neutron-api-plugin-ovn stable    10  no       Unit is ready
nova-cloud-controller  24.0.0   active       1  nova-cloud-controller  stable   566  no       Unit is ready
nova-compute           24.0.0   active       2  nova-compute           stable   550  no       Unit is ready
ntp                    3.5      active       2  ntp                    stable    47  no       chrony: Ready
openstack-dashboard    20.1.0   active       1  openstack-dashboard    stable   514  no       Unit is ready
ovn-central            21.09.0  active       2  ovn-central            stable    16  no       Unit is ready (leader: ovnsb_db northd: active)
ovn-chassis            21.09.0  error        2  ovn-chassis            stable    21  no       hook failed: "config-changed"
placement              6.0.0    active       1  placement              stable    32  no       Unit is ready
placement-mysql-router 8.0.28   blocked      1  mysql-router           stable    15  no       Failed to connect to MySQL
rabbitmq-server        3.8.2    active       1  rabbitmq-server        stable   123  no       Unit is ready
vault                  1.5.9    error        1  vault                  stable    54  no       hook failed: "start"
vault-mysql-router     8.0.28   blocked      1  mysql-router           stable    15  no       Failed to connect to MySQL

Unit                          Workload  Agent  Machine  Public address  Ports              Message
ceph-mon/0                    active    idle   0/lxd/6  10.0.1.100                         Unit is ready and clustered
ceph-mon/1*                   active    idle   1/lxd/7  10.0.1.101                         Unit is ready and clustered
ceph-osd/0*                   active    idle   0        10.0.1.72                          Unit is ready (1 OSD)
  ntp/1*                      active    idle            10.0.1.72       123/udp            chrony: Ready
ceph-osd/1                    active    idle   1        10.0.1.87                          Unit is ready (1 OSD)
  ntp/0                       active    idle            10.0.1.87       123/udp            chrony: Ready
ceph-radosgw/0*               active    idle   0/lxd/7  10.0.1.103      80/tcp             Unit is ready
cinder/0*                     active    idle   1/lxd/8  10.0.1.102      8776/tcp           Unit is ready
  cinder-ceph/0*              active    idle            10.0.1.102                         Unit is ready
  cinder-mysql-router/0*      blocked   idle            10.0.1.102                         Failed to connect to MySQL
glance/0*                     active    idle   1/lxd/6  10.0.1.99       9292/tcp           Unit is ready
  glance-mysql-router/0*      blocked   idle            10.0.1.99                          Failed to connect to MySQL
keystone/0*                   active    idle   0/lxd/3  10.0.1.94       5000/tcp           Unit is ready
  keystone-mysql-router/0*    blocked   idle            10.0.1.94                          Failed to connect to MySQL
mysql-innodb-cluster/0*       error     idle   0/lxd/0  10.0.1.90                          hook failed: "update-status"
mysql-innodb-cluster/1        blocked   idle   1/lxd/0  10.0.1.68                          Cluster is inaccessible from this instance. Please check logs for details.
mysql-innodb-cluster/2        blocked   idle   0/lxd/1  10.0.1.89                          Cluster is inaccessible from this instance. Please check logs for details.
neutron-api/0*                active    idle   1/lxd/3  10.0.1.93       9696/tcp           Unit is ready
  neutron-api-mysql-router/0* blocked   idle            10.0.1.93                          Failed to connect to MySQL
  neutron-api-plugin-ovn/0*   active    idle            10.0.1.93                          Unit is ready
nova-cloud-controller/0*      active    idle   0/lxd/4  10.0.1.97       8774/tcp,8775/tcp  Unit is ready
  ncc-mysql-router/0*         blocked   idle            10.0.1.97                          Failed to connect to MySQL
nova-compute/0*               active    idle   0        10.0.1.72                          Unit is ready
  ovn-chassis/0*              error     idle            10.0.1.72                          hook failed: "config-changed"
nova-compute/1                active    idle   1        10.0.1.87                          Unit is ready
  ovn-chassis/1               active    idle            10.0.1.87                          Unit is ready
openstack-dashboard/0*        active    idle   1/lxd/5  10.0.1.96       80/tcp,443/tcp     Unit is ready
  dashboard-mysql-router/0*   blocked   idle            10.0.1.96                          Failed to connect to MySQL
ovn-central/0*                active    idle   0/lxd/2  10.0.1.91       6641/tcp,6642/tcp  Unit is ready (leader: ovnsb_db northd: active)
ovn-central/1                 active    idle   1/lxd/2  10.0.1.92       6641/tcp,6642/tcp  Unit is ready (leader: ovnnb_db)
placement/0*                  active    idle   0/lxd/5  10.0.1.98       8778/tcp           Unit is ready
  placement-mysql-router/0*   blocked   idle            10.0.1.98                          Failed to connect to MySQL
rabbitmq-server/0*            active    idle   1/lxd/4  10.0.1.95       5672/tcp           Unit is ready
vault/0*                      error     idle   1/lxd/1  10.0.1.88       8200/tcp           hook failed: "start"
  vault-mysql-router/0*       blocked   idle            10.0.1.88                          Failed to connect to MySQL

Machine  State    DNS           Inst id              Series  AZ       Message
0        started  10.0.1.72     nodo2                focal   default  Deployed
0/lxd/0  started  10.0.1.90     juju-8db57a-0-lxd-0  focal   default  Container started
0/lxd/1  started  10.0.1.89     juju-8db57a-0-lxd-1  focal   default  Container started
0/lxd/2  started  10.0.1.91     juju-8db57a-0-lxd-2  focal   default  Container started
0/lxd/3  started  10.0.1.94     juju-8db57a-0-lxd-3  focal   default  Container started
0/lxd/4  started  10.0.1.97     juju-8db57a-0-lxd-4  focal   default  Container started
0/lxd/5  started  10.0.1.98     juju-8db57a-0-lxd-5  focal   default  Container started
0/lxd/6  started  10.0.1.100    juju-8db57a-0-lxd-6  focal   default  Container started
0/lxd/7  started  10.0.1.103    juju-8db57a-0-lxd-7  focal   default  Container started
1        started  10.0.1.87     nodo1                focal   default  Deployed
1/lxd/0  started  10.0.1.68     juju-8db57a-1-lxd-0  focal   default  Container started
1/lxd/1  started  10.0.1.88     juju-8db57a-1-lxd-1  focal   default  Container started
1/lxd/2  started  10.0.1.92     juju-8db57a-1-lxd-2  focal   default  Container started
1/lxd/3  started  10.0.1.93     juju-8db57a-1-lxd-3  focal   default  Container started
1/lxd/4  started  10.0.1.95     juju-8db57a-1-lxd-4  focal   default  Container started
1/lxd/5  started  10.0.1.96     juju-8db57a-1-lxd-5  focal   default  Container started
1/lxd/6  started  10.0.1.99     juju-8db57a-1-lxd-6  focal   default  Container started
1/lxd/7  started  10.0.1.101    juju-8db57a-1-lxd-7  focal   default  Container started
1/lxd/8  started  10.0.1.102    juju-8db57a-1-lxd-8  focal   default  Container started
```

Figura 41: Estado de las aplicaciones tras un apagado de los nodos

Como se puede observar en la figura anterior [42], hay dos aplicaciones que han llegado al estado de error. Una de ellas es *mysql-innodb-cluster* y otra es *vault*, aunque también puede ocurrir que haya muchas más, al

recuperar estas dos el resto suelen funcionar. El error de las aplicaciones está relacionado, en el momento que se desbloquea *mysql-innodb-cluster* la aplicaciones en estado *blocked* se activarán de nuevo. Si se observa con detenimiento la salida del comando *juju status* se ve que todas las bases de datos que utilizan las aplicaciones están bloqueadas. Tras búsquedas de información me di cuenta que esto era un *bug* que viene recogido en el siguiente enlace [4].

Lo que se hace es lanzar un comando *JuJu* que reinicia la aplicación problemática:

```
juju run-action --wait mysql-innodb-cluster/<Nº de leader> reboot-cluster-from-complete-outage
```

El estado de las aplicaciones cuando termina de ejecutarse esta acción es:

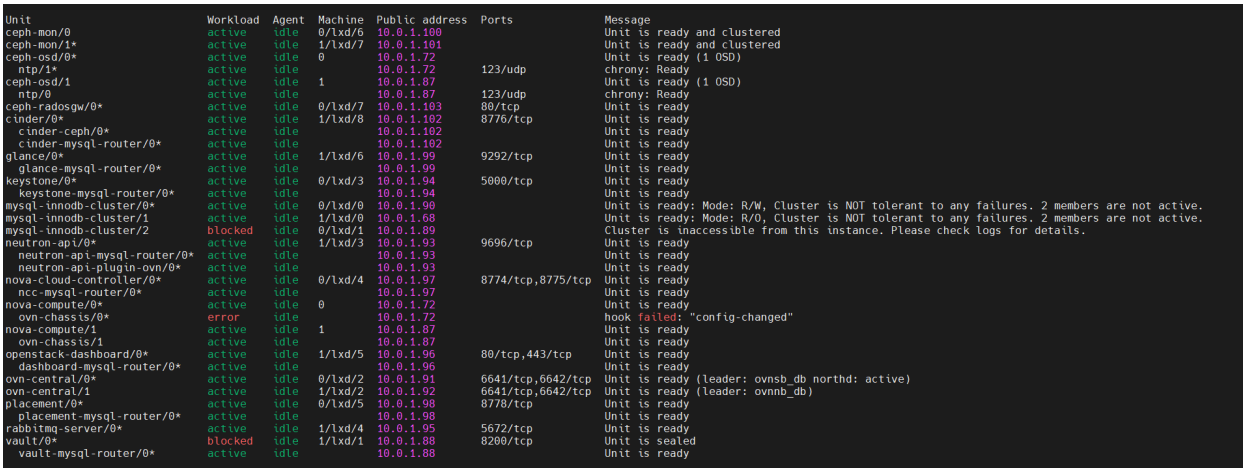


Figura 42: Segundo estado de las aplicaciones tras un apagado de los nodos

Tras esto, *vault* sigue bloqueada, pero se conoce esta situación. La aplicación devuelve en el mensaje de estado que la unidad está sellada. Para desbloquearla, simplemente hay que ir al apartado 8.4.1 y desbloquear *vault* como se ha explicado.

Por último puede ocurrir que no se consiga desbloquear una aplicación, en este caso se puede añadir otra unidad de la aplicación y cuando esté funcionando parar y borrar la que esté bloqueada.

9.2. Añadir nodos de cómputo a la nube

Una nube de computación tiene tres grandes bloques de recursos que deben ser gestionados. El primer bloque es el de almacenamiento, el cual es el encargado de proveer memoria a todos los recursos que ofrece el sistema. Otro recurso es la red. La arquitectura de red que utilice la nube es vital para el buen funcionamiento del sistema. Y por último, otro punto clave es el cómputo. En este apartado, se ofrece una guía para añadir recursos a la nube.

Para ofrecer una explicación de manera clara, se parte de un escenario inicial en el cual las aplicaciones de *OpenStack* están desplegadas en dos nodos de cómputo, es decir nuestro sistema dispone de la siguiente arquitectura de red:

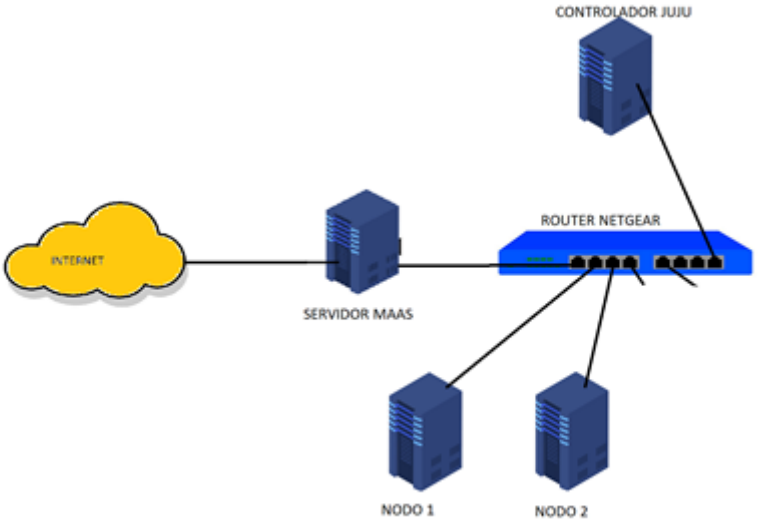


Figura 43: Escenario con dos nodos de cómputo

Este cómputo del que se está hablando no es más que añadir un tercer nodo a nuestro escenario. En la *API* web de *MaaS* se tiene el nodo en estado *Ready* (el apartado [6.2.3] muestra como llegar a este estado) para poder desplegar el nodo. La siguiente imagen muestra en qué punto estamos.

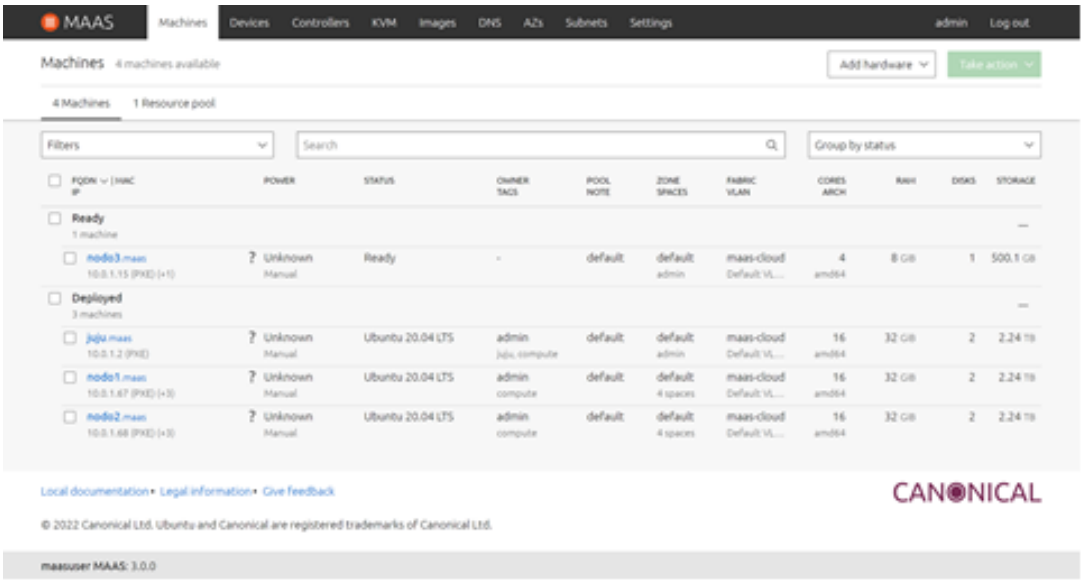


Figura 44: máquinas preparadas para desplegar

La distribución de las aplicaciones en los nodos de es la siguiente:

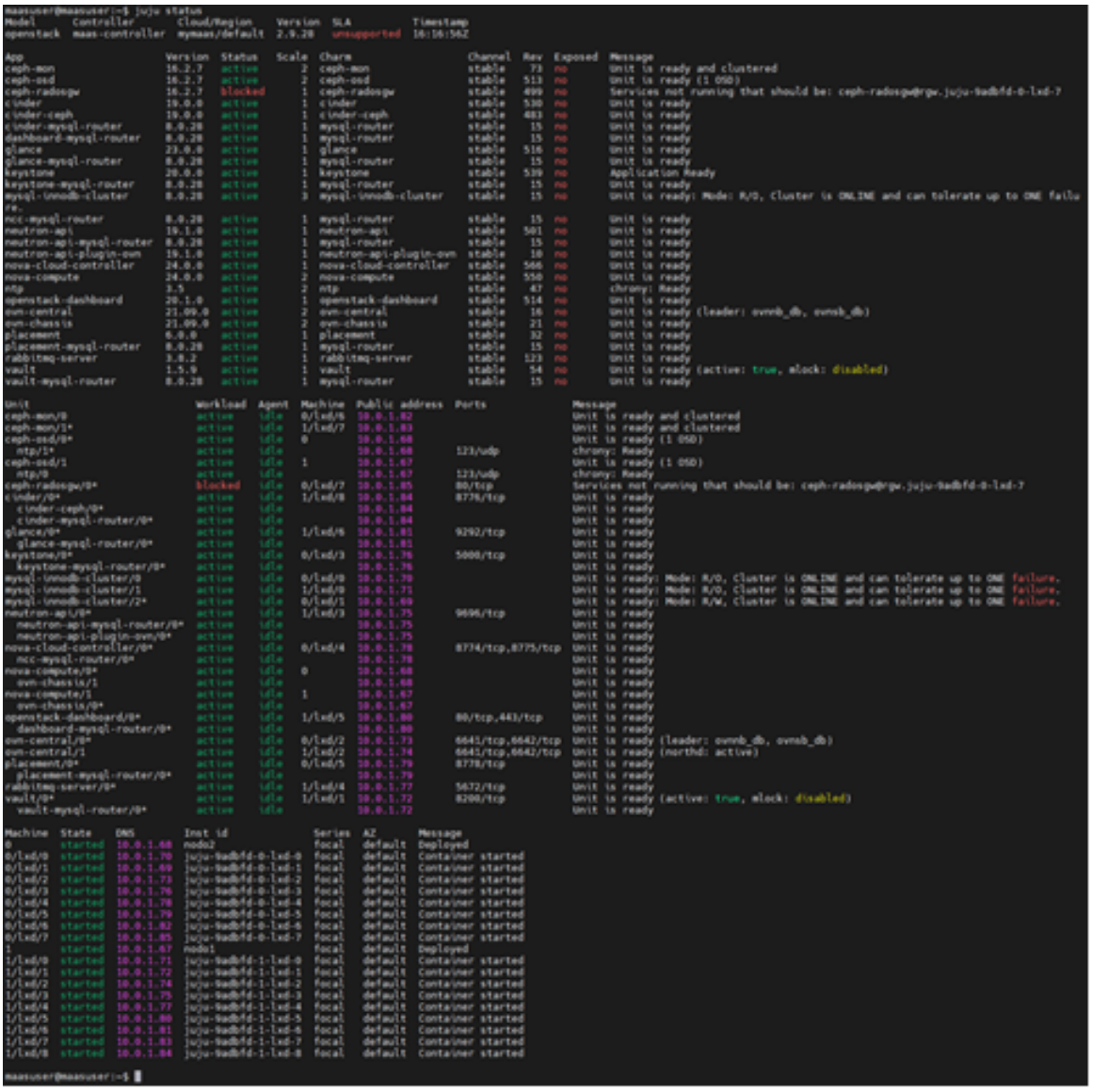


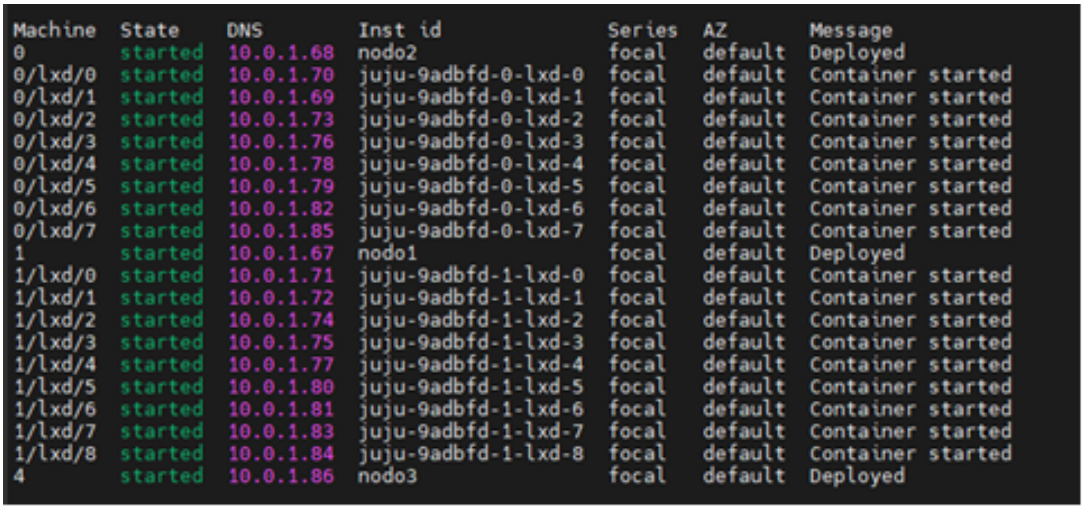
Figura 45: máquinas preparadas para desplegar

Como se puede observar en la imagen anterior [45] la aplicación clave en el nivel de cómputo del sistema es nova-compute. El objetivo entonces es añadir unidades para que esta aplicación tenga capacidad para aguantar más peticiones de recursos a la nube.

Es clave en este punto, que la máquina nueva este comisionada, tenga las interfaces de red bien configuradas como en el apartado 6.2.3 (*postinstalación de MaaS*) y sea etiquetada correctamente (en nuestro caso con la etiqueta compute). En este momento se añade la máquina con el siguiente comando:

```
juju add-machine  --constraints tags=compute
```

La máquina pasará al estado *Deploying* y habrá que encender el servidor para que complete el despliegue que consistirá en instalar el sistema operativo en el disco duro de arranque (el que esté conectado al SATA con número más bajo) y con la configuración de red hecha en el MAAS. El comando *juju machines* después de completar el proceso de reconocimiento del nodo devuelve lo siguiente:



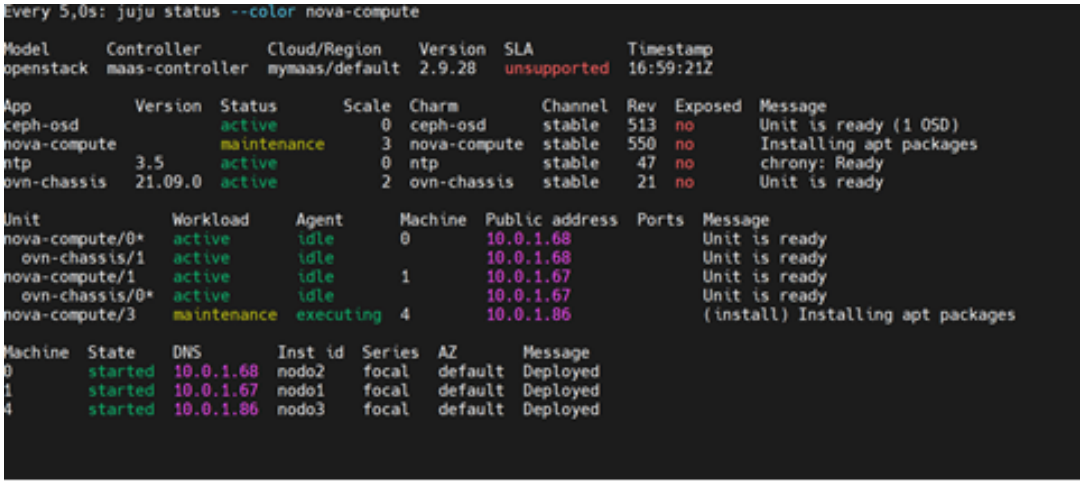
```
Machine  State    DNS          Inst id              Series  AZ       Message
0        started  10.0.1.68    nodo2                focal   default  Deployed
0/lxd/0  started  10.0.1.70    juju-9adbfd-0-lxd-0  focal   default  Container started
0/lxd/1  started  10.0.1.69    juju-9adbfd-0-lxd-1  focal   default  Container started
0/lxd/2  started  10.0.1.73    juju-9adbfd-0-lxd-2  focal   default  Container started
0/lxd/3  started  10.0.1.76    juju-9adbfd-0-lxd-3  focal   default  Container started
0/lxd/4  started  10.0.1.78    juju-9adbfd-0-lxd-4  focal   default  Container started
0/lxd/5  started  10.0.1.79    juju-9adbfd-0-lxd-5  focal   default  Container started
0/lxd/6  started  10.0.1.82    juju-9adbfd-0-lxd-6  focal   default  Container started
0/lxd/7  started  10.0.1.85    juju-9adbfd-0-lxd-7  focal   default  Container started
1        started  10.0.1.67    nodo1                focal   default  Deployed
1/lxd/0  started  10.0.1.71    juju-9adbfd-1-lxd-0  focal   default  Container started
1/lxd/1  started  10.0.1.72    juju-9adbfd-1-lxd-1  focal   default  Container started
1/lxd/2  started  10.0.1.74    juju-9adbfd-1-lxd-2  focal   default  Container started
1/lxd/3  started  10.0.1.75    juju-9adbfd-1-lxd-3  focal   default  Container started
1/lxd/4  started  10.0.1.77    juju-9adbfd-1-lxd-4  focal   default  Container started
1/lxd/5  started  10.0.1.80    juju-9adbfd-1-lxd-5  focal   default  Container started
1/lxd/6  started  10.0.1.81    juju-9adbfd-1-lxd-6  focal   default  Container started
1/lxd/7  started  10.0.1.83    juju-9adbfd-1-lxd-7  focal   default  Container started
1/lxd/8  started  10.0.1.84    juju-9adbfd-1-lxd-8  focal   default  Container started
4        started  10.0.1.86    nodo3                focal   default  Deployed
```

Figura 46: máquina cuatro en estado *Deployed*

Como se puede observar se ha añadido al sistema el nodo tres. Una vez que la máquina pase al estado *Deployed*, tarda unos cinco minutos, se pasa a añadir la unidad de cómputo que nos interesa con el siguiente comando *JuJu*:

```
juju add-unit --to 4 nova-compute
```

Con la etiqueta -–*to* se indica a la aplicación donde se quiere añadir la unidad, en nuestro caso es la máquina cuatro como se puede observar en la imagen anterior [46]. Se observa la salida de *juju status* para ver el proceso de instalación de la unidad:



```
Every 5,0s: juju status --color nova-compute

Model      Controller       Cloud/Region    Version  SLA          Timestamp
openstack  maas-controller  mymaas/default  2.9.28   unsupported  16:59:21Z

App           Version  Status       Scale  Charm         Channel  Rev  Exposed  Message
ceph-osd               active           0  ceph-osd      stable   513  no       Unit is ready (1 OSD)
nova-compute           maintenance      3  nova-compute  stable   550  no       Installing apt packages
ntp           3.5      active           0  ntp           stable    47  no       chrony: Ready
ovn-chassis   21.09.0  active           2  ovn-chassis   stable    21  no       Unit is ready

Unit              Workload     Agent      Machine  Public address  Ports  Message
nova-compute/0*   active       idle       0        10.0.1.68              Unit is ready
  ovn-chassis/1   active       idle                10.0.1.68              Unit is ready
nova-compute/1    active       idle       1        10.0.1.67              Unit is ready
  ovn-chassis/0*  active       idle                10.0.1.67              Unit is ready
nova-compute/3    maintenance  executing  4        10.0.1.86              (install) Installing apt packages

