



Instituto Politecnico Nacional



ESCOM “ESCUELA SUPERIOR DE CÓMPUTO”

ADMINISTRACIÓN DE SERVICIOS EN RED

ENRUTAMIENTO MÚLTIPLE

PROFE: RICARDO MARTÍNEZ ROSALES

ALUMNO: Rojas Alvarado Luis Enrique

GRUPO: 4CM1

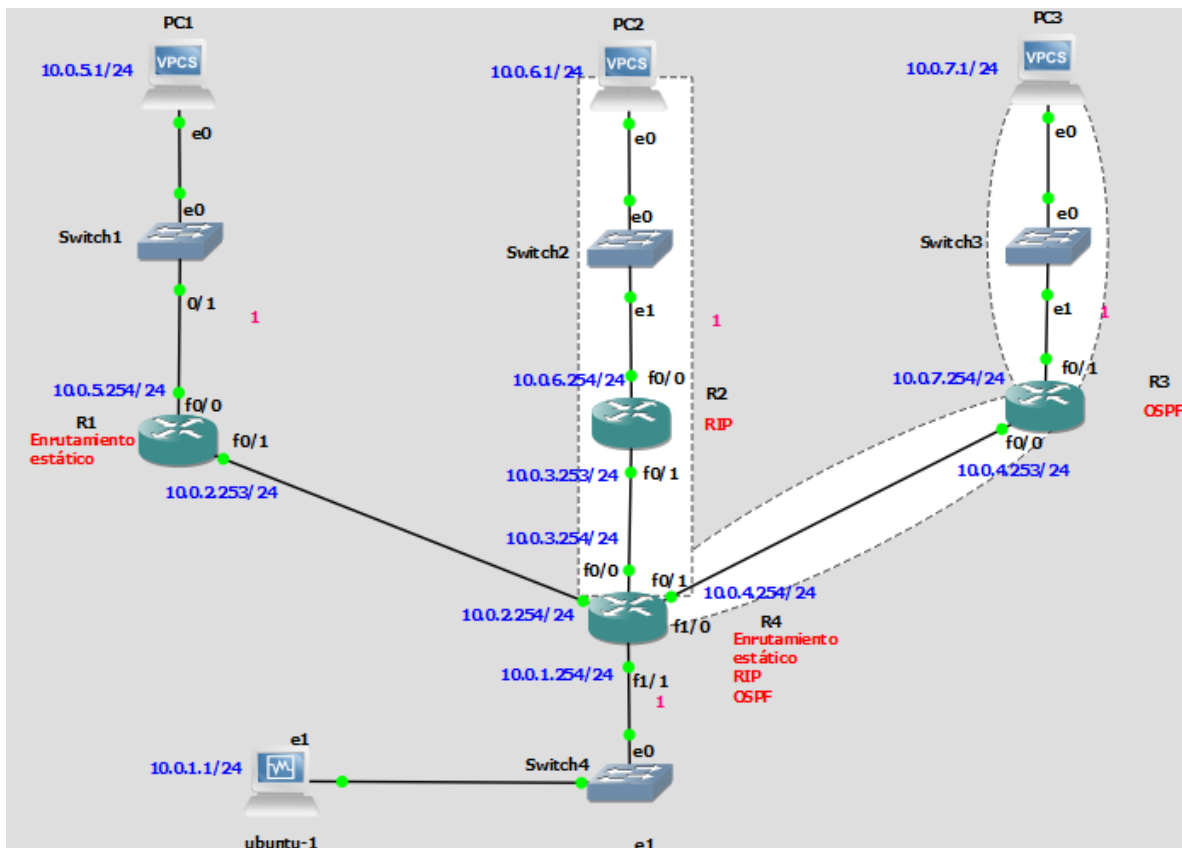
Objetivo

Desarrollar un programa en Python que sea capaz de levantar el enrutamiento estático y dinámico de la columna vertebral de una topología con múltiples métodos de enrutamiento.

Implementación

Implementar en GNS3 la siguiente topología, donde únicamente se van a configurar las interfaces indicadas en enrutadores, MV y VPCS y un usuario admin con contraseña admin en ssh de los enrutadores.

