

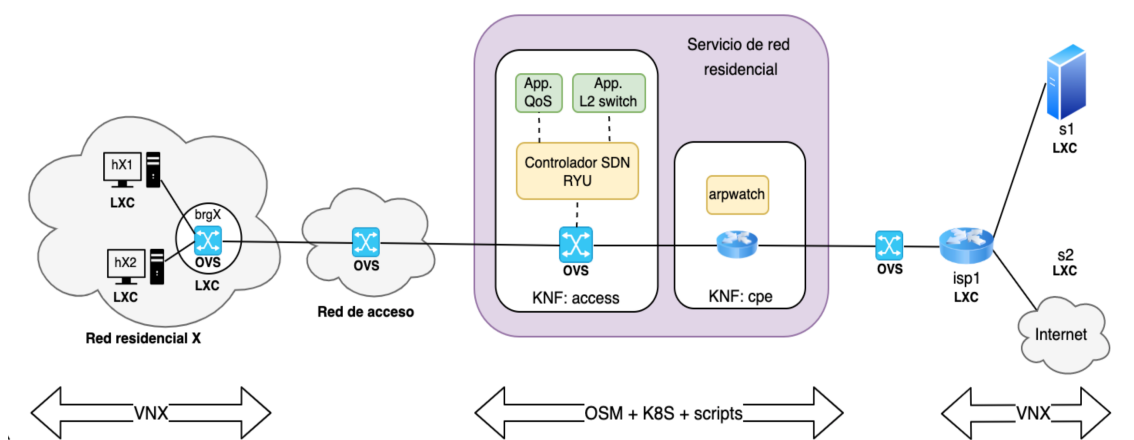
Práctica Final – RDSV

- Luis Alberto López Álvarez y Álvaro de Rojas Maraver

Para la realización de la práctica final de la asignatura, partimos de lo elaborado en la práctica 4, modificando las redes virtualizadas mediante OSM, que ahora contarán con los siguientes requisitos:

- Soporte de QoS implementado mediante SDN con Ryu.
- Añadir servicios adicionales, como el registro de las MACs de la red residencial.

A continuación, se muestra el escenario que se ha implementado en este trabajo final:



Captura 1. Escenario trabajo final

Los **requisitos mínimos** desarrollados establecidos en el enunciado, como modificaciones a la práctica 4 son:

- Sustituir el switch de “KNF:access” por un conmutador controlado por OpenFlow.
- Conectividad IPv4 desde la red residencial hacia Internet. Uso de doble NAT en “KNF:cpe” y en isp1 (igual que en la p4).
- Activar la captura de tráfico ARP mediante “arpwatch” en “KNF:cpe” (escribe logs cuando se entera de otra dirección MAC).
- Gestión de la calidad de servicio en la red de acceso mediante la API REST de Ryu controlando “KNF:access”.
 - Para limitar el ancho de banda de bajada hacia la red residencial.
- Despliegue para dos redes residenciales.
- Todo automatizado mediante OSM y scripts, utilizando como base los comandos de la práctica 4, añadiendo el on-boarding de NS/VNFs y la instanciación de NS mediante el mismo método.

Para realizar el **despliegue** de nuestro proyecto, se pueden seguir los pasos detallados en el **fichero README.txt adjunto** en el proyecto, mientras que en este documento se mostrarán los cambios realizados para lograr las funcionalidades requeridas.

Para ello, las **tareas básicas** que hemos empezado a realizar que permiten cumplir con estos requisitos se enumeran a continuación, una vez se han entendido los detalles internos del escenario de la práctica 4:

- Sustituir los repositorios por nuestros propios repositorios:
 - El contenedor Docker de las KNFs en DockerHub
 - cuenta educaredes → cuenta luichu
 - El repositorio Helm en GitHub Pages
 - <https://educaredes.github.io/nfv-lab> → <https://Luislopal.github.io/repo-rdsv>
- Automatización del despliegue de la práctica anterior mediante OSM y scripts para montar el escenario de las redes.
- Despliegue de lo anterior para dos redes residenciales.
- Comprobación de que existe conectividad IPv4 desde la red residencial hacia Internet mediante un doble NAT: en KNF:cpe y en isp1.
- Añadir la funcionalidad “arpwatch” para capturar tráfico ARP.
 - Activación de la herramienta arpwatch en KNF:cpe
- Gestión de QoS en la frontera de la red de acceso, mediante la API REST de Ryu en KNF:access (y en brgX) para limitar el ancho de banda de bajada hacia la red residencial.
 - Para la red residencial: 12 Mbps de bajada (y 6 Mbps de subida)
 - Para hX1: 8 Mbps mínimo de bajada (y 4 Mbps mínimo de subida)
 - Para hX2: 4 Mbps máximo de bajada (y 2 Mbps máximo de subida)
 - Independiente de la dirección IP asignada por DHCP a hX1 y hX2

Para poder empezar la práctica es necesario montar una carpeta compartida en VirtualBox, con la ruta local en el ordenador anfitrión y punto de montaje “/home/upm/shared” en las máquinas virtuales (K8S y OSM), como se detalla en el enunciado de la práctica (en nuestro caso, se ha realizado en un único MAC, que se utilizará como máquina anfitriona). Se va a trabajar sobre dos máquinas virtuales: “RDSV-K8S” (192.168.56.11), que emula las distintas redes y máquinas del escenario y “RDSV-OSM” (192.168.56.12), que instala el entorno OSM que coordina las funciones de la red virtual.