Machine  State    DNS          Inst id  Series  AZ       Message
0        started  10.0.1.68    nodo2    focal   default  Deployed
1        started  10.0.1.67    nodo1    focal   default  Deployed
4        started  10.0.1.86    nodo3    focal   default  Deployed
```

Figura 47: Máquina tres en estado *Deployed*

Cuando se termina el proceso, puede tardar más de una hora, la salida del comando *juju status* es la siguiente:

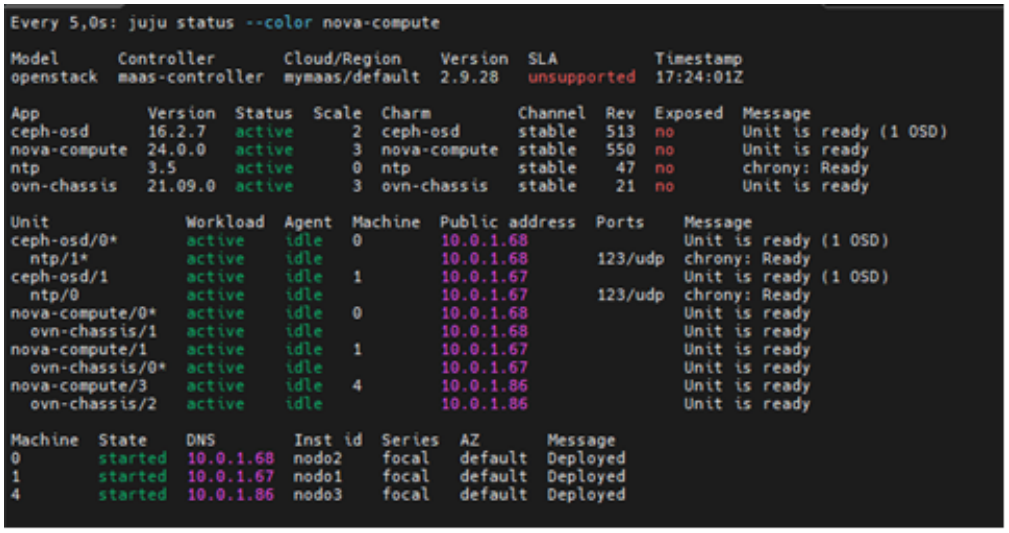


Figura 48: Activación de la unidad nueva de *nova-compute*

Como se puede observar se ha añadido la unidad correctamente ya que la aplicación queda en el estado activo. Se ha aumentado la capacidad de cómputo de la nube y ninguna de las otras aplicaciones habrá sufrido ningún cambio de estado. La arquitectura de red de la nube queda reflejada en la siguiente imagen:

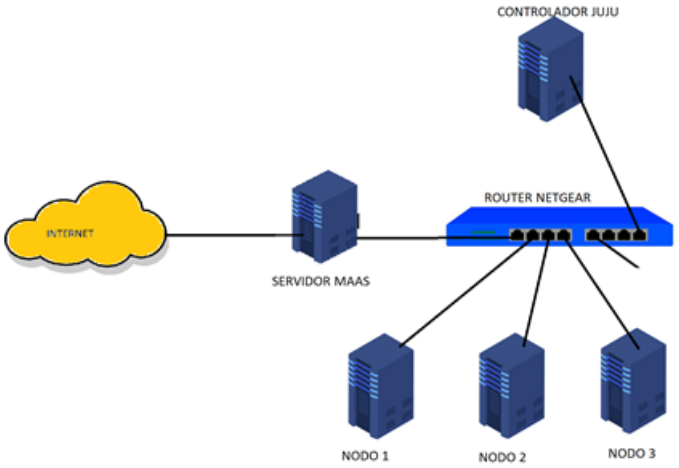


Figura 49: Nueva máquina de cómputo

9.3. Quitar nodos de cómputo de la nube

Quitar nodos es una acción destructiva del entorno. Si se quiere quitar recursos de cómputo al sistema, es importante tener constancia de lo que está corriendo en dicha entidad. Como ya se ha hablado en apartados anteriores [7.1.2], las aplicaciones tienen relaciones entre sí que pueden verse afectadas por esta acción, pasando las mismas a un estado de bloqueado. La configuración de las aplicaciones juega un papel fundamental en el caso que se está exponiendo, ya que por ejemplo, en el fichero *ceph-mon.yaml* viene indicado el número de unidades que se van a desplegar en nuestras máquinas y si se destruye una de estas unidades, la aplicación pasaría a un estado bloqueado. Con respecto a las relaciones que están creadas entre las aplicaciones del sistema, con esta acción quedan eliminadas.

La acción que destruye un nodo de cómputo o cualquier contenedor contenedor que contenga dicho nodo es:

```
juju remove-machine [nombre de la máquina] --force
```

En principio esto también elimina las aplicaciones que corren en el nodo. Conviene comprobarlo con *juju status aplicación*, si no se elimina más adelante se indica el comando para ello. Un caso de uso de esto, es una situación que se nos presentó, en la que un nodo de cómputo experimentó un fallo. Se requería migrar todas las aplicaciones que corrían en este nodo a los nodos que no experimentaban fallos. La situación viene recogida en la salida del comando *juju status*:

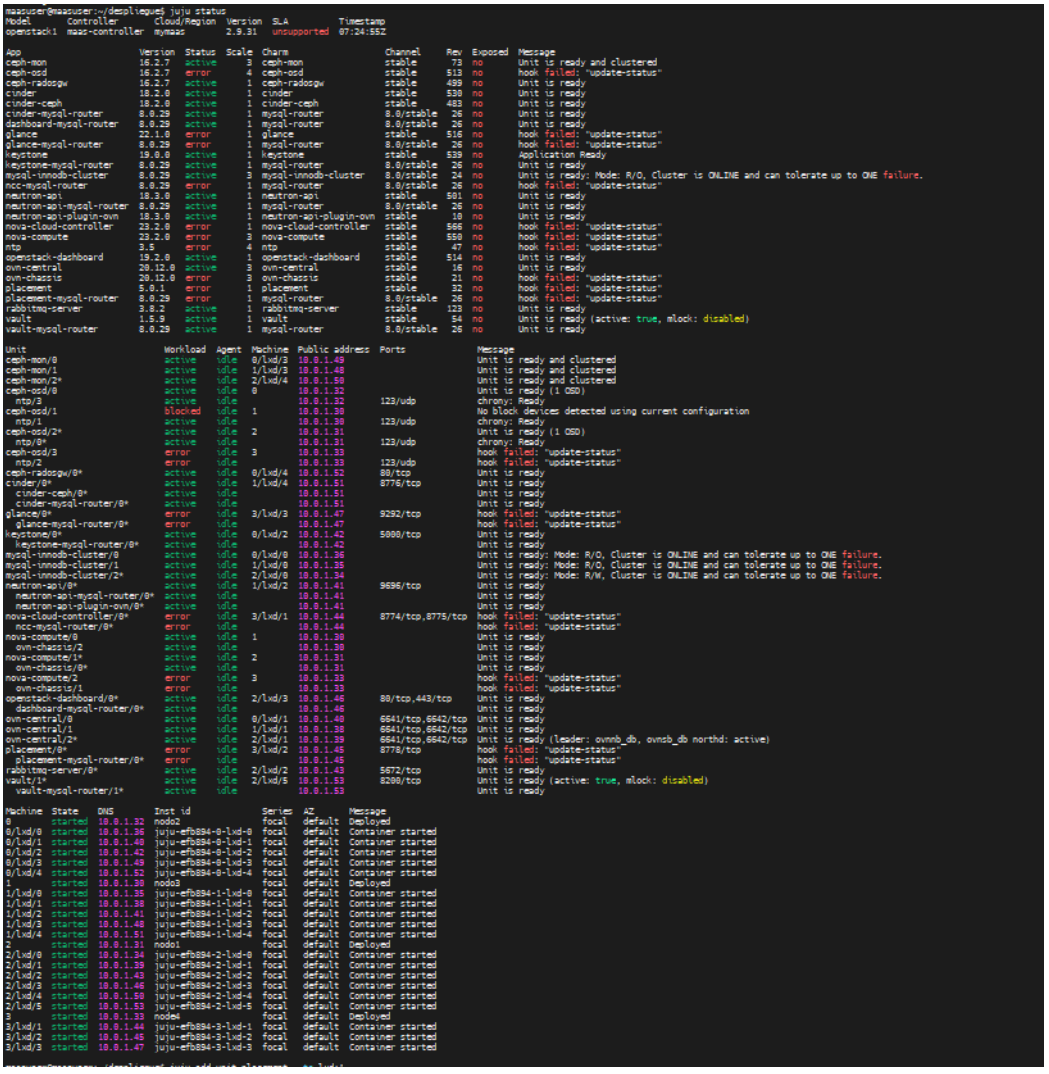


Figura 50: Fallo del nodo 4

Como se puede observar, las aplicaciones que corren en el nodo cuatro están todas en estado de error.

Por cada aplicación de error se añade una unidad, siempre repartidas para compensar la carga de los nodos en buen estado. Los comandos para añadir unidades a las aplicaciones que en este caso, están en estado de error son:

```
juju add-unit nova-c-compute --to 0
juju add-unit nova-compute --to 0
juju add-unit nova-cloud-controller --to lxd:2
juju add-unit glance --to lxd:1
```

Como se aprecia en estos comandos, con la etiqueta *–to* decidimos en qué máquina se instalan las nuevas unidades.

Ahora, se va a eliminar las unidades que quedan en estado de error en la máquina que ha fallado:

```
juju remove-unit placement/0 --force --no-wait
juju remove-unit nova-cloud-controller/0 --force --no-wait
juju remove-unit ceph-osd/3 --force --no-wait
juju remove-unit nova-compute/2 --force --no-wait
juju remove-unit glance/0 --force --no-wait
```

La salida del comando *juju status* después de esto es la siguiente:

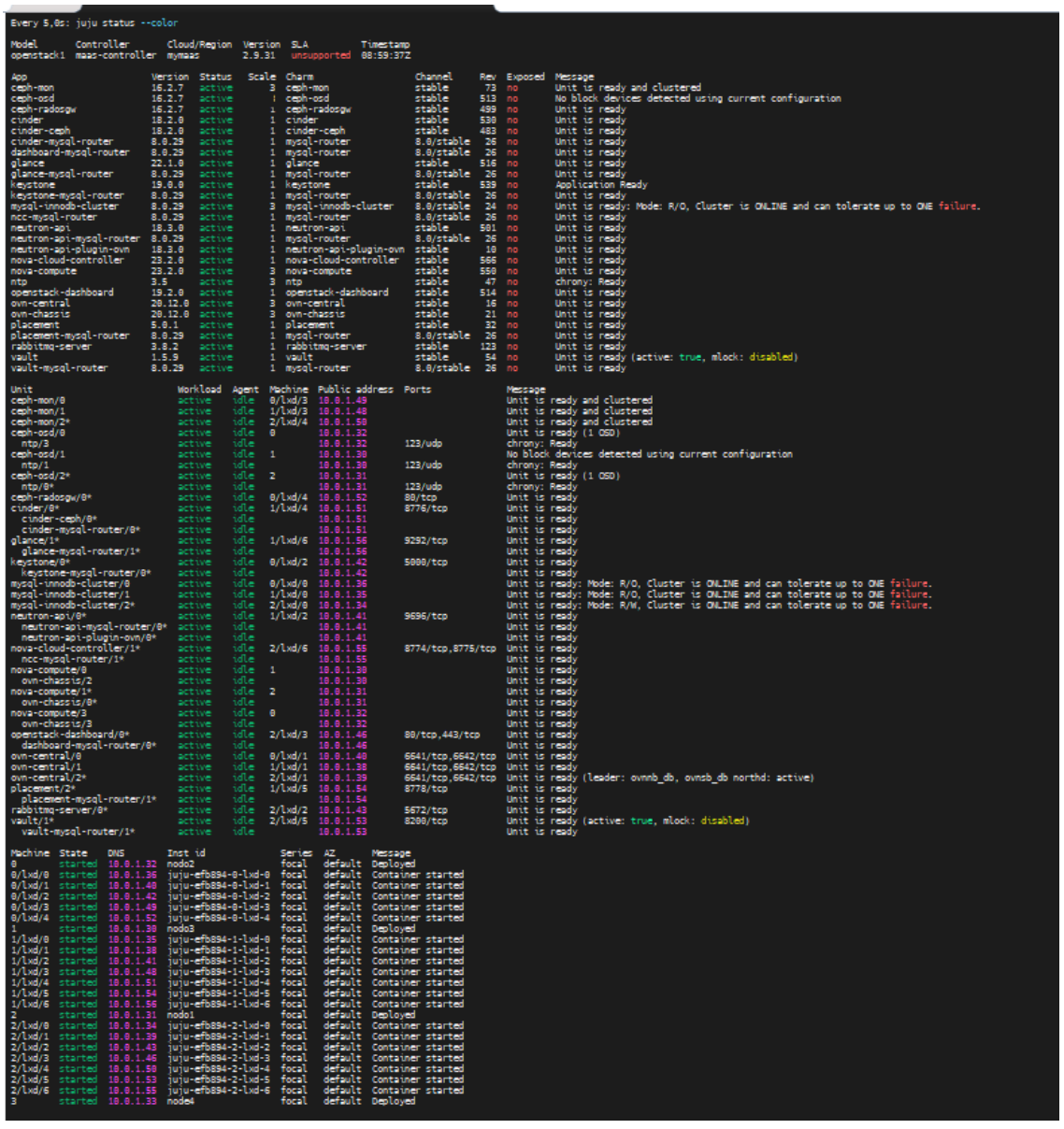


Figura 51: Migración de las aplicaciones

La nube vuelve a funcionar con normalidad. En este momento, se puede proceder a estudiar el problema del nodo sin afectar al funcionamiento de la nube. Cuando se solucione el problema se podría volver a utilizar el nodo, desplegando nuevas unidades en él.

Para actualizar un nodo habría que borrarlo del *MaaS* y empezar el proceso de añadirlo desde la fase de reconocimiento.

9.4. Escenario de pruebas para usuarios de la nube

Con todas las aplicaciones funcionando, lo primero que se hace, será acceder al interfaz web de *Horizon* para generar un simulacro que probará nuestra nube de computación. El comando que nos dice la dirección IP privada por la cual se accede a dicha interfaz es:

```
juju status openstack-dashboard | grep public-address | awk '{print $2}' | head -1
```

La salida de este comando, se copia en cualquier buscador del navegador de nuestro portátil personal, es imprescindible en este primer paso, que la máquina desde la que se accede este en la misma red que los nodos y el controlador *JuJu*. Se comprueba que esté accesible. En nuestro caso, la interfaz web ha caído en la IP privada: 10.0.1.65:80.

El usuario por defecto, es *admin* y la contraseña la conseguimos con el siguiente comando:

```
juju run --unit keystone/leader leader-get admin_passwd
```

Para hacer accesible desde la VPN de la UAH, la interfaz web desde la cual se pueden lanzar instancias, se tiene que realizar una traducción NAT en la máquina que comunica la nube con el exterior (Servidor *MaaS*). En nuestro caso, se tienen accesibles once direcciones IP del laboratorio para nuestra prueba, todas estas direcciones se asociarán al interfaz *eth0* del servidor *MaaS* con el siguiente comando:

```
ip a add {ip_addr/mask} dev {interface}
```

En nuestro caso sustituimos lo que está dentro de los corchetes por las once direcciones disponibles. Las direcciones disponibles para la prueba son: 172.29.21.191/22-191.29.21.200/22. Además de la que está ya configurada en el servidor *MaaS* que es la 172.29.21.202/22.

Se utilizan los comandos *iptables* para traducir la dirección privada del *OpenStack Dashboard* a una IP pública (desde la VPN de la UAH). En este caso se redirige el tráfico que entra por el puerto 80 de la dirección 172.29.21.202 hacia el puerto 80 de la IP privada 10.0.1.65. Los comandos *iptables* que realizan esto son:

```
sudo iptables -t nat -A PREROUTING -i eth0 -d 172.29.21.202 -j DNAT --to-destination 10.0.1.65
sudo iptables -A FORWARD -s 172.29.21.202 -j ACCEPT
sudo iptables -A FORWARD -d 10.0.1.65 -j ACCEPT
```

Una vez llegados a este punto, en el cual ya se tiene acceso a la aplicación *OpenStack Dashboard*, se puede empezar a construir nuestro escenario *OpenStack*.

El escenario que se quiere crear viene representado en la siguiente figura:

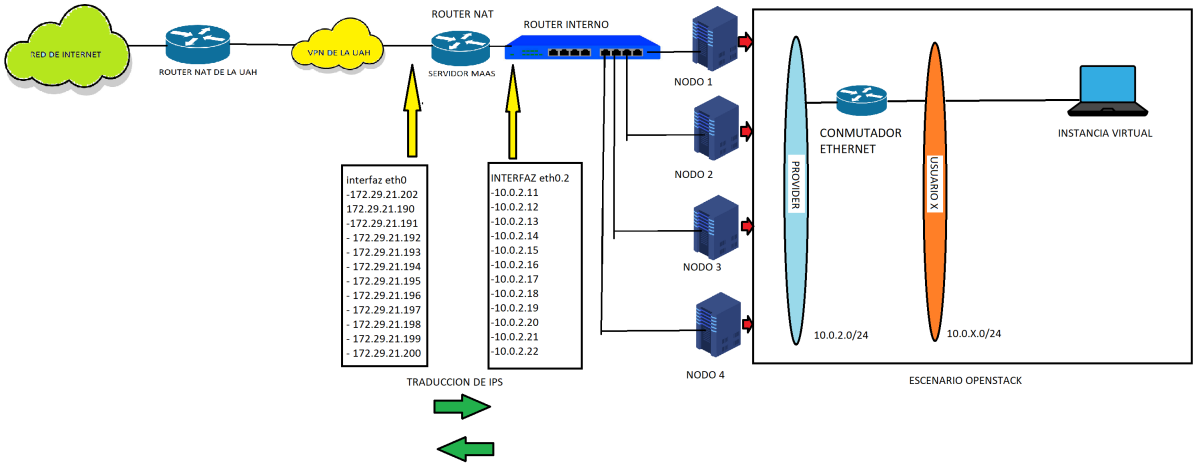


Figura 52: Escenario en el mundo *OpenStack*

La red que se ha llamado *provider* que tiene direcciones del rango 10.0.2.0/24 es por la que las instancias saldrán a internet y de esta red colgaran las redes de los usuarios que pertenecerán al rango 10.0.X.0/24, excluyendo la 10.0.2.0/24 ya que se produciría solapamiento. Entre la red perteneciente a cada usuario y la red *provider*, se crea un *router NAT OpenStack*. Este router es el encargado de traducir las direcciones de las IP flotantes asociadas a las instancias virtuales.

La red *provider*, tiene que coincidir con la que se ha configurado en el archivo *neutron.yaml*, para que el escenario funcione. Para crear la red y que realice la función descrita anteriormente, se ejecutan los siguientes comandos:

```
source ~/openstack-bundles/stable/openstack-base/openrc
openstack network create provider --share --external --provider-network-type flat --provider-physical-network physnet1
openstack subnet create sub_provider --allocation-pool start=10.0.2.200,end=10.0.2.250 --subnet-range 10.0.2.0/24
    --no-dhcp --gateway 10.0.2.1 --dns-nameserver 10.0.2.1 --dns-nameserver 8.8.8.8 --network provider
```

Todo este ecosistema se construye mediante un *script*, que crea los usuarios mediante un fichero de texto plano donde vienen recogidos los correos electrónicos de los usuarios. El *script* tiene el siguiente aspecto:

```
#!/bin/bash
source ~/openstack-bundles/development/openstack-base-focal-wallaby/openrc
i=235
floatip=131
while IFS= read -r correo
do
   IFS='@' read -ra ADDR <<< "$correo"
   echo "Generando  proyecto: ${ADDR[0]}"
   comando="openstack project create --domain admin_domain ${ADDR[0]}"
   comando2="$($comando)"
   echo "$comando2"
   echo "Configurando proyecto: ${ADDR[0]}"
   comando3="openstack role add --user admin --project ${ADDR[0]} admin"
   comando4="$($comando3)"
   echo "Generando usuario: ${ADDR[0]}"
   comando7="openstack user create --domain admin_domain --project ${ADDR[0]} --email $correo --password ${ADDR[0]}$fecha ${ADDR[0]}"
   comando8="$($comando7)"
   echo "$comando8"
   comando9="openstack role add --user ${ADDR[0]} --project ${ADDR[0]} member"
   comando10="$($comando9)"
   echo "$comando10"
   echo "Reservando IP Flotante para el usuario ${ADDR[0]}"
   comando21="openstack floating ip create --floating-ip-address 10.0.2.$floatip --project ${ADDR[0]} provider"
   comando22="$($comando21)"
   ((floatip++))
done < correos.txt
#segúndo bucle
while IFS= read -r correo
do
   ((i++))
   IFS='@' read -ra ADDR <<< "$correo"
   echo "Generando network: ${ADDR[0]}"
   comando11="openstack network create --project ${ADDR[0]} --no-share --internal ${ADDR[0]}"
   comando12="$($comando11)"
   echo "$comando12"
   comando13="openstack subnet create --project ${ADDR[0]} --subnet-range 10.0.1.$i/24 --network ${ADDR[0]} ${ADDR[0]}"
   comando14="$($comando13) --dns-nameserver 8.8.8.8"
   echo "$comando14"
   echo "Generando router: ${ADDR[0]}"
```

```
    comando15="openstack router create --project ${ADDR[0]} ${ADDR[0]}"
    comando16="$($comando15)"
    echo "$comando16"
    comando17="openstack router set --external-gateway provider ${ADDR[0]}"
    comando18="$($comando17)"
    echo "$comando18"
    comando19="openstack router add subnet ${ADDR[0]} ${ADDR[0]}"
    comando20="$($comando19)"
    echo "$comando20"
done < correos.txt
```

Para hacer bueno el *script* simplemente se ejecuta:

```
./script-usuarios.sh
```

Este *script*, nos crea el escenario descrito anteriormente a excepción de la instancia virtual [52], que eso se va a realizar desde la interfaz web, autenticándonos con cualquiera de los usuarios que se han creado. Para ello, se accede a la web de *Horizon* que gestiona los recursos (http://172.29.21.202/horizon si se está conectado a la VPN de la universidad o http://10.0.1.65/horizon si se está dentro de la red de la nube). Nos pedirá las credenciales de acceso.

Una vez nos hemos autenticado, se comprueba que se ha creado el escenario que se ha descrito. Se pincha sobre cualquiera de los proyectos creados (se crea un proyecto por usuario) y se visualiza la topología que se ha descrito:

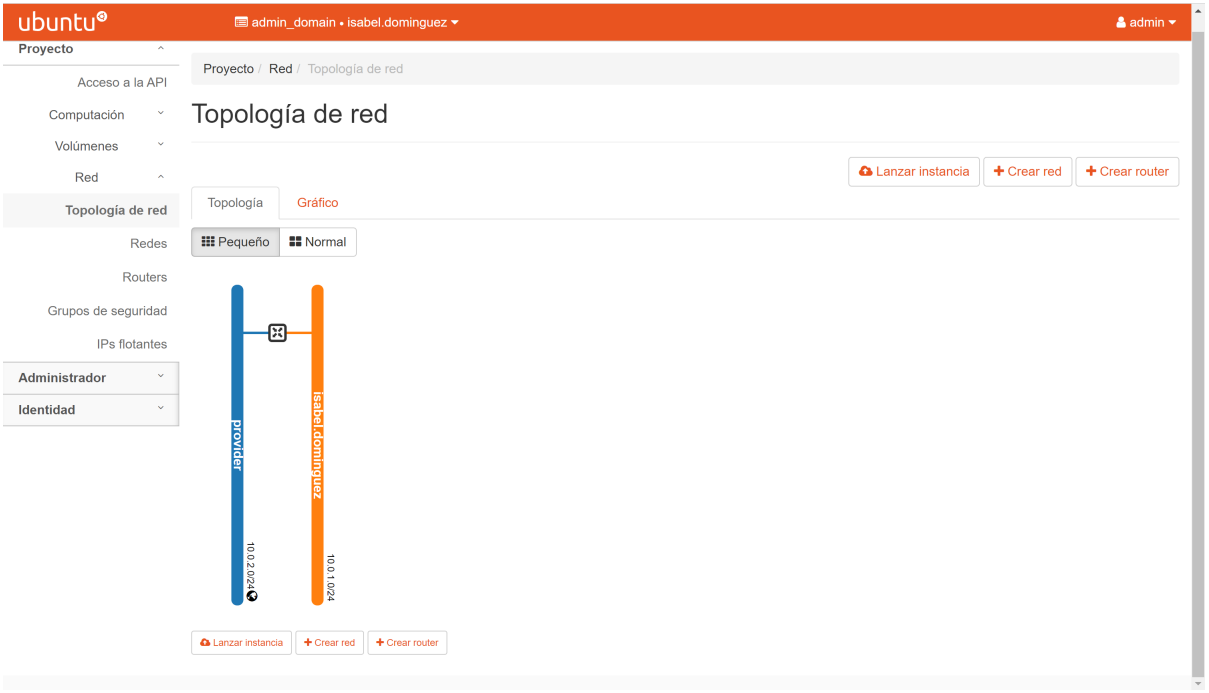


Figura 53: Comprobación de la topología de red

Como se puede observar, el escenario coincide con el explicado. Se puede levantar la primera instancia. Antes de ello, se debe crear una imagen para la misma, se agrega dicha imagen de la siguiente manera:

```
sudo wget https://cloud-images.ubuntu.com/focal/current/focal-server-cloudimg-amd64-disk-kvm.img
openstack image create --public --min-disk 5 --min-ram 1024 --container-format bare --disk-format qcow2
   --property architecture=x86_64 --property hw_disk_bus=virtio --property hw_vif_model=virtio
       --file focal-server-cloudimg-amd64-disk-kvm.img "Ubuntu20"
```

También, se tiene que crear un *Sabor* con el siguiente comando:

```
openstack flavor create --public ubuntu20.flavor --id auto --ram 1024 --disk 5 --vcpus 1 --rxtx-factor 1
```

Se crean un par de claves para las instancias, eso se hace con el siguiente comando:

```
openstack keypair create --public-key ~/.ssh/id_rsa.pub MyKeypair
```

Nos dirigiremos a la pizarra *Instancias* y se pulsa sobre el botón lanzar instancia:

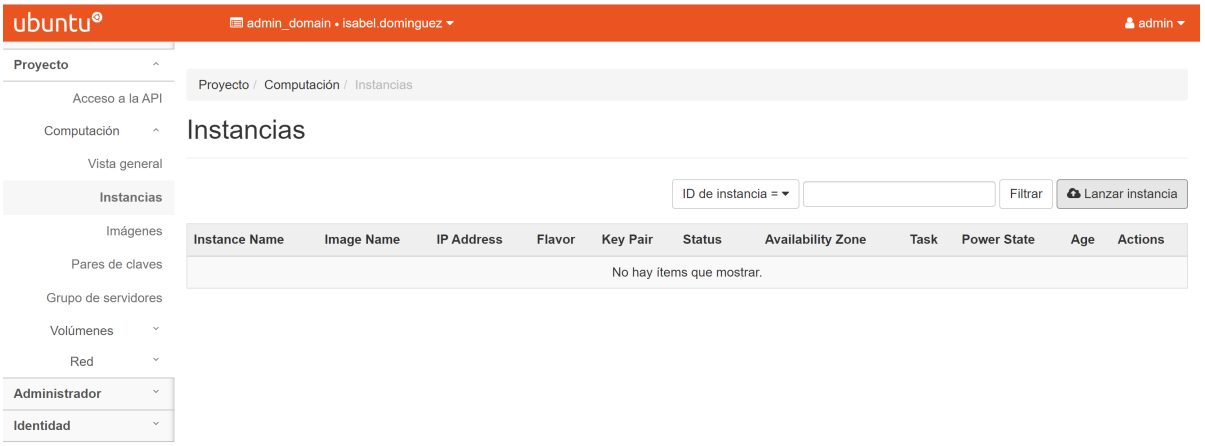


Figura 54: Lanzando la instancia

La siguiente pantalla, nos revela las características que va a tener nuestra instancia. Lo siguiente que se debe hacer es dotar de un nombre a la misma:

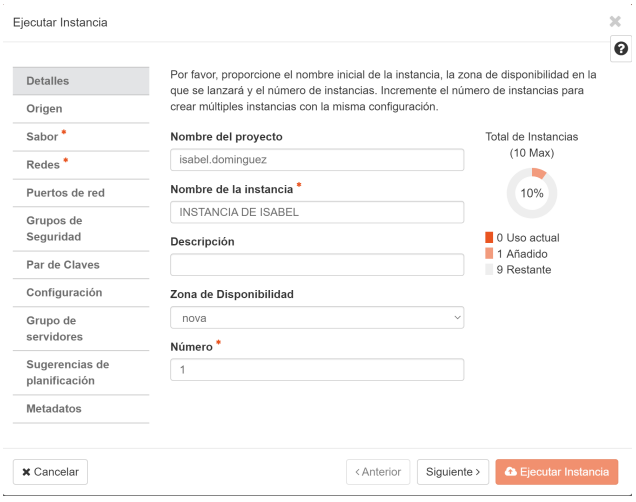


Figura 55: Nombre de la instancia

Se usa el sabor que se ha creado anteriormente:

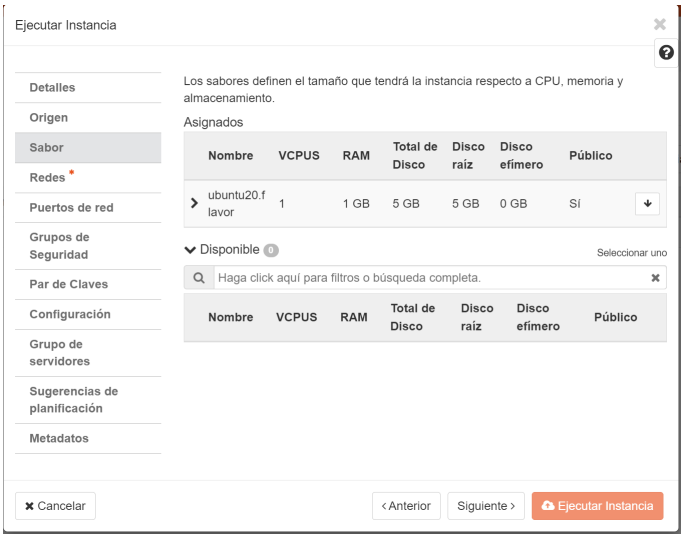


Figura 56: Sabor de la instancia

Como se puede observar, la máquina virtual tendrá 1 GB de memoria RAM, 5 GB de memoria destinada al almacenamiento entre otras características... Ahora se usa la imagen que se ha creado anteriormente para desplegar el sistema operativo en la instancia virtual (en nuestro caso es un Ubuntu Server 20.04).

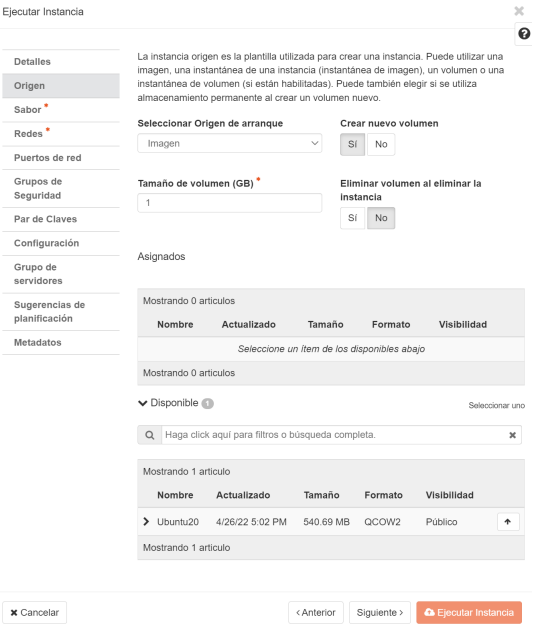


Figura 57: Origen de la instancia

La siguiente configuración revela sobre qué red va a colgar la instancia que se está creando. En nuestro caso, se crea una red por usuario, así que estará situada en dicha red (según la imagen habría que elegir la red: isabel.dominguez):

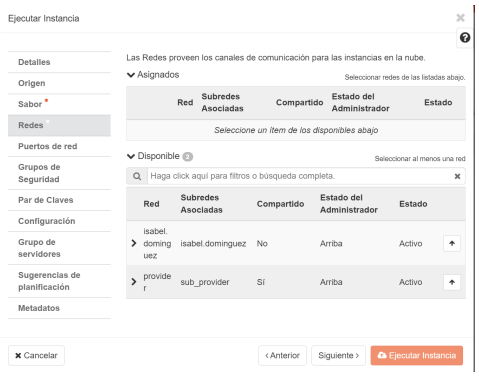


Figura 58: Red en la que está colgando la instancia

Ahora la máquina virtual se empezará a construir, durante este proceso se podrá pulsar sobre el botón *Asociar una IP flotante*. Esto no es más que una traducción *NAT* en el router *OpenStack* que hay entre la red personal que se ha creado para cada usuario y la red *provider* que es la encargada de la salida a internet de las instancias.

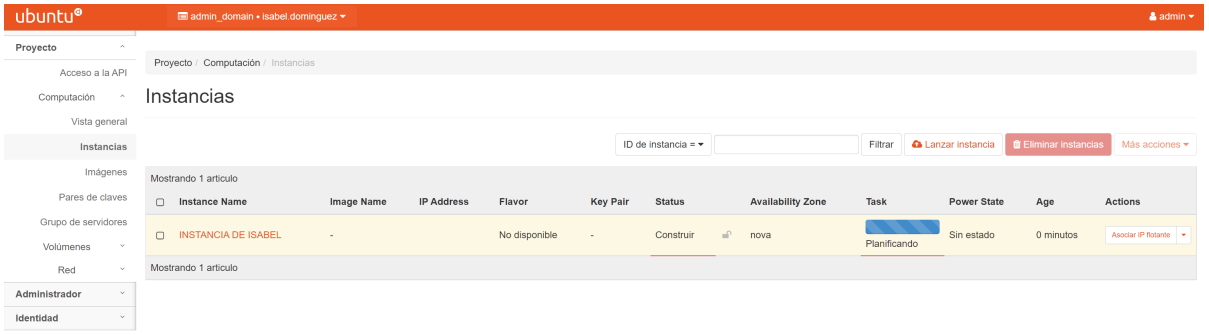


Figura 59: Creación de la instancia

Cuando se pulse, nos saltará una pantalla como la siguiente:

Figura 60: Asociación de la IP flotante

En el caso que nos toca, se accede a la instancia por medio de la IP privada: 10.0.2.131 ya que la IP privada 10.0.1.133 es la de la red de cada usuario (en nuestro caso la red llamada isabel.dominguez).

Para hacer accesible la instancia por *SSH* y dotarla de conexión a internet, se tiene que editar los grupos de seguridad. Cada instancia cuando es creada, tiene por defecto un grupo de seguridad llamado *default*. En este grupo se añaden las siguientes reglas:

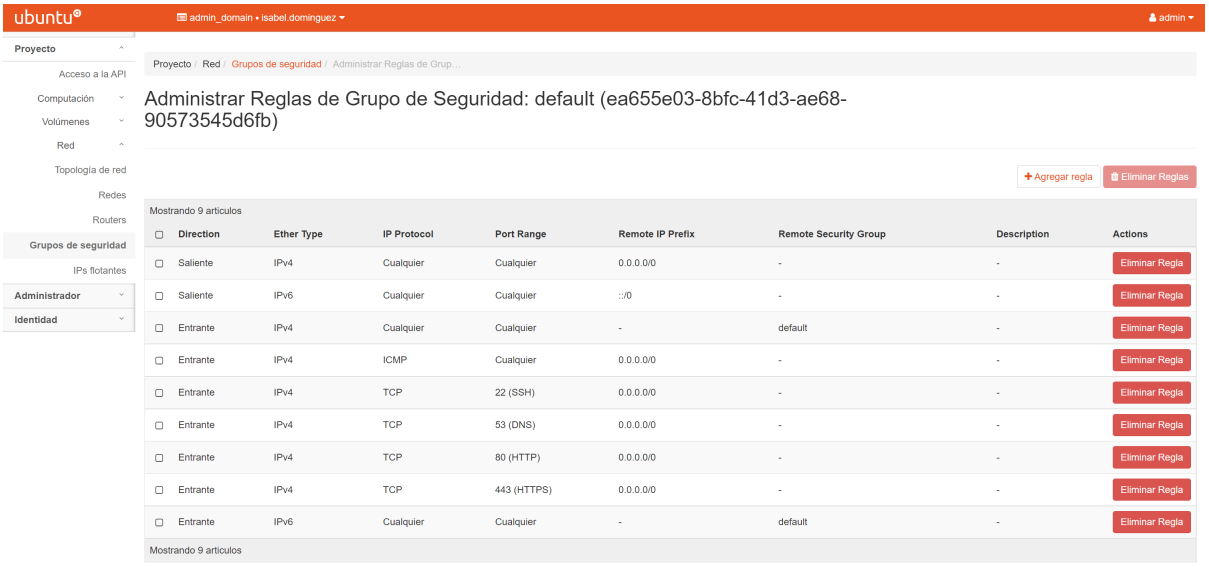


Figura 61: Edición de los grupos de seguridad

Como se puede observar se ha creado una regla que permite el tráfico entrante por el puerto 22 (puerto SSH), igual para los puertos 80 y 443 (puertos HTTP y HTTPS), otra para el puerto 53 (DNS) y por último se habilita el tráfico ICMP para pruebas de conexión de la instancia.

Una vez se han realizado todas las configuraciones descritas, nuestra instancia estará disponible para realizar cualquier experimento. Para comprobar que realmente el escenario que se ha montado está funcionando, se hace un *ping* desde el servidor *MaaS*, sobre la IP flotante de la máquina virtual que se acaba de levantar.

```
ping 10.0.2.131
```

Si la máquina responde al *ping*, nos encontraremos ante la situación que pongo en contexto en la imagen 52.

El siguiente objetivo será entonces, acceder mediante el puerto 22 (puerto SSH) a la instancia. Este es un punto crítico en la configuración ya que la red *provider* tiene que tener el *bridge* configurado de manera correcta. Este *bridge* por el que saldrán las instancias a internet, se ha configurado en el fichero *neutron.yaml* en concreto en las variables: *bridge-interface-mappings* y *ovn-bridge-mappings*. Si no nos hemos dado cuenta de este detalle, estas configuraciones se pueden cambiar mediante los siguientes comandos *JuJu*:

```
juju config ovn-chassis bridge-interface-mappings=br-ex:eth0.2
juju config ovn-chassis ovn-bridge-mappings=physnet1:br-ex
```

Nos situamos en el servidor *MaaS* y se accede por SSH a la instancia mediante el siguiente comando:

```
ssh -i private-key nombre-usuario@10.0.2.131
```

Si se tiene éxito nos encontraremos dentro del servidor. El objetivo aún no está completo porque se debe hacer accesible esa instancia desde la *VPN* de la universidad y no solo desde el servidor *MaaS*. Como se refleja en la imagen [52] hay que realizar una traducción de IP´s. Esta traducción de IP´s es del tipo *NAT*. Se dispone de diez direcciones, desde la 172.29.21.190 hasta la 172.29.21.200, estas direcciones serán asignadas a la interfaz con nombre *eth0* mediante el siguiente comando:

```
ip a add 172.29.21.190 dev eth0
```

Una vez que se han asignado las IP´s a la interfaz *eth0*, los comandos que nos hacen la traducción *NAT* entre la IP flotante de la instancia virtual (en el ejemplo es la IP 10.0.2.131) y la IP disponible en la *VPN* de la universidad son:

```
sudo iptables -t nat -A POSTROUTING -o eth0 -s 10.0.2.131 -j SNAT --to-source 172.29.21.190
sudo iptables -t nat -A PREROUTING -i eth0 -d 172.29.21.190 -j DNAT --to-destination 10.0.2.131
sudo iptables -A FORWARD -s 172.29.21.190 -j ACCEPT
sudo iptables -A FORWARD -d 10.0.2.131 -j ACCEPT
```

Se añade el siguiente *script* para realizar el *NAT-1-to-1* de todas las direcciones que van a ser usadas por las instancias:

```
#!/bin/bash
Nro_IPs=32
IP_Inicio=128
DIR_Base="172.23.54."
DIR_Base_priv="10.0.2."
IP_Fin=$IP_Inicio+$Nro_IPs
for (( contador=$IP_Inicio; contador<$IP_Fin; contador++ ))
do
  echo "ip a add "$DIR_Base$contador" dev eth0"
  ip a add $DIR_Base$contador dev eth0
  sudo iptables -t nat -A POSTROUTING -o eth0 -s $DIR_Base_priv$contador -j SNAT --to-source $DIR_Base$contador
  sudo iptables -t nat -A PREROUTING -i eth0 -d $DIR_Base$contador -j DNAT --to-destination $DIR_Base_priv$contador
  sudo iptables -A FORWARD -s $DIR_Base$contador -j ACCEPT
  sudo iptables -A FORWARD -d $DIR_Base_priv$contador -j ACCEPT
done
```

Para comprobar que la configuración es buena, se accede mediante *mobaXterm* (conectados a la *VPN* de la universidad) por el puerto 22, con nuestra clave creada anteriormente:

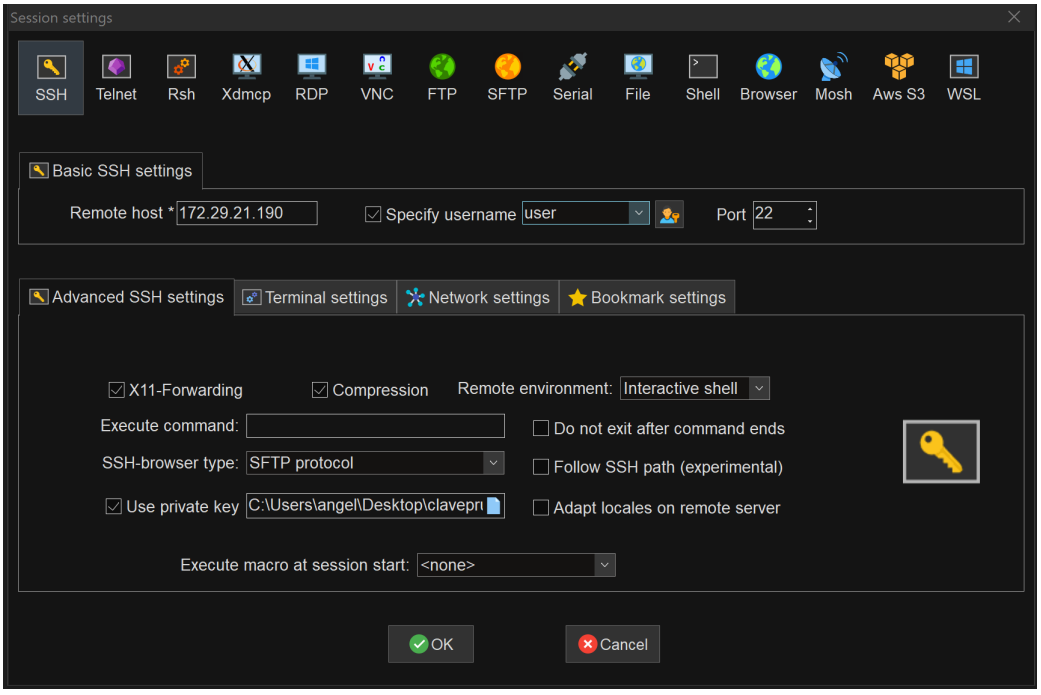


Figura 62: Acceso a la instancia desde la VPN de la UAH

En este momento la instancia estaría accesible desde la *VPN de la UAH* para realizar cualquier prueba. Por último, queda comprobar si la instancia tiene salida a internet. Para ello, se instala un servidor de apache con el siguiente comando (este comando debe ser lanzado dentro de la instancia):

```
sudo apt-get install apache2
```

Se reinicia y se comprueba el estado del servidor con el comando *systemctl*.

```
sudo systemctl restart apache2
sudo systemctl status apache2
```

Si todo va bien (la salida del comando anterior es *Running*) en el puerto 80 de la dirección 172.29.21.190 se tiene el servidor disponible. Se busca en cualquier navegador: *http://172.29.21.190:80/html* y se visualiza la página ejemplo de bienvenida de *Apache*.

Figura 63: Página de bienvenida Apache

Con esto finalizaría la prueba de concepto. Cabe resaltar, que esta configuración habría que realizarla en cada usuario que quiera desplegar una instancia de la nube y gastaría una dirección IP de las disponibles en la *VPN* de la universidad.

Para eliminar usuarios de la nube se usa el siguiente *script*:

```
#!/bin/bash
source ~/openstack-bundles/stable/openstack-base/openrc
month=$(date +%m)
year=$(date +%Y)
fecha=$year$month
openstack server list --all-projects | grep 192 | awk '//{print $2}' >> instancias.txt
while IFS= read -r instancia
do
   #echo "$instancia"
   IFS='\n' read -ra ADDR <<< "$instancia"
   echo "Eliminando instancia: ${ADDR[0]}"
   comando="openstack server delete --wait  ${ADDR[0]}"
   comando2="$($comando)"
   echo  "$comando2"
done < instancias.txt
openstack floating ip list | grep 192 | awk '//{print $4}' >> floatingip.txt
while IFS= read -r floatingip
do
   #echo "$floatingip"
   IFS='\n' read -ra ADDR <<< "$floatingip"
   echo "Liberando IP Flotante: ${ADDR[0]}"
   comando3="openstack floating ip delete ${ADDR[0]}"
   comando4="$($comando3)"
   echo "$comando4"
done < floatingip.txt

while IFS= read -r correo
do
   # echo "$correo"
   IFS='@' read -ra ADDR <<< "$correo"
   echo "Eliminando network: ${ADDR[0]}"
   comando5="openstack router remove subnet ${ADDR[0]} ${ADDR[0]}"
   comando6="$($comando5)"
   echo "$comando6"
   comando7="openstack subnet delete ${ADDR[0]}"
   comando8="$($comando7)"
   echo "$comando8"
   comando9="openstack network delete ${ADDR[0]}"
   comando10="$($comando9)"
   echo "$comando10"
   echo "Eliminando router: ${ADDR[0]}"
   comando11="openstack router delete ${ADDR[0]}"
   comando12="$($comando11)"
   echo "$comando12"
   echo "Eliminando usuario: ${ADDR[0]}"
   comando13="openstack user delete ${ADDR[0]}"
   comando14="$($comando13)"
   echo "$comando14"
```

```
    echo "Eliminando proyecto: ${ADDR[0]}"
    comando15="openstack project delete ${ADDR[0]}"
    comando16="$($comando15)"
    echo "$comando16"
done < correos.txt
rm instancias.txt
rm floatingip.txt
```

9.5. Procedimiento de encendido y apagado de la nube

Dado el uso docente de la nube, en vacaciones se debe apagar y volver a encender durante el curso. Se podrá consultar mas información en la siguiente referencia sobre este apartado. [6].

9.5.1. Encendido

Primero se encenderá el servidor *MaaS*, se comprobará que la configuración de red es correcta y se correrá el *script* que contiene los comandos *iptables* iniciales.

Segundo, se encenderá el servidor *JuJu*. Comprobar que arranca via PXE y si no con F10 arrancarlo desde el disco duro. Se probará algún comando fácil (*juju status*, por ejemplo) para comprobar que se ha desplegado correctamente el controlador. Esto puede tardar mas de seis horas en funcionar. En caso de no funcionar habrá que redesplegar todo a partir del despliegue del servidor *Juju* a partir del comando *juju bootstrap*.

Tercero, se encenderán los nodos y se comprobará que se levantan correctamente, como antes asegurarse que arrancan vía *PXE* (puede tardar unos minutos) o disco duro. El siguiente paso viene recogido en el apartado de recuperación de la nube [9.1]

Para finalizar, se comprobará el estado de las aplicaciones con *juju status*.

9.5.2. Apagado

El procedimiento de apagado de las aplicaciones viene recogido en la siguiente referencia [6].

Según lo recoge el apartado mencionado anteriormente [6], se pueden detener las aplicaciones mediante el agente *JuJu* con el siguiente comando:

```
juju run-action --wait <unit> <action>
```

Sustituiremos *unit* por la unidad que se desea parar o reiniciar y *action* por *pause* (pausar) o *resume* (reanudar). Lo ideal para un detenimiento de la nube sería parar todas las aplicaciones y después apagar los nodos de cómputo.

El apagado de las máquinas que componen la nube es indiferente con respecto al orden. Por seguir un patrón, se apagan primero los nodos de cómputo y el servidor *JuJu*, seguidamente el servidor *MaaS*.

Se debe tener en cuenta que el apagado forzado de las máquinas, supone un impacto en las aplicaciones que corren en los nodos de cómputo. Por ejemplo, si se apaga una unidad en la que corre *ceph-osd* (aplicación encargada del almacenamiento de la nube), los datos que maneja dicha aplicación pasan a otra unidad de cómputo. Esta casuística se repite en las demás aplicaciones.

10. Monitorización de los nodos de cómputo

10.1. Instalación de las aplicaciones

Para la monitorización se va a usar el entorno de la aplicación *Grafana*. Esta aplicación no es más que una pizarra que muestra métricas que recoge de otra aplicación llamada *Prometheus*.

Lo primero que se va a hacer es configurar *Prometheus* para obtener métricas de los nodos de cómputo. El escenario contra el cual se van a realizar las pruebas dispone de cuatro nodos de cómputo más la controladora *JuJu*.

Se instala la aplicación *Prometheus* en el nodo controlador *JuJu*. La ubicación de esta aplicación en el nodo *JuJu*, es por repartir la carga de tráfico de la nube ya que en los nodos de cómputo corren todas las aplicaciones del modelo *Openstack*. Se accede por *SSH* al nodo *JuJu* y se ejecutan los siguientes comandos:

```
ssh ubuntu@10.0.1.11
sudo apt-get install prometheus prometheus-node-exporter prometheus-pushgateway prometheus-alertmanager -y
sudo systemctl stop prometheus
```

Como se puede observar, también se instala *node-exporter-prometheus*, *prometheus-alert-manager* y *prometheus-push-gateway*, este *Software* nos proporciona las métricas que nos mostrará el *Dashboard de Prometheus* y las alarmas que se configuren en el entorno.

En este punto, se va a realizar la configuración de la aplicación *Prometheus*. Para ello, se entra en el siguiente fichero situado en la ruta */etc/systemd/system/multi-user.target.wants/prometheus.service* y se añade la siguiente línea:

```
ExecStart=/usr/bin/prometheus --web.enable-lifecycle $ARGS
```

En este momento se lanza la aplicación con los siguientes comandos:

```
sudo systemctl daemon-reload
sudo systemctl start prometheus
sudo systemctl status prometheus
```

Ahora, si se está en un ordenador conectado a la red de la nube, se puede acceder al *Dashboard de Prometheus*, que se ha alojado en el servidor *JuJu* (http://10.0.1.X:9100). Se visualiza algo como esto:

Figura 64: Página de bienvenida *Prometheus*

Una vez configurado el servidor que aloja *Prometheus* se instala una unidad de *node-exporter-prometheus* en cada nodo que se quiere monitorizar. Instalamos y activamos esta aplicación. Los comandos que realizan esto son:

```
sudo apt install prometheus-node-exporter -y
sudo systemctl start prometheus-node-exporter
sudo systemctl enable prometheus-node-exporter
sudo systemctl status prometheus-node-exporter
sudo ufw allow from 10.0.1.12 to any port 9100
```

Como se puede observar en los comandos anteriores, primero se arranca el exportador de métricas, después se activa el servicio y seguidamente se ve el estado de la aplicación. El último comando permite el tráfico de datos por el puerto 9100 de la máquina que soporta el exportador de métricas. Esto último es para que le lleguen los datos de las métricas a *Prometheus* (servidor *JuJu*).

Ahora, para que todos los nodos manden sus métricas al servidor donde corre *Prometheus* (en nuestro caso en el servidor *JuJu*), se debe realizar la siguiente configuración en el nodo que aloja *Prometheus*. La ruta del archivo de configuración es */etc/prometheus/prometheus.yml*:

```
global:
  scrape_interval: 1s
  evaluation_interval: 1s
alerting:
  alertmanagers:
  - static_configs:
    - targets: ['10.0.1.11:9093']
scrape_configs:
  - job_name: 'prometheus'
    scrape_interval: 1s
    scrape_timeout: 1s
    static_configs:
      - targets: ['10.0.1.11:9090']
  - job_name: 'nodes'
    scrape_interval: 1s
    scrape_timeout: 1s
    static_configs:
      - targets: ['10.0.1.11:9100', '10.0.1.12:9100', '10.0.1.13:9100', 10.0.1.14:9100, 10.0.1.15:9100]
```

Como se puede observar en el anterior fichero de tipo *yaml*, se configuran las *IP´s* donde corren los exportadores de métricas y el propio servidor de *Prometheus* con sus respectivos puertos.

Una vez realizada esta configuración se dispone del servidor de *Prometheus* para mostrar métricas. No se usará el *Dashboard de Prometheus* para mostrar las medidas, y aquí es donde entra en juego *Grafana*.

Para instalar *Grafana*, se usa el agente *JuJu*:

```
juju deploy grafana --to lxd:3
```

Como se puede ver, se instala la aplicación en un contenedor de la máquina tres.

Una vez la aplicación esté activa, se accede mediante cualquier navegador al *Dashboard de Grafana* y nos pedirá unas credenciales para entrar a la pizarra. El usuario será *admin* y la contraseña se consigue con el siguiente comando *JuJu*:

```
juju run-action --wait grafana/0 get-admin-password
```

Para acceder a *Grafana* se pone en nuestro navegador (tras hacer una traducción *NAT* con una dirección accesible desde la VPN):