Desarrollar un programa en Python que correrá en la máquina virtual host1 y que sea capaz de levantar los distintos métodos de enrutamiento indicados en la topología y permitir que exista conectividad en toda la red.



Se configuran los routers R1 con el enrutamiento estático, el R2 con el enrutamiento dinámico RIP y el R3 con el protocolo de enrutamiento OSPF. Cabe destacar que en el R2 solo se configuran las redes 10.0.6.0 y 10.0.3.0 con el protocolo RIP y para el R3 solo se debe configurar las redes 10.0.7.0 y la 10.0.4.0. El R1 se tendrá que configurar toda la red con 0.0.0.0 para evitar problemas al enrutar el R4. Cabe destacar que se preconfiguró las interfaces antes de levantar ambos protocolos.

dinámicos. Así como también las interfaces del R4 posteriormente a enrutar con Python y flex.

Puesto que hemos configurado todos los routers con ssh, no deberíamos de poder acceder a los routers R1, R2, R3 debido a que no está enrutado el R4 y la máquina virtual no puede alcanzar esas redes.

Haciendo ping a la puerta de enlace de la máquina virtual.

```
luis@luis-VirtualBox:~$ ping 10.0.1.254
PING 10.0.1.254 (10.0.1.254) 56(84) bytes of data.
64 bytes from 10.0.1.254: icmp_seq=1 ttl=255 time=13.0 ms
64 bytes from 10.0.1.254: icmp_seq=2 ttl=255 time=5.95 ms
64 bytes from 10.0.1.254: icmp_seq=3 ttl=255 time=5.54 ms
64 bytes from 10.0.1.254: icmp_seq=4 ttl=255 time=7.93 ms
^C
--- 10.0.1.254 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 5.544/8.108/13.014/2.972 ms
luis@luis-VirtualBox:~$
```

Haciendo ping a la red de la máquina PC3 de GNS3. Como se podrá observar, la red es inalcanzable.

```
luis@luis-VirtualBox:~$ ping 10.0.7.0
PING 10.0.7.0 (10.0.7.0) 56(84) bytes of data.
From 10.0.1.254 icmp_seq=1 Destination Host Unreachable
From 10.0.1.254 icmp_seq=2 Destination Host Unreachable
From 10.0.1.254 icmp_seq=3 Destination Host Unreachable
From 10.0.1.254 icmp_seq=4 Destination Host Unreachable
^C
--- 10.0.7.0 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3003ms
luis@luis-VirtualBox:~$
```

Accediendo por la puerta de enlace 10.0.1.254 (R4) a través de SSH.

```
luis@luis-VirtualBox:~$ ssh admin@10.0.1.254
Password:

R4#
```

Accediendo por la puerta de enlace 10.0.7.254 (R3) a través de SSH.

```
luis@luis-VirtualBox:~$ ssh admin@10.0.7.254
ssh: connect to host 10.0.7.254 port 22: No route to host
luis@luis-VirtualBox:~$
```

Para configurar el R4 tendremos que usar el framework de Python flask para visualizar la información en una página web.

Se creó un método de flask llamado “enrutar”. En el que con un comando GET de HTTP, y colocando la URL en el navegador <http://127.0.0.1:5000/dispositivos/4/enrutar> agregando el número de dispositivo (en este caso el 4), y llamando al método enrutar del programa conexiónSSH.py el cual hace la conexión por medio de SSH que se explica a continuación.

```
@app.route('/dispositivos/<int:id>/enrutar', methods=['GET'])
def dame_dispositivo_enrutar(id):
    dispositivo = Dispositivo.query.get_or_404(id)
    hostname = dispositivo.hostname
    ip = dispositivo.ip
    # si el id es el router 4, entonces lo enrutamos
    if id == 4:
        # mandamos a llamar a los métodos del programa conexionSSH.py
        resultado = enrutar(hostname, ip, username, password)
        return jsonify({"enrutado": str(resultado)})
    else:
        # si queremos enrutar cualquier otro dispositivo solo muestra la tabla de enrutamiento
        resultado = show_ip_route(hostname, ip, username, password)
        return jsonify({"Ya enrutado": str(resultado)})
```