Dentro de ambas máquinas es necesario realizar unos scripts que realizan la configuración inicial y otorgan los permisos necesarios al escenario, para poder realizar el despliegue posterior correctamente. Además, dentro de la máquina “RDSV-K8S”, se realizan los despliegues de los escenarios “home” y “server” utilizando los siguientes comandos dentro del directorio donde se ubica la descarga del proyecto.

```
> sudo vnx -f vnx/nfv3_home_lxc_ubuntu64.xml -t
> sudo vnx -f vnx/nfv3_server_lxc_ubuntu64.xml -t
```

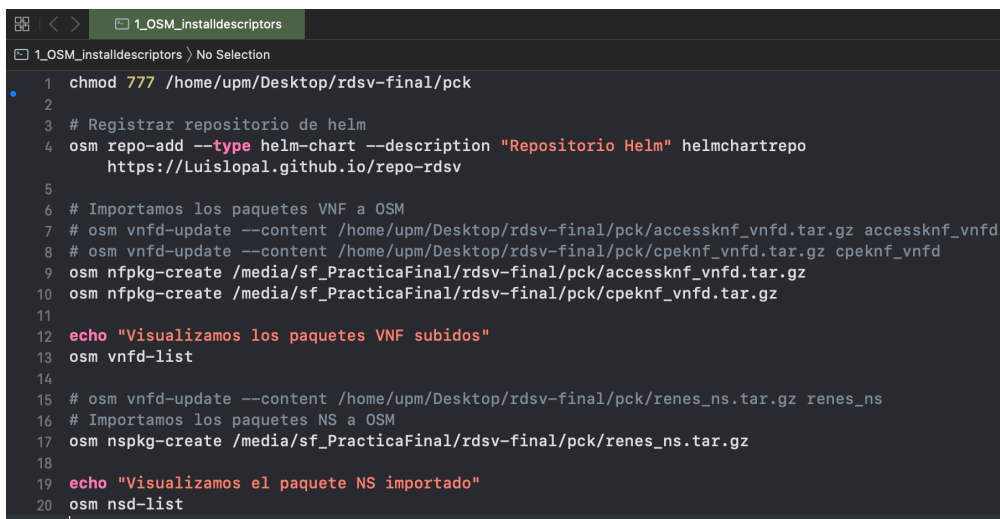
Se ha realizado de igual forma un segundo script, que permite apagar el escenario sin tener que introducir ambos comandos de forma recurrente. Podemos ejecutarlo utilizando el siguiente comando:

```
> ./0_OSM_configuracion_off.sh
```

Ya desde la máquina virtual “RDSV-OSM”, es necesario registrar el repositorio Helm que se va a utilizar posteriormente, para poder subir tanto los paquetes VNF como los NS a OSM, lo que se conoce como on-boarding, unido a la instanciación de cada una de las dos instancias que se van a desplegar siguiendo el proceso realizado en la práctica anterior.

Todo esto se realiza mediante la ejecución de dos scripts, que se adjuntan a continuación y que habrá que ejecutar desde la máquina OSM.

- **1_OSM_installdescriptors.sh:** Instalación de descriptors en “RDSV-OSM”, y servicio de red onboarding. La explicación de cada comando se puede observar en los comentarios en línea del script.
 - Se automatiza el proceso de subida de los OSM Packages -> VNF packages (accessknf_vnfd.tar.gz y cpeknf_vnfd.tar.gz).
 - Se automatiza el proceso de subida de los OSM Packages -> NS packages (renes_ns.tar.gz).
- **2_OSM_instanciacion.sh:** Creación y visualización de las instancias del servicio renes1 y renes2. La explicación de cada comando se puede observar en los comentarios en línea del script.

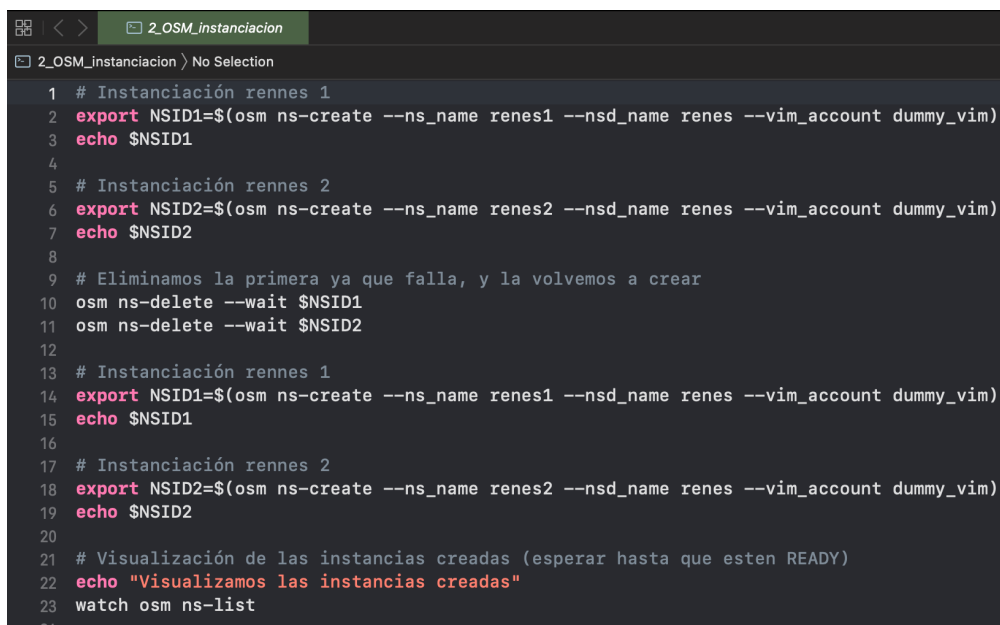


```
1 chmod 777 /home/upm/Desktop/rdsv-final/pck
2
3 # Registrar repositorio de helm
4 osm repo-add --type helm-chart --description "Repositorio Helm" helmchartrepo
   https://Luislopal.github.io/repo-rdsv
5
6 # Importamos los paquetes VNF a OSM
7 # osm vnfd-update --content /home/upm/Desktop/rdsv-final/pck/accessknf_vnfd.tar.gz accessknf_vnfd
8 # osm vnfd-update --content /home/upm/Desktop/rdsv-final/pck/cpeknf_vnfd.tar.gz cpeknf_vnfd
9 osm nfpkg-create /media/sf_PracticaFinal/rdsv-final/pck/accessknf_vnfd.tar.gz
10 osm nfpkg-create /media/sf_PracticaFinal/rdsv-final/pck/cpeknf_vnfd.tar.gz
11
12 echo "Visualizamos los paquetes VNF subidos"
13 osm vnfd-list
14
15 # osm vnfd-update --content /home/upm/Desktop/rdsv-final/pck/renes_ns.tar.gz renes_ns
16 # Importamos los paquetes NS a OSM
17 osm nspkg-create /media/sf_PracticaFinal/rdsv-final/pck/renes_ns.tar.gz
18
19 echo "Visualizamos el paquete NS importado"
20 osm nsd-list
21
```