```
http://172.29.21.200
```

El aspecto de la aplicación en primera instancia es el siguiente:

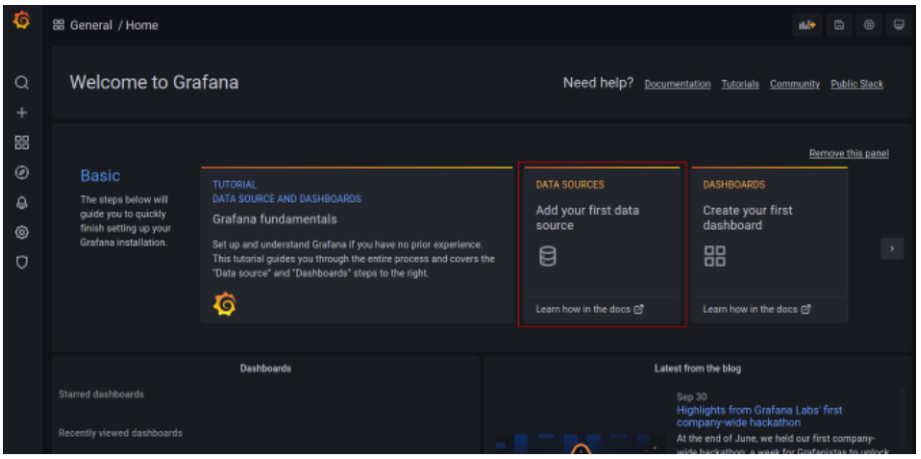


Figura 65: Página de bienvenida *Grafana*

Se añade el servidor *Prometheus* que nos dará las métricas, de la siguiente manera:

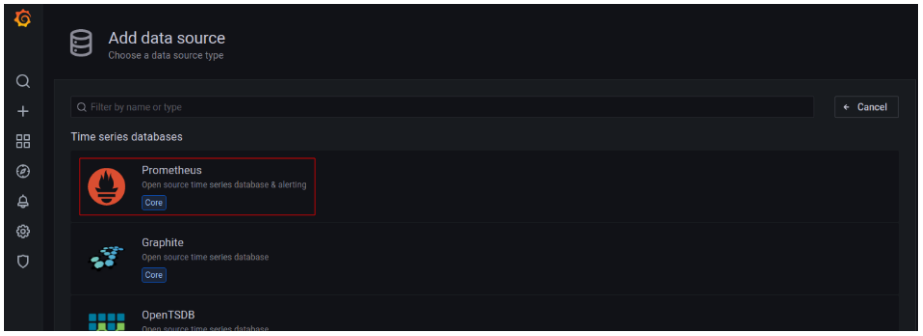


Figura 66: Conexión *Grafana Prometheus*

Ahora se añade la *URL* de donde se van a recoger los datos de *Prometheus*. En nuestro caso, es la dirección del servidor *JuJu* (10.0.1.11) puerto 9090.

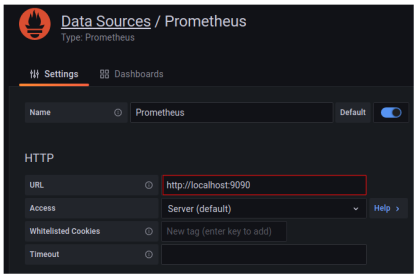


Figura 67: URL conexión *Prometheus Grafana*

Ahora se importa uno de los *Dashboard* que viene preparados, en concreto el número 1860. Para añadir un *Dashboard* se pulsa sobre el botón + que aparece en la ilustración [65] y en el desplegable se pulsa sobre *Import*. Nos saldrá una pantalla con el siguiente aspecto:

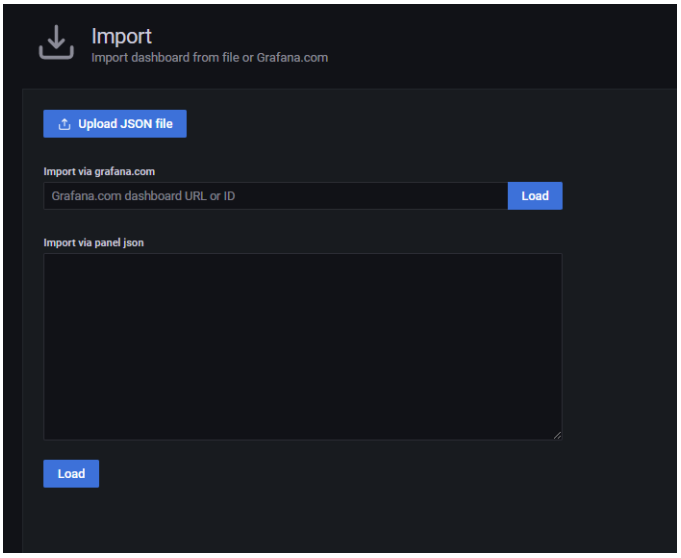


Figura 68: Añadir un *Dashboard Grafana*

Se pone el siguiente enlace en el cuadro con titulo *import via grafana.com*:

```
https://grafana.com/grafana/dashboards/1860
```

Se está añadiendo una pizarra que ya está subida a un repositorio que posee la aplicación *Grafana*. Esta pizarra muestra métricas de los nodos que se han monitorizado en la nube:

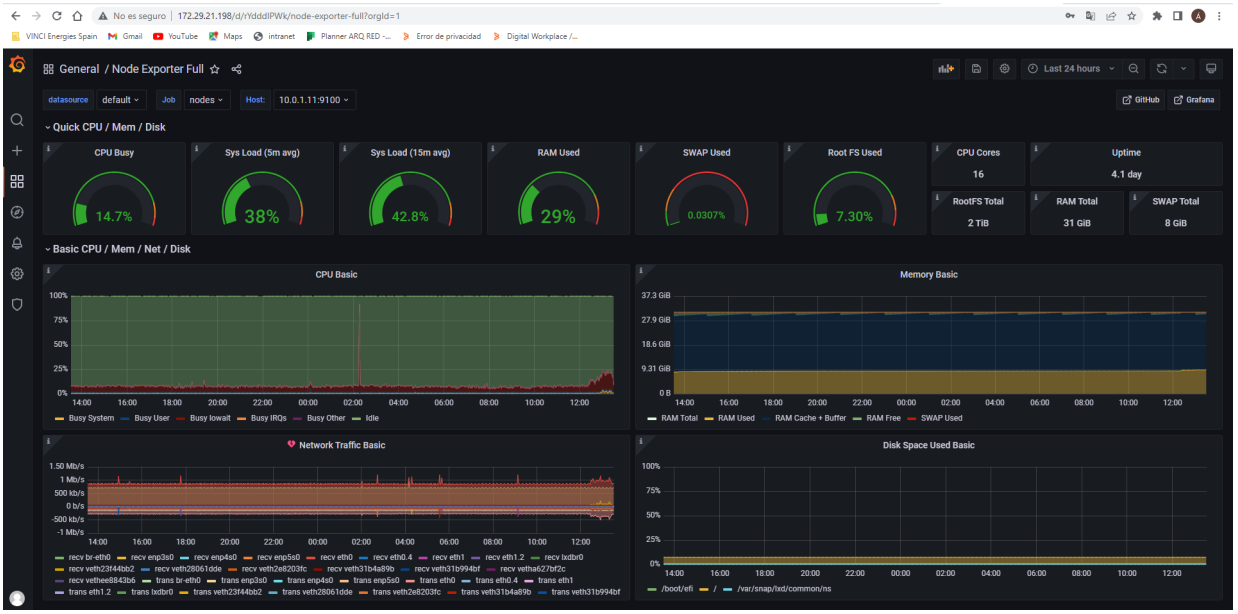


Figura 69: Métricas de los nodos de cómputo

La siguiente figura, muestra el flujo de datos que intercambian las aplicaciones que se han presentado. La aplicación *Prometheus Node Exporter* reúne la métricas de cada nodo, seguidamente *Prometheus* agrupa

en sí todas estas medidas y las pasa a *Grafana* que es donde se visualizan los datos.

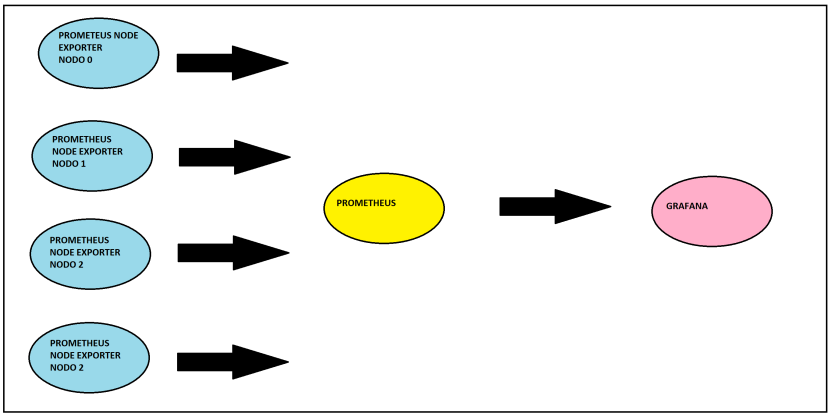


Figura 70: Esquema de funcionamiento de la monitorización del sistema

Como se puede observar en esta pizarra de la figura [69] vienen varias pestañas con métricas de todo tipo (RAM, tráfico de red en las interfaces de los nodos, CPU en uso de los nodos, almacenamiento, características hardware de los nodos, etc).

10.2. Pruebas de estrés a los servidores

Llegados a este punto, lo que se hace es generar pruebas para ver si las métricas que se están visualizando en esta pizarra se corresponden con la realidad. Estas pruebas de estrés se realizarán para monitorizar principalmente la CPU, la RAM y el tráfico de red que soportan las interfaces que se tienen en nuestro sistema.

Se instala en cualquiera de los nodos la herramienta para estresar la CPU. Desde *MaaS*, se hace *SSH* a cualquiera de los nodos y se realiza lo expuesto anteriormente:

```
ssh ubuntu@10.0.1.XX
sudo apt-get install stress
```

La *CPU* en uso viene monitorizada en las siguientes gráficas:

Figura 71: Porcentaje de CPU ocupada de un nodo

Figura 72: Gráfica de uso de la CPU de un nodo

Se somete a una prueba de estrés de *CPU* al nodo en cuestión y se visualizan como cambian las gráficas que se han mostrado con el nodo en reposo [71] y [72]. El comando para proporcionar estrés a la *CPU* es:

```
sudo stress --cpu  8 --timeout 20
```

Este comando de estrés, lanza una prueba de 8 procesos de CPU durante 20 segundos. Estos procesos, mantienen ocupada a la *CPU*. La salida de las gráficas se muestra en las figuras 73 y 74 .

Figura 73: Porcentaje de uso de la *CPU* después de estrés

Como se puede observar, cuando se usa la herramienta de estrés, se comprueba que queda plasmado un cambio en las gráficas, por lo que se puede afirmar que realmente se está monitorizando el uso de la *CPU*.

Otro aspecto a tener en cuenta a la hora de monitorizar el sistema que se ha creado, es el tráfico de red que van a recibir y transmitir las instancias que crean los usuarios *Openstack*.

En el capítulo de instalación de nuestro escenario [9.4] se crea una red llamada *provider* de la que cuelgan todas las redes que van a usar los usuarios para levantar sus instancias. El tráfico de esa red es el que nos interesa monitorizar. La configuración propuesta es que dicha red sea la 10.0.2.0/24 es decir, la *VLAN eth0.2* de cada nodo. Es ahora, cuando toma sentido el despliegue de un servidor *Apache* en el capítulo mencionado anteriormente [9.4] ya que la prueba de estrés se lanzará contra este servidor.

Nos situaremos en el servidor *MaaS* y se instala la herramienta habitual para estresar servidores Apache:

```
sudo apt-get install apache2-utils
```

Figura 74: Gráfica de uso de la *CPU* de un nodo después de estrés

Cabe resaltar que en este punto, se tiene una instancia virtual que está accesible mediante la *IP flotante* 10.0.2.131 y en la cual cual hay instalado un servidor Apache. La gráfica en estado de reposo de la red *provider* que se ha comentado es la siguiente:

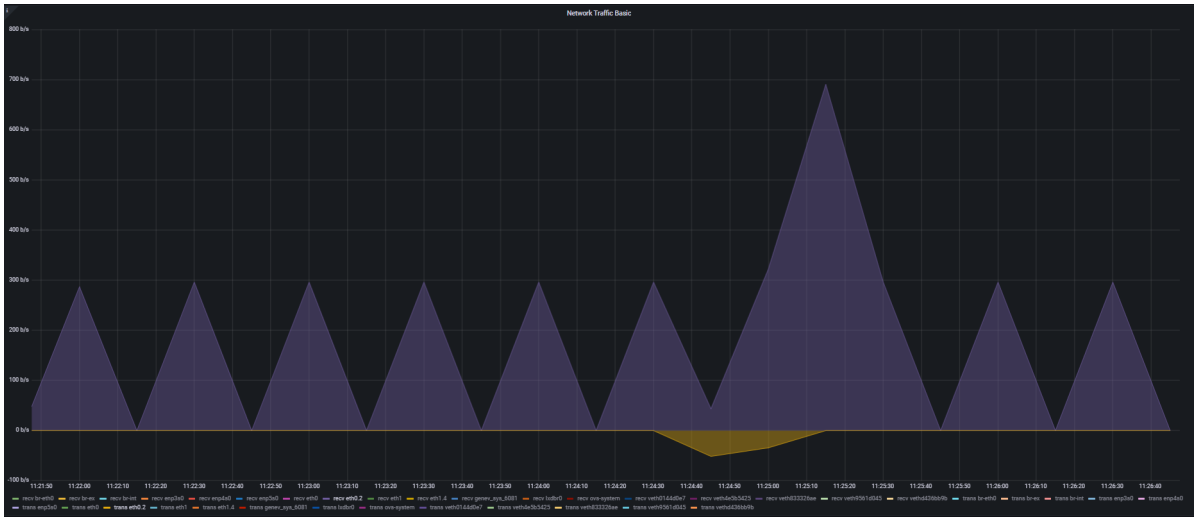


Figura 75: Gráfica de uso de la red de un nodo

Como se puede observar el tráfico recibido en el servidor correspondería con la gráfica con valores de bits/segundo positivos (eje de ordenadas) y el tráfico transmitido corresponde con valores en el eje de ordenadas negativos.

El comando que provoca estrés en este servidor Apache es el siguiente:

```
ab -r -n 200000 -c 1000 http://10.0.2.131:80/
```

El comando lo que hace es mandar un total de 200000 peticiones de las cuales 1000 son concurrentes a la dirección *IP* en la que corre el servidor.

La salida de la gráfica después de lanzar esta prueba es la siguiente:

Figura 76: Gráfica de uso de la red de un nodo después de estrés

Como se puede observar el eje de ordenadas tanto el tráfico transmitido (gráfica amarilla) como recibido (gráfica morada) sube en razón de Mbits/segundo. Con esto se prueba que se está monitorizando las peticiones que se realizan al servidor Apache, es decir el tráfico que transmiten y reciben las posibles instancias que estén activas en el interfaz *eth0.2* del nodo en cuestión.

10.3. Configuración de las alarmas

Grafana dispone de un servicio de alarmas. Se va configurar la gráfica que nos muestra los datos que hay en la red para que salte una alarma cuando el trafico medio de todas la interfaces supere cierto umbral. Al superar dicho umbral, lo que sucederá es que se configurará una dirección de correo electrónico para que se genere un email.

Para que esto funcione, hay que realizar una configuración en el servidor en el que corre *Grafana*. El archivo de configuración se encuentra en el directorio */etc/grafana* y se llama *grafana.ini*. Se descomentan las siguientes lineas y se añaden los siguientes datos:

```
[smtp]

enabled = true
host = smtp.office365.com:587
user = angel.dominguezf@edu.uah.es
password = password
;cert_file =
;key_file =
skip_verify = true
from_address = angel.dominguezf@edu.uah.es
```

Figura 77: Fichero de configuración grafana.ini

Como se puede observar se usa el servidor de correo de *Microsoft*, el usuario que mandaría el correo es mi correo personal y a quien iría dirigido es a mi mismo correo.

Para configurar la alarma se va a la siguiente pantalla de *Grafana*:

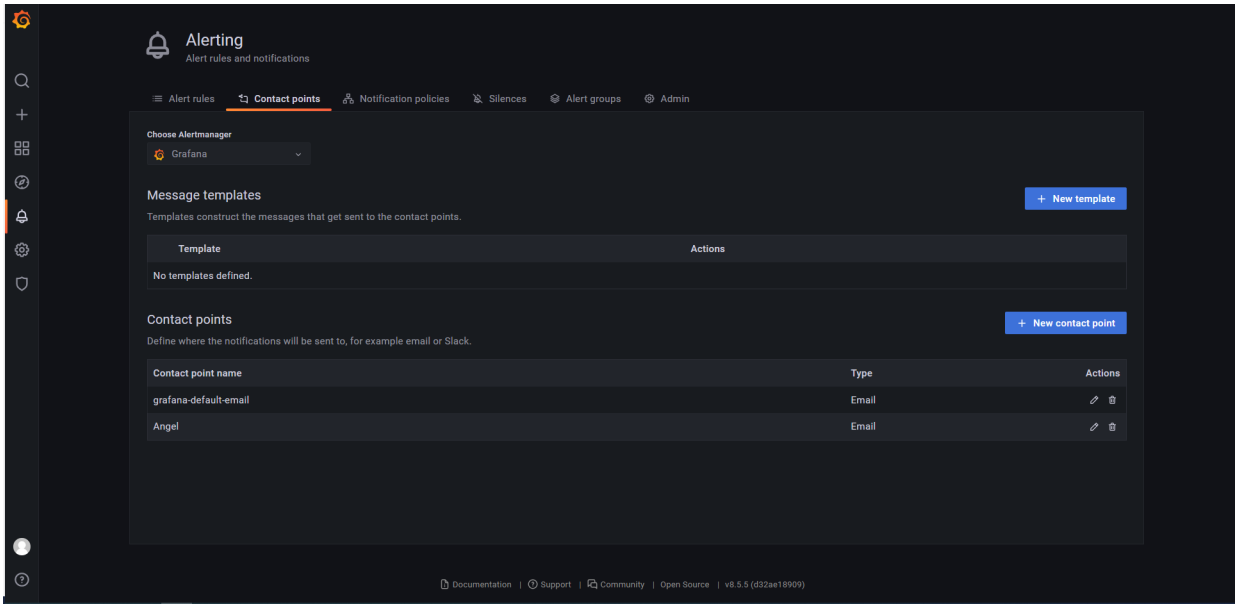


Figura 78: Configuración del correo para alarmas

Como divisamos en la pantalla [78], se configura el correo donde se quiere que se notifiquen las alarmas.

Ahora nos dirigiremos a la gráfica en la cual se quiere poner la alarma, según la figura 69 y se pulsa sobre sobre la propia gráfica en la opción *Edit*.

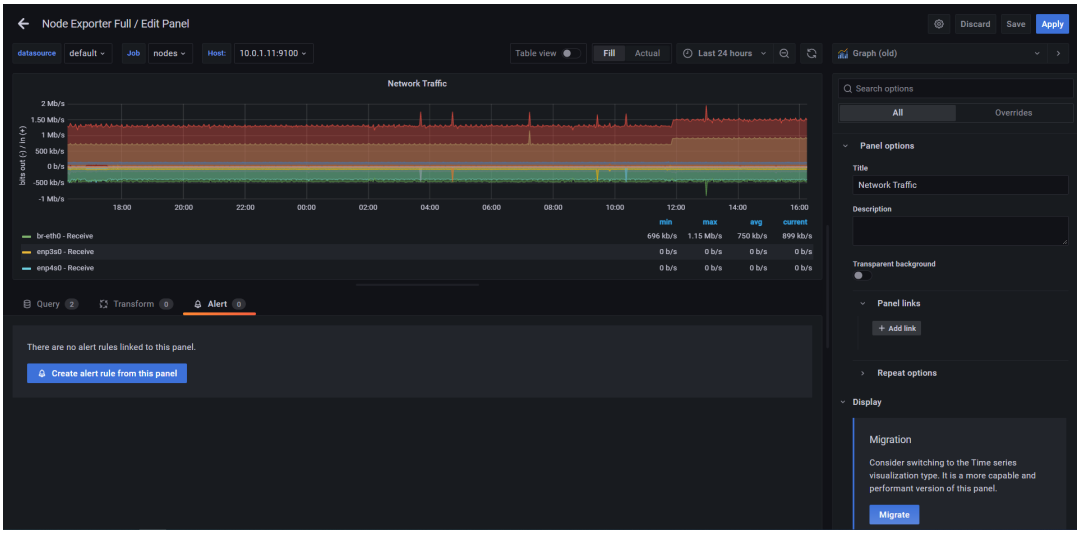


Figura 79: Configuración del correo para alarmas

Se pulsa la opción *Create alert rule for this panel* y se configura la alarma. El aspecto en la aplicación es el siguiente:

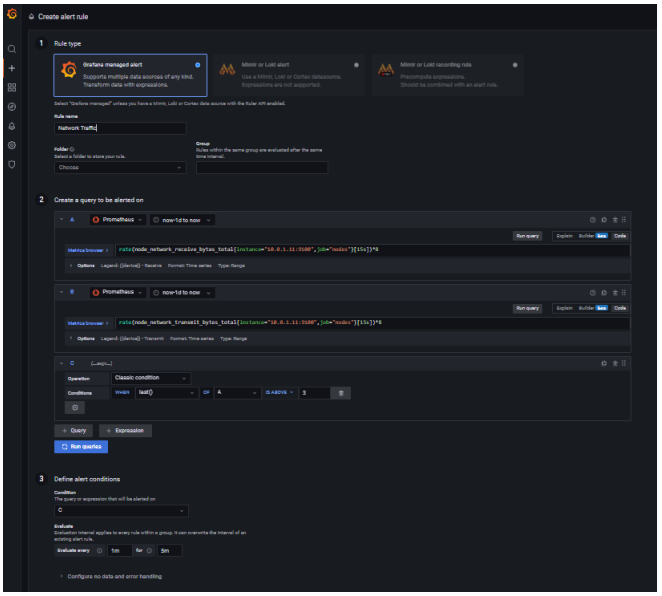


Figura 80: Configuración de las alarmas

Si nos fijamos en la imagen anterior [80], en los apartados *A y B* viene refleja la propia métrica procedente de *Prometheus* y en el *C* la configuración de la alarma. En el campo *operation* elegimos la opción *Classic Condition* y en el campo *Condition*: *When avg() of A is above: [UMBRAL]*. Este último campo nos indica que cuando superemos la variable [UMBRAL] saltará la alarma.

Se ha comprobado mientras se estresaba el servidor que la alarma funciona. El aspecto del correo que llega a la cuenta configurada es el siguiente:

Figura 81: E-mail de alarmas

11. Conclusiones y trabajo futuro

11.1. Conclusiones

La principal conclusión a la que se puede llegar después de terminar el proyecto es que:

"*Es posible crear un sistema que haga las funciones de una nube privada para usuarios de la universidad*"

Cabe resaltar, que este sistema se ha formado con unas aplicaciones cuyo *software* y manual de despliegue está accesible para cualquier usuario de internet.

La tecnología de *Cloud Computing* está a la orden del día en el ámbito empresarial. No solo la tecnología, si no también la arquitectura de la propia red, te da nociones de cómo funcionan este tipo de sistemas en un entorno de producción real.

11.2. Trabajo futuro

Un punto de fallo del proyecto de cara a poner el entorno en producción, es el servidor *MaaS*. Este servidor a parte de tener la capacidad de añadir o quitar recursos a la nube es el enlace entre la red de internet y nuestra nube. Todo el tráfico de datos que transita la nube hacía el exterior tiene que pasar necesariamente por este equipo. En caso de que este equipo caiga, nuestro servicio se vería tremendamente afectado. Como solución a este problema, se propone poner un nuevo servidor que también contenga la lógica de *MaaS*. Este servidor en redundancia, tendría que convivir con el servidor que ya se tiene y además, de cara a repartir el tráfico, sería interesante poner algún balanceador de carga en el sistema. Con esta mejora, se conseguiría una mejor experiencia para los usuarios que están consumiendo recursos de la nube.

El servidor *MaaS*, se ha probado en un entorno en el que se desplegaban cuatro nodos de cómputo y un servidor que hacía de *JuJu*, en total en ese caso se hace un reconocimiento de cinco máquinas que conforman la nube.

Con respecto al apagado y encendido en los nodos de cómputo de la nube, se ha configurado que sea de forma manual. Esto implica que en cada despliegue hay que pulsar el botón de encendido del servidor. Para un despliegue masivo de nodos habría que utilizar el encendido y apagado remoto. Esto no se ha realizado porque había algunos PC´s antiguos que no soportaban esta opción. En el futuro sería conveniente reemplazar estos nodos.

Con respecto al apagado de la nube redactado en el apartado [9.5.2], no se ha probado este método, que consiste en parar primero las aplicaciones y después volver a iniciarlas. Se sospecha que si se hace así, no haría falta recurrir al apartado [9.1] para volver a levantar las aplicaciones que conforman la nube.

Una posible linea de trabajo futuro de la nube sería intentar que fuera de tipo híbrido. Habría que estudiar los diferentes proveedores de este tipo de recursos, por ejemplo, *Amazon, Azure, Google Cloud...* y en caso de querer reconocer nodos de cómputo, lo hiciera pero con máquinas de las entidades mencionadas.

Otro punto de mejora en la parte de monitorización, es que las métricas que nos proporciona *Grafana* fueran más sofisticadas. Se podría poner una gráfica que sacara los diez nodos que más trafico soportan en sus interfaces de red, entre otras. No es viable en un despliegue masivo de servidores, mirar de uno en uno las métricas, tal y como está desarrollado en este proyecto. También sería conveniente la visualización simultánea de métricas de varios nodos.

La parte de las alarmas, por falta de tiempo, no se ha depurado lo suficiente. Se puede afinar mucho más el umbral de las alarmas y comprobar su funcionamiento de manera mas dedicada.

Por último, para reducir el número de nodos, sería muy interesante virtualizar el servidor *JuJu* en una máquina virtual de *VMWare o Virtual Box*, en un servidor de la nube o en el mismo nodo *MAAS*. Esto sería de cara a ahorrar servidores.

12. Presupuesto

Durante el desarrollo de este proyecto, se ha invertido un tiempo estimado que se va a dividir en varias fases. Cada una de las fases, supone un esfuerzo que se cuantifica en horas.

- Estudio, implementación y utilización del *Software* de *MaaS*. En esta primera fase invertimos el tiempo en el estudio y la instalación de la aplicación. El estudio es a conciencia para diseñar el escenario que se describe en este proyecto.
- Estudio, implementación y utilización del *Software* de *JuJu*. En esta fase, se estudia la integración de *JuJu* con *MaaS*. Además se estudian todas las funcionalidades que nos proporciona el software en cuestión.
- Estudio, implementación y utilización del *Software* de *Openstack*. En este apartado, se despliegan todas las aplicaciones que van conformar nuestro modelo *OpenStack*.
- Modelo que prueba el escenario. Se crea un escenario que prueba el sistema y se comprueba que todo funciona según lo esperado.
- Monitorización y pruebas de test del sistema. En este apartado se muestran las características hardware de los nodos y el tráfico de red que soportan.
- Elaboración de la memoria/manual de instalación.

12.1. Cuantificación del esfuerzo

Se realiza la siguiente estimación de horas:

FASE	HORAS
Estudio, implementación y utilización del *Software* de *MaaS*.	80
Estudio, implementación y utilización del *Software* de *JuJu*.	80
Estudio, implementación y utilización del *Software* de *Openstack*.	120
Modelo que prueba el escenario.	40
Monitorización y pruebas de test del sistema.	20
Elaboración de la memoria/manual de instalación.	30
TOTAL	370

Referencias

[1] Marin Fotache y Marius-Iulian Cluci. «Big Data Performance in Private Clouds. Some Initial Findings on Apache Spark Clusters Deployed in OpenStack». En: *2021 20th RoEduNet Conference: Networking in Education and Research (RoEduNet)*. IEEE. 2021, págs. 1-6.

[2] «https://bugs.launchpad.net/charm-mysql-innodb-cluster/+bug/1917332». En: 2022.

[3] «https://discourse.charmhub.io/t/failed-to-connect-to-mysql-but-vault-is-unsealed-and-mysql-is-ready-openstack-base/5105/19». En: 2022.

[4] «https://discourse.charmhub.io/t/vault-hook-failed-start/4729/3». En: 2022.

[5] «https://docs.openstack.org/project-deploy-guide/charm-deployment-guide/wallaby/». En: 2022.

[6] «https://docs.openstack.org/project-deploy-guide/charm-deployment-guide/wallaby/app-managing-power-events.html». En: 2022.

[7] «https://www.salicru.com/sps-2000-one-1.html». En: 2022.

[8] Hai Jin y col. «Cloud types and services». En: *Handbook of cloud computing*. Springer, 2010, págs. 335-355.

[9] Pedro Ignacio Santiago Pastelero. «Diseño e implementación de funcionalidades de gestión de recursos en la nube». En: 2020, págs. 1-200.

[10] Pankaj Sareen. «Cloud computing: types, architecture, applications, concerns, virtualization and role of it governance in cloud». En: *International Journal of Advanced Research in Computer Science and Software Engineering* 3.3 (2013).

Universidad de Alcalá
Escuela Politécnica Superior