El método de flask muestra en la pantalla del navegador en forma de json, el resultado que devuelve el método enrutar del programa conexionSSH.py

En el programa conexionSSH.py contiene la apertura de un archivo de texto en el cual nosotros colocaremos los comandos para enrutar el R4. Y se guarda en la variable “commands” que es de tipo lista.

```
# abre el archivo de texto que contiene los comandos a ejecutar.
with open('R4_comandos.txt', 'r') as f:
    commands = [line for line in f.readlines()]
```

El método enrutar del programa conexionSSH.py recibe el nombre del dispositivo a enrutar, la dirección IP de acceso al router, el usuario y contraseña para ingresar a través de la conexión SSH. Con la ayuda de un bloque try - except se declara una variable en la que se guardará un archivo de texto para comprobar el enrutamiento al final de la ejecución de la función. Y se hace la respectiva conexión SSH con ayuda de la librería paramiko para la administración de redes, después de hacer la conexión se abre el archivo de texto que se mencionó para escribir en él los comandos que se mandarán y que adquirimos en otro archivo de texto, al final se regresa la salida que recibió la conexión de SSH y el nombre del dispositivo que enrutamos.

```
def enrutar (device, ip, username, password):
    try:
        outputFileName = device + '_salida_enrutamiento.txt'
        connection = paramiko.SSHClient()
        connection.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        connection.connect(ip, username=username, password=password, look_for_keys=False, allow_agent=False)
        new_connection = connection.invoke_shell()
        output = clear_buffer(new_connection)
        time.sleep(2)
        # new_connection.send("terminal length 0\n")
        # output = clear_buffer(new_connection)
        with open(outputFileName, 'wb') as f:
            for command in commands:
                new_connection.send(command)
                time.sleep(2)
                output = new_connection.recv(max_buffer)
                f.write(output)
        result = output.decode('utf-8').splitlines()
        new_connection.close()
        return device, result
    except:
        print('Error interno: ', sys.exc_info())
        return 'Error en enrutar'
```

Para comprobar, usamos un método de flask en el cual accedemos con el método GET de HTTP y usando la URL en un navegador http://127.0.0.1:5000/dispositivos/4/tabla_enrutamiento en el cual llamaremos a una función del programa conexionSSH.py que muestra la tabla de enrutamiento con el comando “show ip route” por medio de la conexión SSH de pexpect. El método regresa en forma de JSON el texto que se recuperó al ingresar el comando remotamente, y se muestra en la ventana del navegador.

```
@app.route('/dispositivos/<int:id>/tabla_enrutamiento', methods=['GET'])
def dame_dispositivo_tabla(id):
    dispositivo = Dispositivo.query.get_or_404(id)
    hostname = dispositivo.hostname
    ip = dispositivo.ip
    resultado = show_ip_route(hostname, ip, username, password)
    return jsonify({"Tabla de enrutamiento: ": str(resultado)})
```

```
def show_ip_route(device, ip, username, password):
    child = pxssh.pxssh()
    # iniciamos sesión
    child.login(ip, username, password, auto_prompt_reset=False)
    # enviamos cada comando de la lista
    child.sendline('show ip route list 1')
    # esperamos la etiqueta del router
    child.expect(device+'#')
    # la propiedad before guarda el texto imprimible hasta el texto esperado
    result = child.before
    # cierra la sesión
    child.logout()
    # regresa el dispositivo y el resultado
    return device, result
```

Cabe destacar que los dispositivos y sus direcciones IP se ingresaron por medio de una base de datos, usando SQLAlchemy. Y usando el método POST de HTTP y agregando la URL 127.0.0.1:5000/dispositivos/ y agregando las variables a agregar