Captura 2. Script instalar descriptors

Para utilizar el script ejecutaremos el siguiente comando en un terminal de la máquina OSM.

```
> ./1_OSM_installdescriptors.sh
```



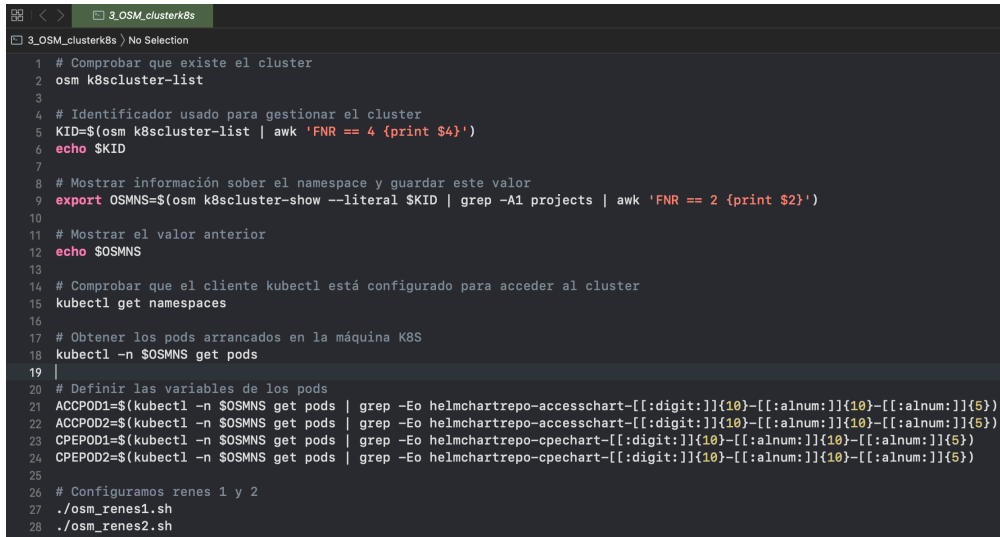
```
1 # Instanciación renes 1
2 export NSID1=$(osm ns-create --ns_name renes1 --nsd_name renes --vim_account dummy_vim)
3 echo $NSID1
4
5 # Instanciación renes 2
6 export NSID2=$(osm ns-create --ns_name renes2 --nsd_name renes --vim_account dummy_vim)
7 echo $NSID2
8
9 # Eliminamos la primera ya que falla, y la volvemos a crear
10 osm ns-delete --wait $NSID1
11 osm ns-delete --wait $NSID2
12
13 # Instanciación renes 1
14 export NSID1=$(osm ns-create --ns_name renes1 --nsd_name renes --vim_account dummy_vim)
15 echo $NSID1
16
17 # Instanciación renes 2
18 export NSID2=$(osm ns-create --ns_name renes2 --nsd_name renes --vim_account dummy_vim)
19 echo $NSID2
20
21 # Visualización de las instancias creadas (esperar hasta que esten READY)
22 echo "Visualizamos las instancias creadas"
23 watch osm ns-list
24
```

Captura 3. Script instanciación renes

Para utilizar el script ejecutaremos el siguiente comando en un terminal de la máquina OSM.

```
> ./2_OSM_instanciacion.sh
```

