**Documentación Infraestructura Virtual**

1. **OSM**

* Instalación y Configuración:

Para el Sprint 1 y 2 de Assurance se utilizó la el release 6.0 de OSM cuyo procedimiento de instalación se encuentra en el enlace: [*https://osm.etsi.org/wikipub/index.php/OSM\_Release\_SIX*](https://osm.etsi.org/wikipub/index.php/OSM_Release_SIX). Teniendo en cuenta que cuando se realizan de manera simultanea diferentes procedimientos (ej.: Instanciación, borrado, escalado, etc.), se evidenció aumento en el uso de recursos de la maquina, se aumentaron los requerimientos:

Imagen Base: Ubuntu18.04 Bionic

CPU: 4 vCPUs, Memory: 16 GB RAM, Storage: 50GB

Luego de realizar la conexión con la VIM y con el fin de enviar comandos de Openstack en la maquina de OSM se sugiere ejecutar el siguiente comando: *sudo apt-get install -y python-openstackclient* (para esto se deben enviar las variables de entorno de la VIM).

* Descriptores

A continuación los descriptores utilizados para el Sprint 1 y 2 de assurance:

1. **Openstack**

* Instalación y Configuración:

Para el Sprint 1 y 2 de la PoC assurance se utilizo la versión:

| **OpenStack Release** | **OpenStackClient Release** |
| --- | --- |
| Train | 4.0.0 |

La instalación se realizó en GCP utilizando los siguientes requerimientos

Imagen Base: Ubuntu18.04 Bionic

CPU: 16 vCPUs, Memory: 64 GB RAM, Storage: 100GB

Se debe crear una VPC diferente para cada nic, en este caso se requieren mínimo dos, una para la gestión de la maquina en GCP y otra para la red de gestión (mgmt) que conectará las VM a OSM. Adicionalmente la instancia GCP que va a correr el Openstack debe tener habilitado el IP FORWARDING. Esto se debe habilitar al momento de crear la instancia, ya que después de ser creado no es posible activarlo. Esta funcionalidad se activa al editar la interfaz por defecto (nic0) de la instancia (Reenvío de IP). También debe habilitarse el tráfico http. La instalación se realiza de acuerdo al procedimiento del enlace: [*https://docs.openstack.org/devstack/latest/*](https://docs.openstack.org/devstack/latest/)*.*

Una vez instalado se debe crear una red publica tipo *Physical Network* con el mismo direccionamiento de la red VPC de la nic1 la cual se utilizará para gestionar las VM desde OSM, el gateway debe ser la IP de la nic1 de la VM de Openstack. De igual forma se editan los *Security groups* para permitir ICMP y SSH. En la instancia de Openstack se configuran las interfaces para que tengan comunicación con redes externas:

*#subir el br-ex en modo promiscuo*

*sudo ifconfig br-ex promisc up*

*#asignar el puerto ensX al switch con la IP <ip> del host (ensX=nic1) y la red VPC.*

*sudo ovs-vsctl add-port br-ex ensX & sudo ifconfig br-ex <ip> netmask <mascara VPC>*

*#borrar la ip del ensX*

*sudo ip addr del <ip>/32 dev ensX*

*#reiniciar servicios*

*sudo systemctl restart devstack@\**

Después de tener conectividad entre el host de Openstack y las instancias, debemos habilitar la conectividad entre las instancias y redes externas (OSM). Para esto se debe habilitar el enrutamiento en el host. En el la instancia de openstack:

*#Editar el archivo /etc/sysctl.conf*

*vi /etc/sysctl.conf*

*#Descomentar la línea:*

*net.ipv4.ip\_forward=1*

Adicional a esto se habilita el siguiente NAT:

*sudo iptables -t nat -A POSTROUTING -o br-ex -j MASQUERADE*

En el Shell GCP se debe enrutar todo el trafico hacia la VPC de gestión de las maquinas configurando una ruta estática. Ej:

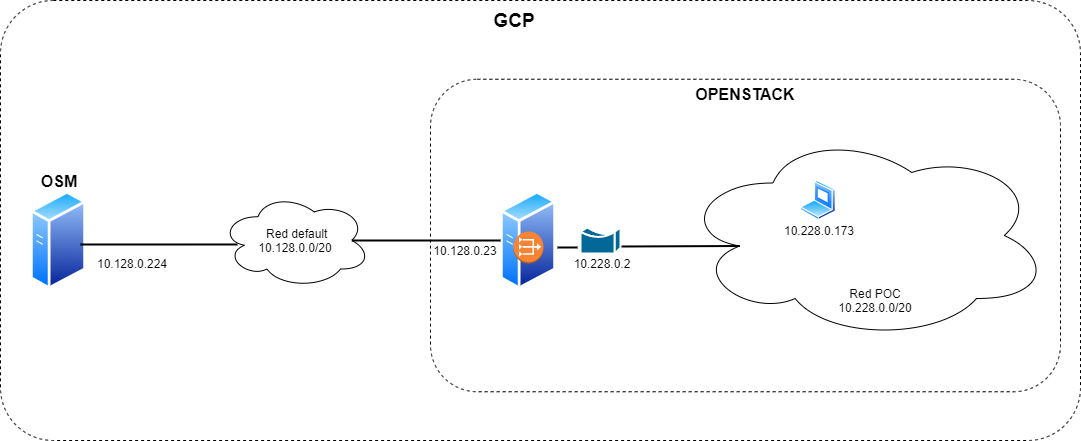
*gcloud compute routes create openstackInstances \*

*--destination-range=<redVPC>/<mascara de red> \*

*--network=default \*

*--next-hop-address=<ip>*

Luego de esta configuración se tendrá una topología como la siguiente:



Debido a que el Openstack es una VM (instancia) de GCP, el servicio metadata es “gobernado” por GCP. Por esta razón openstack utiliza el mismo servicio Metadata publicado en la IP 169.254.169.254 de GCP y se genera conflicto. Para dar solución se configura OpenStack para enviar Metadata en una unidad de configuración especial que se adjunta a la instancia cuando se inicia: Config Drive. La documentación y requerimientos se encuentran en el enlace: <https://docs.openstack.org/nova/queens/user/config-drive.html>.