entre comillas: http POST http://127.0.0.1:5000/dispositivos/ 'hostname'='R1'
'loopback'='192.168.1.0' 'ip'='10.0.5.254'

```
class Dispositivo(bd.Model):
    __tablename__ = 'Dispositivos'
    id = bd.Column(bd.Integer, primary_key=True)
    hostname = bd.Column(bd.String(64), unique=True)
    loopback = bd.Column(bd.String(120), unique=True)
    ip = bd.Column(bd.String(120), unique=True)

    def dame_url(dis):
        return url_for('dame_dispositivo', id=dis.id, _external=True)

    def exporta_datos(dis):
        return {
            'disp_url': dis.dame_url(),
            'hostname': dis.hostname,
            'loopback': dis.loopback,
            'ip': dis.ip,
        }

    def importa_datos(dis, datos):
        try:
            dis.hostname = datos['hostname']
            dis.loopback = datos['loopback']
            dis.ip = datos['ip']
        except KeyError as e:
            raise ValidationError('Invalid dispositivo: missing ' + e.args[0])
        return dis
```

```
@app.route('/dispositivos/', methods=['POST'])
def nuevo_dispositivo():
    dispositivo = Dispositivo()
    dispositivo.importa_datos(request.json)
    bd.session.add(dispositivo)
    bd.session.commit()
    return jsonify({}), 201, {'Locacion': dispositivo.dame_url()}
```

Con el comando GET se obtiene la información de todos los dispositivos ingresados a la base de datos. En formato JSON se visualiza en una página de un navegador.

```
@app.route('/dispositivos/', methods=['GET'])
def dame_dispositivos():
    return jsonify({'dispositivos': [dispositivo.dame_url()
                                     for dispositivo in Dispositivo.query.all()]})
```

Con el método PUT podemos modificar cualquiera de los datos ingresados en la base de datos.

```
@app.route('/dispositivos/<int:id>', methods=['PUT'])
def edita_dispositivo(id):
    dispositivo = Dispositivo.query.get_or_404(id)
    dispositivo.importa_datos(request.json)
    bd.session.add(dispositivo)
    bd.session.commit()
    return jsonify({})
```

Finalmente, con el método GET y agregando el número de dispositivo a la URL en el navegador, se puede visualizar la información de dicho dispositivo ingresado.

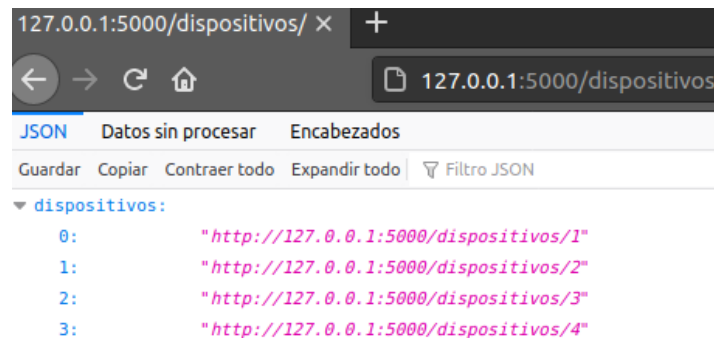
```
@app.route('/dispositivos/<int:id>', methods=['GET'])
def dame_dispositivo(id):
    return jsonify(Dispositivo.query.get_or_404(id).exporta_datos())
```

PRUEBAS:

Desde una terminal de Linux, activamos nuestro entorno virtual y ejecutamos el programa de flask (el programa principal) y nos muestra que está ejecutándose en el servidor localhost. Por lo que procedemos a abrir una ventana de navegador.

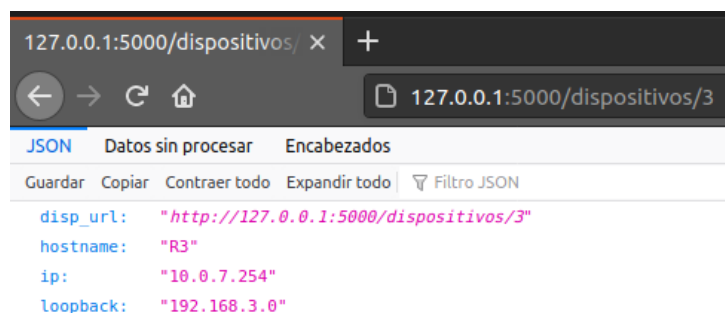
```
luis@luis-VirtualBox:~/Documentos/Enrutamiento_Multiple$ source venv/bin/activate
(venv) luis@luis-VirtualBox:~/Documentos/Enrutamiento_Multiple$ python3 EnrutamientoMultiple.py
* Running on http://0.0.0.0:5000/
* Restarting with reloader
```

Como nos podemos darnos cuenta, despliega la información contenida en nuestra base de datos llenada con anterioridad. Nos indica la página de flask que tenemos 4 dispositivos.