Entonces, una vez definidas cada una de las variables de los pods, tal como se muestra en la captura siguiente, es necesario lanzar los scripts de rennes que configuran las instancias de red, y se adjuntan de igual forma.



```
1 # Comprobar que existe el cluster
2 osm k8scluster-list
3
4 # Identificador usado para gestionar el cluster
5 KID=$(osm k8scluster-list | awk 'FNR == 4 {print $4}')
6 echo $KID
7
8 # Mostrar información sobre el namespace y guardar este valor
9 export OSMNS=$(osm k8scluster-show --literal $KID | grep -A1 projects | awk 'FNR == 2 {print $2}')
10
11 # Mostrar el valor anterior
12 echo $OSMNS
13
14 # Comprobar que el cliente kubectl está configurado para acceder al cluster
15 kubectl get namespaces
16
17 # Obtener los pods arrancados en la máquina K8S
18 kubectl -n $OSMNS get pods
19 |
20 # Definir las variables de los pods
21 ACCPOD1=$(kubectl -n $OSMNS get pods | grep -Eo helmchartrepo-accesschart-[:digit:]{10}-[:alnum:]{10}-[:alnum:]{5})
22 ACCPOD2=$(kubectl -n $OSMNS get pods | grep -Eo helmchartrepo-accesschart-[:digit:]{10}-[:alnum:]{10}-[:alnum:]{5})
23 CPEPOD1=$(kubectl -n $OSMNS get pods | grep -Eo helmchartrepo-cpechart-[:digit:]{10}-[:alnum:]{10}-[:alnum:]{5})
24 CPEPOD2=$(kubectl -n $OSMNS get pods | grep -Eo helmchartrepo-cpechart-[:digit:]{10}-[:alnum:]{10}-[:alnum:]{5})
25
26 # Configuramos rennes 1 y 2
27 ./osm_renes1.sh
28 ./osm_renes2.sh
```

Captura 5. Configuración instancias de red

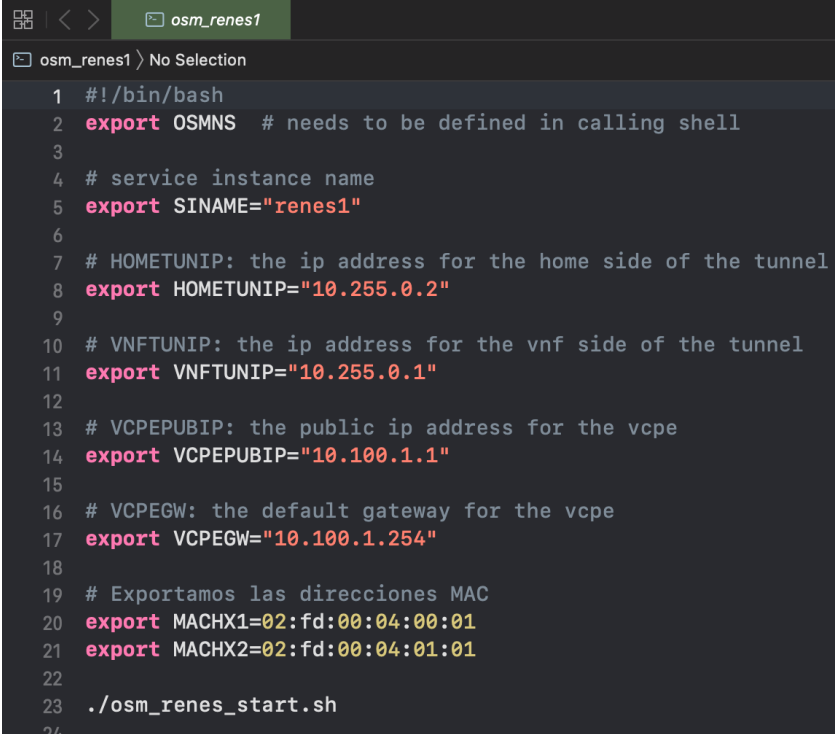
Para utilizar el script ejecutaremos el siguiente comando en un terminal de la máquina OSM, que llamará a cada una de las instancias de red (renes 1 y rennes 2) para proceder con su configuración en los pods arrancados.

```
> ./3_OSM_clusterk8s.sh
```

En la captura siguiente, se puede apreciar que la dirección IP que utiliza el NAT del CPE para dar salida al tráfico es la '10.100.1.1' corresponde con la 'VCPEPUBIP', que representa el enlace con el switch ExtNet1.

Se utilizan dos direcciones, una para cada interfaz dentro de cada una de las redes que hay en cada switch de acceso. En el caso del AccessNet1 la '10.255.0.0/24', siendo la '10.255.0.1' la interfaz que conecta con la instancia de 'rennes' y la '10.255.0.2' la que conecta con los equipos h11 y h12. Con ExtNet1 ocurre igual, pero ahora con la red '10.100.1.0/24', utilizando la IP '10.100.1.1' para la conexión con 'rennes' y la '10.100.1.254' para la conexión con Internet.