*#Instalar el programa genisoimage*

*#Editar el archivo nova.conf*

*sudo vi /etc/nova/nova.conf*

*# Add lines:*

*mkisofs\_cmd = /usr/bin/genisoimage*

*config\_drive\_format = iso9660*

*config\_drive\_cdrom = true*

*force\_config\_drive = true*

*#Editar /etc/neutron/dhcp\_agent.ini habilitando las líneas:*

*enable\_isolated\_metadata = True*

*enable\_metadata\_network = True*

*#reiniciar servicios*

*sudo systemctl restart devstack@\**

A partir de este momento es posible enviar información metadata y gestionar las VM desde OSM.

1. **Script Sprint 2 PoC Assurance**

La variable OSM\_VAR\_PCTX simula una métrica de red con forma sinusoidal que presenta un comportamiento anormal de forma aleatoria. Es usada para las alarmas tipo NS\_FAIL y MASSIVE\_FAIL y se obtiene por medio de VCA (Juju) utilizando los siguientes script que corren de forma permanente en la instancia:

*#scriptseno.sh*

*#!/bin/bash*

*typeset -i Ciclos=1*

*typeset -i T=300*

*typeset -i Eje=60*

*typeset -i A=10*

*typeset -i MaxA=100*

*typeset -i PorcError=20*

*typeset -i MaxCiclos=0 Ciclo=0 Error=0*

*MaxCiclos=$(( 60 \* ${Ciclos} ))*

*CiclosError=$(( ${MaxCiclos}\*${PorcError}/100 ))*

*typeset -i CicloError=0 bError=0 TotCicloError=0*

*CantError=$(( ${MaxA} - ${Eje} - ${A} ))*

*while true; do*

*n=1*

*Ciclo=0*

*TotCicloError=0*

*while [[ ${Ciclo} -lt ${MaxCiclos} ]]; do*

*Ciclo=$(( ${Ciclo}+1 ))*

*#Seg=$( date "+%S" )*

*ExisteError=$(( $RANDOM % 100 + 1 ))*

*if [[ ${bError} -eq 1 ]]; then*

*CicloError=$(( ${CicloError}+1 ))*

*if [[ ${CicloError} -ge ${DuracionError} ]]; then*

*bError=0*

*TotCicloError=$(( ${TotCicloError}+${CicloError} ))*

*CicloError=0*

*ValorError=0*

*fi*

*fi*

*if [[ bError -eq 0 ]] && [[ ${ExisteError} -lt ${PorcError} ]] && [[ ${TotCicloError} -lt ${CiclosError} ]]; then*

*bError=1*

*DuracionError=$(( $RANDOM % ${CiclosError} + 1 ))*

*ValorError=$(( $RANDOM % ${CantError} + 1 ))*

*fi*

*echo "" | awk -v Eje=${Eje} -v n=${n} -v PorcError=${PorcError} -v Ciclo=${Ciclo} -v A=${A}\*

*-v bError=${bError} -v CantError=${CantError} -v ValorError=${ValorError}\*

*'{ MiRand=rand();*

*Seno=A\*sin(6\*n);*

*if ((bError) && (Seno < 0))*

*ValorError=-1\*ValorError;*

*metrica=Eje+ValorError+Seno;*

*printf "%.2f\n",metrica > "/home/ubuntu/metrica.txt"*

*}'*

*n=$(($n+3))*

*sleep ${T}*

*done*

*done*

*#simulate\_event.sh*

*#!/bin/bash*

*tag=$( tail -n 1 metrica.txt)*

*val="1"*

*number=$(awk '{print $1\*$2}' <<<"${tag} ${val}")*

*echo $number*

Para el caso VirtualInfraColapse se utiliza una métrica obtenida de la VIM ( almacenada en gnocchi):

| **Name** | **Type** | **Unit** | **Resource** | **Origin** | **Support** | **Note** |
| --- | --- | --- | --- | --- | --- | --- |
| memory.usage | Gauge | MB | instance ID | Pollster | Libvirt, Hyper-V, vSphere, XenAPI | Volume of RAM used by the instance from the amount of its allocated memory |

Para generar la alarma se simula un alto consumo de memoria en la VM cada 30 minutos con el script a continuación que corre de forma permanente:

*vi scriptmemoria.sh*

*#!/bin/bash*

*sudo umount /media/ramdisk/*

*sudo rm -rf /media/ramdisk*

*sudo mkdir -p /media/ramdisk*

*sudo mount -t tmpfs -o size=1234M tmpfs /media/ramdisk/*

*while true; do*

*cat /dev/zero > /media/ramdisk/large*

*sleep 600*

*rm /media/ramdisk/large*

*sleep 1800*

*done*

*#correr el script en Background*

*nohup ./scriptmemoria.sh &*