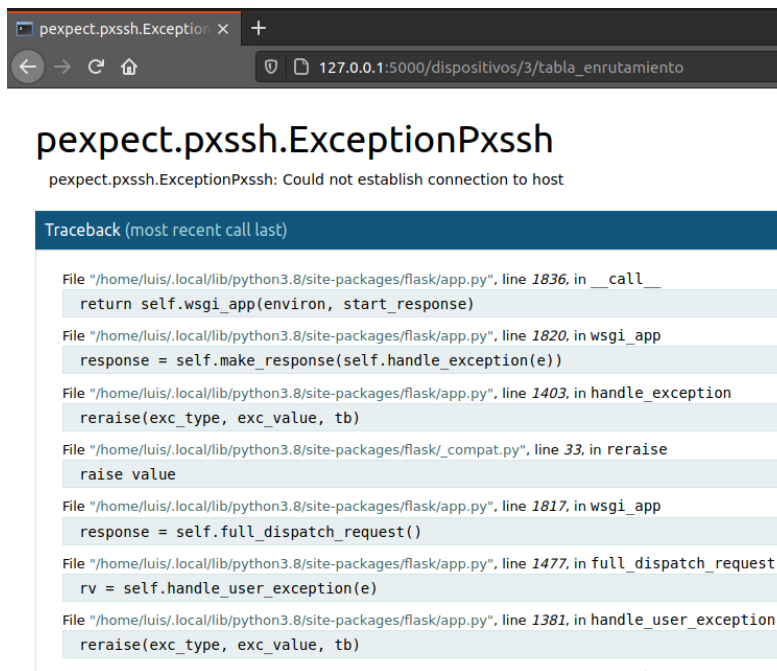
```
{
  "dispositivos": [
    {
      "url": "http://127.0.0.1:5000/dispositivos/1"
    },
    {
      "url": "http://127.0.0.1:5000/dispositivos/2"
    },
    {
      "url": "http://127.0.0.1:5000/dispositivos/3"
    },
    {
      "url": "http://127.0.0.1:5000/dispositivos/4"
    }
  ]
}
```

Al agregar el número de dispositivo, podemos visualizar la información que tenemos de él en la base de datos en formato JSON.



```
{
  "disp_url": "http://127.0.0.1:5000/dispositivos/3",
  "hostname": "R3",
  "ip": "10.0.7.254",
  "loopback": "192.168.3.0"
}
```

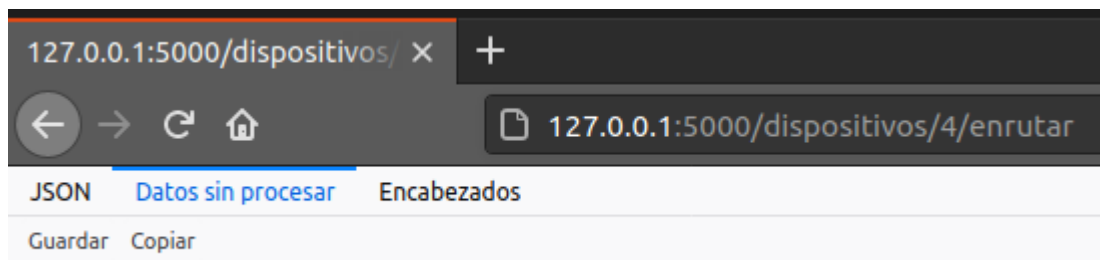

Si queremos ver su tabla de enrutamiento agregamos “tabla_enrutamiento” al final para ver lo que nos sale.