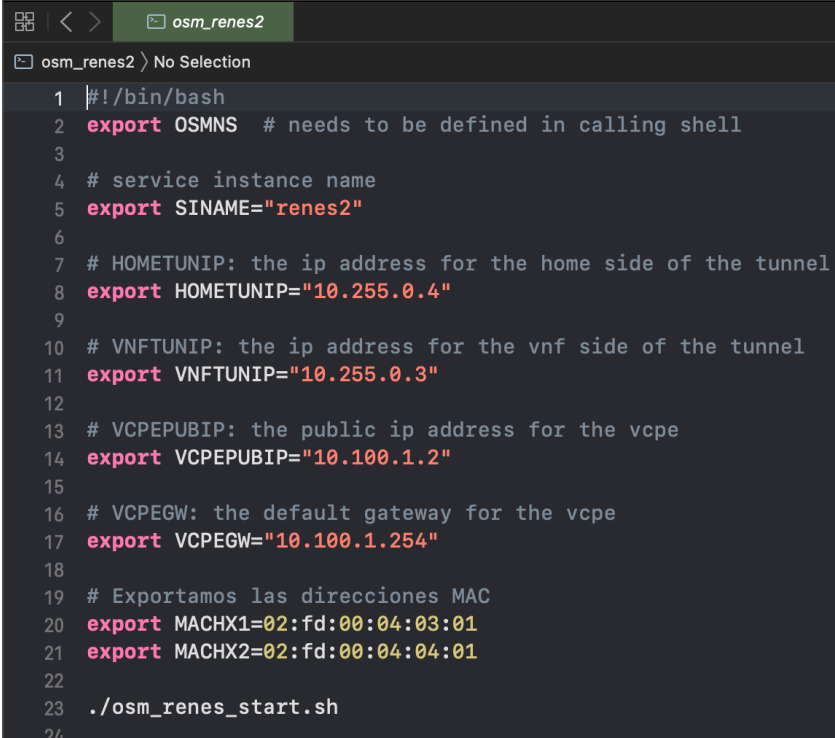
El primer script que hemos configurado ha sido `osm_renes1`, a continuación incluimos una captura de pantalla del fichero, en el que se pueden observar las IPs utilizadas para cada interfaz de red de las máquinas virtuales. Todas las líneas se encuentran comentadas con una breve explicación del código ejecutado en cada línea/comando.

A screenshot of a code editor showing the contents of the `osm_renes1` file. The editor has a dark theme and a tab labeled `osm_renes1`. The code is a shell script with 24 lines. Lines 2, 5, 8, 11, 14, 17, and 23 are executable commands, while the others are comments. The comments explain the purpose of each variable: `OSMNS` (needs to be defined), `SINAME` (service instance name), `HOMETUNIP` (home side of the tunnel), `VNFTUNIP` (vnf side of the tunnel), `VCPEPUBIP` (public ip address for the vcpe), `VCPEGW` (default gateway for the vcpe), and MAC addresses `MACHX1` and `MACHX2`. The script ends with `./osm_renes_start.sh`.

```
1 #!/bin/bash
2 export OSMNS # needs to be defined in calling shell
3
4 # service instance name
5 export SINAME="renes1"
6
7 # HOMETUNIP: the ip address for the home side of the tunnel
8 export HOMETUNIP="10.255.0.2"
9
10 # VNFTUNIP: the ip address for the vnf side of the tunnel
11 export VNFTUNIP="10.255.0.1"
12
13 # VCPEPUBIP: the public ip address for the vcpe
14 export VCPEPUBIP="10.100.1.1"
15
16 # VCPEGW: the default gateway for the vcpe
17 export VCPEGW="10.100.1.254"
18
19 # Exportamos las direcciones MAC
20 export MACHX1=02:fd:00:04:00:01
21 export MACHX2=02:fd:00:04:01:01
22
23 ./osm_renes_start.sh
24
```

Captura 6. Fichero de configuración `osm_renes1.sh`

En este segundo script “`osm_renes2`” hemos utilizado una configuración muy similar.

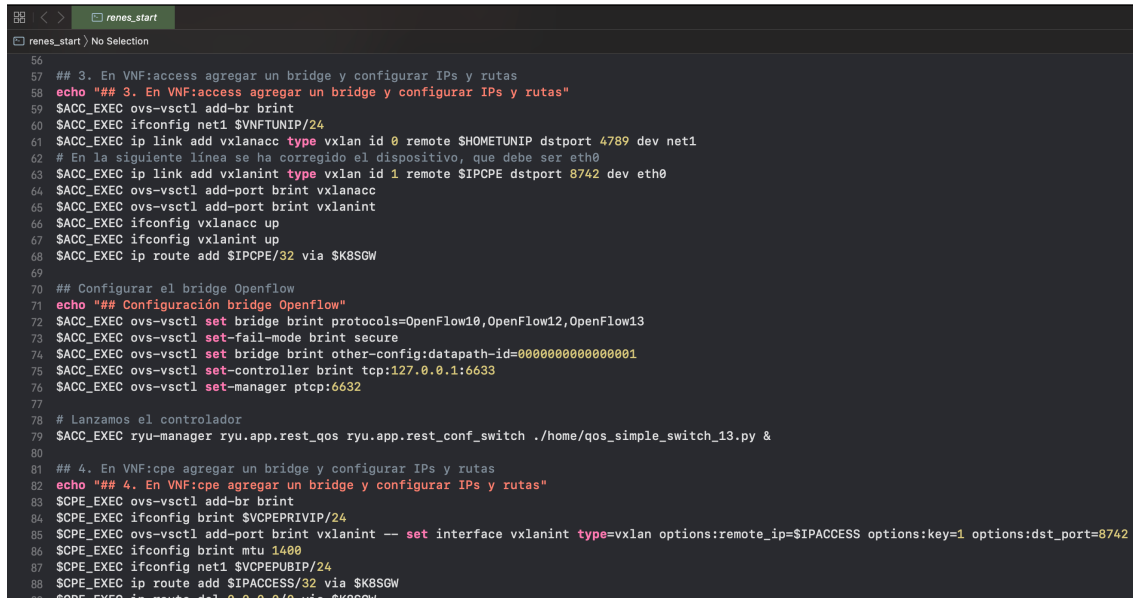
A screenshot of a code editor showing the contents of the `osm_renes2` file. The editor has a dark theme and a tab labeled `osm_renes2`. The code is a shell script with 24 lines, following the same structure as `osm_renes1`. The only differences are in the values of `SINAME` (now `renes2`), `HOMETUNIP` (now `10.255.0.4`), and `VNFTUNIP` (now `10.255.0.3`). The MAC addresses and the final command remain the same. The script ends with `./osm_renes_start.sh`.

```
1 #!/bin/bash
2 export OSMNS # needs to be defined in calling shell
3
4 # service instance name
5 export SINAME="renes2"
6
7 # HOMETUNIP: the ip address for the home side of the tunnel
8 export HOMETUNIP="10.255.0.4"
9
10 # VNFTUNIP: the ip address for the vnf side of the tunnel
11 export VNFTUNIP="10.255.0.3"
12
13 # VCPEPUBIP: the public ip address for the vcpe
14 export VCPEPUBIP="10.100.1.2"
15
16 # VCPEGW: the default gateway for the vcpe
17 export VCPEGW="10.100.1.254"
18
19 # Exportamos las direcciones MAC
20 export MACHX1=02:fd:00:04:03:01
21 export MACHX2=02:fd:00:04:04:01
22
23 ./osm_renes_start.sh
24
```

Captura 7. Fichero de configuración ‘`osm_renes2.sh`’

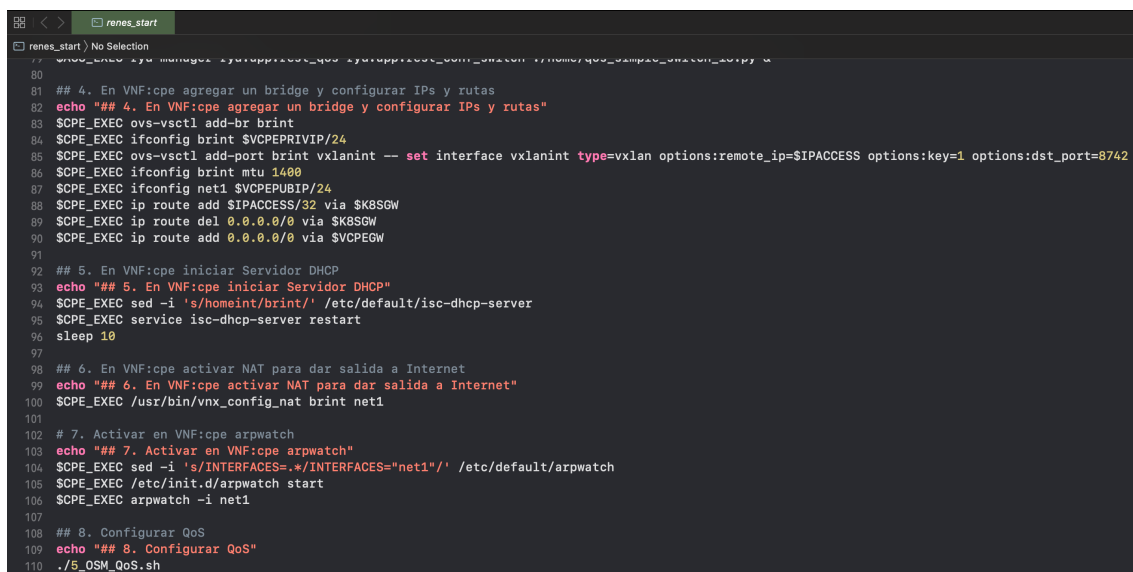
En este caso, **h21 y h22 han obtenido las direcciones IP '10.250.0.23' y la '10.250.0.22'** respectivamente, valores incluidos en el rango de direcciones posibles a adjudicar por DHCP (y justo los dos siguientes a los dos primeros adjudicados previamente a h11 y h12).

Cada uno de los dos ficheros anteriores, a su vez, llama al fichero `renes_start.sh`, que el script mediante el cual se configura cada instancia que se desee desplegar en la red.



```
56
57 ## 3. En VNF:access agregar un bridge y configurar IPs y rutas
58 echo "## 3. En VNF:access agregar un bridge y configurar IPs y rutas"
59 SACC_EXEC ovs-vsctl add-br brint
60 SACC_EXEC ifconfig net1 $VNFTUNIP/24
61 SACC_EXEC ip link add vxlanacc type vxlan id 0 remote $HOMETUNIP dstport 4789 dev net1
62 # En la siguiente línea se ha corregido el dispositivo, que debe ser eth0
63 SACC_EXEC ip link add vxlanint type vxlan id 1 remote $IPCPE dstport 8742 dev eth0
64 SACC_EXEC ovs-vsctl add-port brint vxlanacc
65 SACC_EXEC ovs-vsctl add-port brint vxlanint
66 SACC_EXEC ifconfig vxlanacc up
67 SACC_EXEC ifconfig vxlanint up
68 SACC_EXEC ip route add $IPCPE/32 via $K8SGW
69
70 ## Configurar el bridge Openflow
71 echo "## Configuración bridge Openflow"
72 SACC_EXEC ovs-vsctl set bridge brint protocols=OpenFlow10,OpenFlow12,OpenFlow13
73 SACC_EXEC ovs-vsctl set-fail-mode brint secure
74 SACC_EXEC ovs-vsctl set bridge brint other-config:datapath-id=0000000000000001
75 SACC_EXEC ovs-vsctl set-controller brint tcp:127.0.0.1:6633
76 SACC_EXEC ovs-vsctl set-manager ptcp:6632
77
78 # Lanzamos el controlador
79 SACC_EXEC ryu-manager ryu.app.rest_qos ryu.app.rest_conf_switch ./home/qos_simple_switch_13.py &
80
81 ## 4. En VNF:cpe agregar un bridge y configurar IPs y rutas
82 echo "## 4. En VNF:cpe agregar un bridge y configurar IPs y rutas"
83 SCPE_EXEC ovs-vsctl add-br brint
84 SCPE_EXEC ifconfig brint $VCPEPRIVIP/24
85 SCPE_EXEC ovs-vsctl add-port brint vxlanint -- set interface vxlanint type=vxlan options:remote_ip=$IPACCESS options:key=1 options:dst_port=8742
86 SCPE_EXEC ifconfig brint mtu 1400
87 SCPE_EXEC ifconfig net1 $VCPEPUBIP/24
88 SCPE_EXEC ip route add $IPACCESS/32 via $K8SGW
89 SCPE_EXEC ip route del 0.0.0.0/0 via $K8SGW
90 SCPE_EXEC ip route add 0.0.0.0/0 via $K8SGW
```