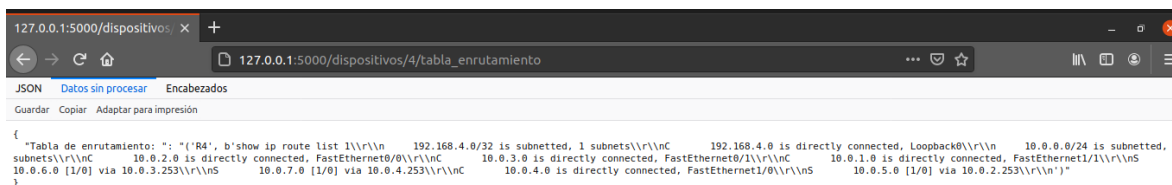
The screenshot shows a web browser with a single tab titled 'pexpect.pxssh.ExceptionPxssh'. The address bar displays '127.0.0.1:5000/dispositivos/3/tabla_enrutamiento'. The main content area shows a 500 error with the title 'pexpect.pxssh.ExceptionPxssh' and the message 'pexpect.pxssh.ExceptionPxssh: Could not establish connection to host'. Below this is a 'Traceback (most recent call last)' section with the following code:

```
File "/home/luis/.local/lib/python3.8/site-packages/flask/app.py", line 1836, in __call__
    return self.wsgi_app(environ, start_response)
File "/home/luis/.local/lib/python3.8/site-packages/flask/app.py", line 1820, in wsgi_app
    response = self.make_response(self.handle_exception(e))
File "/home/luis/.local/lib/python3.8/site-packages/flask/app.py", line 1403, in handle_exception
    reraise(exc_type, exc_value, tb)
File "/home/luis/.local/lib/python3.8/site-packages/flask/_compat.py", line 33, in reraise
    raise value
File "/home/luis/.local/lib/python3.8/site-packages/flask/app.py", line 1817, in wsgi_app
    response = self.full_dispatch_request()
File "/home/luis/.local/lib/python3.8/site-packages/flask/app.py", line 1477, in full_dispatch_request
    rv = self.handle_user_exception(e)
File "/home/luis/.local/lib/python3.8/site-packages/flask/app.py", line 1381, in handle_user_exception
    reraise(exc_type, exc_value, tb)
```

En este caso es una excepción de pexpect, puesto que aún no hemos enrutado el R4 que es el router de frontera y la máquina virtual no puede llegar a ese router. Por lo que procederemos a enrutar el R4.



Podemos observar ahora que no nos arroja ningún tipo de excepción o error. Por lo que al comprobar mostrando su tabla de enrutamiento podemos observar que todas las redes ahora son accesibles.




```

R4#sh ip route list 1
  192.168.4.0/32 is subnetted, 1 subnets
C    192.168.4.0 is directly connected, Loopback0
  10.0.0.0/24 is subnetted, 7 subnets
C    10.0.2.0 is directly connected, FastEthernet0/0
C    10.0.3.0 is directly connected, FastEthernet0/1
C    10.0.1.0 is directly connected, FastEthernet1/1
S    10.0.6.0 [1/0] via 10.0.3.253
S    10.0.7.0 [1/0] via 10.0.4.253
C    10.0.4.0 is directly connected, FastEthernet1/0
S    10.0.5.0 [1/0] via 10.0.2.253
R4#
*Nov 28 15:58:14.555: %SYS-5-CONFIG_I: Configured from console by admin on vty0 (10.0.1.1)
R4#

```

Por lo que si hacemos ping a la misma red de la PC3 de GNS3 ahora podemos llegar.

```

luis@luis-VirtualBox:~/Documentos/Enrutamiento_Multiple$ ping 10.0.7.0
PING 10.0.7.0 (10.0.7.0) 56(84) bytes of data.
64 bytes from 10.0.4.253: icmp_seq=1 ttl=254 time=34.1 ms
64 bytes from 10.0.4.253: icmp_seq=2 ttl=254 time=30.1 ms
64 bytes from 10.0.4.253: icmp_seq=3 ttl=254 time=23.8 ms
64 bytes from 10.0.4.253: icmp_seq=4 ttl=254 time=31.7 ms
^C
--- 10.0.7.0 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 23.778/29.900/34.099/3.815 ms
luis@luis-VirtualBox:~/Documentos/Enrutamiento_Multiple$

```

Y si accedemos por SSH podemos ingresar de igual manera.

```

luis@luis-VirtualBox:~/Documentos/Enrutamiento_Multiple$ ssh admin@10.0.7.254
Password:
R3#

```

Con esto hemos concluido la configuración remota de un router usando flask.