Captura 9. Fichero `renes_start.sh` (I)



```
77 SACC_EXEC ovs-vsctl set-manager ptcp:6632
78
79 ## 4. En VNF:cpe agregar un bridge y configurar IPs y rutas
80 echo "## 4. En VNF:cpe agregar un bridge y configurar IPs y rutas"
81 SCPE_EXEC ovs-vsctl add-br brint
82 SCPE_EXEC ifconfig brint $VCPEPRIVIP/24
83 SCPE_EXEC ovs-vsctl add-port brint vxlanint -- set interface vxlanint type=vxlan options:remote_ip=$IPACCESS options:key=1 options:dst_port=8742
84 SCPE_EXEC ifconfig brint mtu 1400
85 SCPE_EXEC ifconfig net1 $VCPEPUBIP/24
86 SCPE_EXEC ip route add $IPACCESS/32 via $K8SGW
87 SCPE_EXEC ip route del 0.0.0.0/0 via $K8SGW
88 SCPE_EXEC ip route add 0.0.0.0/0 via $VCPEGW
89
90 ## 5. En VNF:cpe iniciar Servidor DHCP
91 echo "## 5. En VNF:cpe iniciar Servidor DHCP"
92 SCPE_EXEC sed -i 's/homeint/brint/' /etc/default/isc-dhcp-server
93 SCPE_EXEC service isc-dhcp-server restart
94 sleep 10
95
96 ## 6. En VNF:cpe activar NAT para dar salida a Internet
97 echo "## 6. En VNF:cpe activar NAT para dar salida a Internet"
98 SCPE_EXEC /usr/bin/vnx_config_nat brint net1
99
100 # 7. Activar en VNF:cpe arpwatch
101 echo "## 7. Activar en VNF:cpe arpwatch"
102 SCPE_EXEC sed -i 's/INTERFACES=.*//INTERFACES="net1"/' /etc/default/arpwatch
103 SCPE_EXEC /etc/init.d/arpwatch start
104 SCPE_EXEC arpwatch -i net1
105
106 ## 8. Configurar QoS
107 echo "## 8. Configurar QoS"
108 ./5_QSM_QoS.sh
```

Captura 10. Fichero `renes_start.sh` (II)

En cuanto al uso de la API REST, se añade el bridge a Ryu y se configura la cola de bajada para que tenga una tasa máxima de 12 Mbps, como se puede apreciar en la captura siguiente. La cola correspondiente a hX1 tiene un mínimo de 8 Mbps de bajada y un máximo de 4 Mbps en el caso de hX2, por lo que, una vez está el escenario funcionando por completo, se han llevado a cabo varias pruebas utilizando el comando speedtest para comprobar que la configuración se ha llevado a cabo correctamente y se adjuntan las pruebas, de igual forma, a continuación.

```

1 #!/bin/bash
2
3 set -u # to verify variables are defined
4 : $ACC_EXEC
5 : $MACHX1
6 : $MACHX2
7
8 # Set ovsdb_addr in order to access OVSDb
9 echo "Set ovsdb_addr in order to access OVSDb"
10 $ACC_EXEC curl -X PUT -d '{"tcp:127.0.0.1:6632"}' http://127.0.0.1:8080/v1.0/conf/switches/0000000000000001/ovsdb_addr
11
12 # Set queues
13 echo "Set Queues"
14 $ACC_EXEC curl -X POST -d '{"port_name": "vxlanacc", "type": "linux-htb", "max_rate": "12000000", "queues": [{"max_rate": "4000000"}, {"min_rate": "8000000"}]}' http://127.0.0.1:8080/qos/queue/0000000000000001
15
16 #Set QoS rules
17 echo "Set QoS rules"
18 $ACC_EXEC curl -X POST -d '{"match": {"d1_dst": "${MACHX1}", "d1_type": "IPv4", "nw_proto": "TCP"}, "actions": {"queue": "1"}}' http://127.0.0.1:8080/qos/rules/0000000000000001
19 $ACC_EXEC curl -X POST -d '{"match": {"d1_dst": "${MACHX2}", "d1_type": "IPv4", "nw_proto": "TCP"}, "actions": {"queue": "0"}}' http://127.0.0.1:8080/qos/rules/0000000000000001
20
21 # Get Rules of QoS
22 $ACC_EXEC curl -X GET http://127.0.0.1:8080/qos/rules/0000000000000001

```

Captura 8. Script configuración QoS

El comando speedtest nos permitirá medir las métricas del rendimiento de la conexión a Internet como la descarga, la carga, la latencia y la pérdida de paquetes de manera nativa sin depender de un navegador web, para el uso que lo vamos a utilizar es para medir el tráfico de bajada de h11, h12, h21 y h22.

- Ejecución del comando speedtest en un terminal de la máquina virtual h12:

```

vnxe12:~$ speedtest
Retrieving speedtest.net configuration...
Testing from M247 Ltd (146.70.74.184)...
Retrieving speedtest.net server list...
Selecting best server based on ping...
Hosted by IBERSONTEL (Ciudad Real) [160.43 km]: 465.41 ms
Testing download speed.....
Download: 2.22 Mbit/s
Testing upload speed.....
Upload: 5.80 Mbit/s

```

- Ejecución del comando speedtest en un terminal de la máquina virtual h22:

```

vnxe22:~$ speedtest
Retrieving speedtest.net configuration...
Testing from M247 Ltd (146.70.74.184)...
Retrieving speedtest.net server list...
Selecting best server based on ping...
Hosted by IBERSONTEL (Ciudad Real) [160.43 km]: 524.721 ms
Testing download speed.....
Download: 3.04 Mbit/s
Testing upload speed.....
Upload: 2.90 Mbit/s

```

Se puede observar que están ambos valores por debajo de los 4 Mbps máximos de bajada determinados mediante la calidad de servicio. Se podría haber limitado también el tráfico de subida que es una mejora opcional, pero nos hemos centrado únicamente en configurar la QoS en el tráfico de bajada.

- Ejecución del comando speedtest en un terminal de la máquina virtual h21:

```

vnx@h21:~$ speedtest
Retrieving speedtest.net configuration...
Testing from M247 Ltd (146.70.74.184)...
Retrieving speedtest.net server list...
Selecting best server based on ping...
Hosted by IBERSONTEL (Ciudad Real) [160.43 km]: 410.167 ms
Testing download speed.....
.....
Download: 8.06 Mbit/s
Testing upload speed.....
.....
Upload: 10.02 Mbit/s
vnx@h21:~$

```

- Ejecución del comando speedtest en un terminal de la máquina virtual h11:

```

vnx@h11:~$ speedtest
Retrieving speedtest.net configuration...
Testing from M247 Ltd (146.70.74.184)...
Retrieving speedtest.net server list...
Selecting best server based on ping...
Hosted by IBERSONTEL (Ciudad Real) [160.43 km]: 412.175 ms
Testing download speed.....
.....
Download: 9.30 Mbit/s
Testing upload speed.....
.....
Upload: 3.32 Mbit/s
vnx@h11:~$

```

En el segundo caso, se puede ver como la velocidad mínima de bajada para hX1 corresponde con los valores delimitados por la QoS de 8 Mbps como mínimo.

Repositorios utilizados

Para llevar a cabo la práctica se han creado dos repositorios de GitHub.

- repo-rdsv: <https://github.com/Luislopal/repo-rdsv.git>
 - En este repositorio público se ha subido la carpeta Helm y todo su contenido.
- rdsv-final: <https://github.com/Luislopal/rdsv-final.git>
 - Este repositorio es privado y pertenece a los dos integrantes de la práctica, contiene todo el proyecto listo para ser ejecutado, con sus instrucciones de despliegue. Este repositorio se subirá al buzón de entrega de la práctica de la asignatura para su calificación. Las instrucciones de despliegue están en el README.md

También hemos creado un repositorio Docker con nuestro contenedor.

- Luichu/vnf-img: <https://hub.docker.com/r/luichu/vnf-img>
 - El Dockerfile incluye los paquetes de ubuntu "ryu-bin" y "arpwatch" y se ha añadido el fichero qos_simple_switch_13.py con la modificación que se propone en la práctica de QoS.

[README.md](#)

Instrucciones:

- Usar las dos máquinas virtuales RDSV-OSM y RDSV-K8S en ordenador propio.
 - Disponible en <http://idefix.dit.upm.es/download/vnx/vnx-vm/RDSV2022-v1.ova>
- Crear una carpeta compartida en el host y en las dos máquinas virtuales RDSV-OSM, llamar a la carpeta "PracticaFinal". Este paso se recomienda realizarlo desde VirtualBox, seleccionar "configuración" -> "Carpetas compartidas", realizar este paso con las máquinas RDSV-OSM y RDSV-K8S.
- Clonar el repositorio en la carpeta compartida:

```
cd /Desktop/PracticaFinal
```

```
git clone https://github.com/Luislopal/rdsv-final.git
```

Pasos para el despliegue:

Todos los comandos se ejecutarán desde el directorio clonado de la carpeta compartida.

1. Máquina OSM: Ejecutar script 0

```
./0_OSM_configuracion.sh
```

2. Máquina K8S: Ejecutar script 0

```
./0_K8S_configuracion_on.sh
```

3. Máquina OSM: Ejecutar script 1

```
./1_OSM_installdescriptors.sh
```

4. Máquina OSM: Ejecutar script 2

```
./2_OSM_instanciacion.sh
```

5. Máquina OSM: Ejecutar script 3

```
./3_OSM_clusterk8s.sh
```

6. Máquina K8S: Ejecutar script 4

```
./4_K8S_iPerf.sh
```