



Instituto Politécnico Nacional

Escuela Superior de Cómputo

Ingeniería en Sistemas Computacionales



Administración de Servicios en Red

Práctica 3

Servidor Web

Alumnos:

Edgar Flores Altamirano
Jimena Rosalía Campillo López

Profesor:

Josué Rangel Gonzáles

Grupo: 4CV2

22 de Octubre de 2019

1. Desarrollo

El objetivo de esta práctica es trabajar las preguntas de las tablas de los Routers CISCO. Estas preguntas nos ayudan a tener un mejor conocimiento sobre el funcionamiento de los routers y sobre como podemos configurarlos para obtener información específica que nos sea de utilidad dependiendo de la funcionalidad que queremos obtener. En esta parte de la práctica, trabajamos 3 apartados:

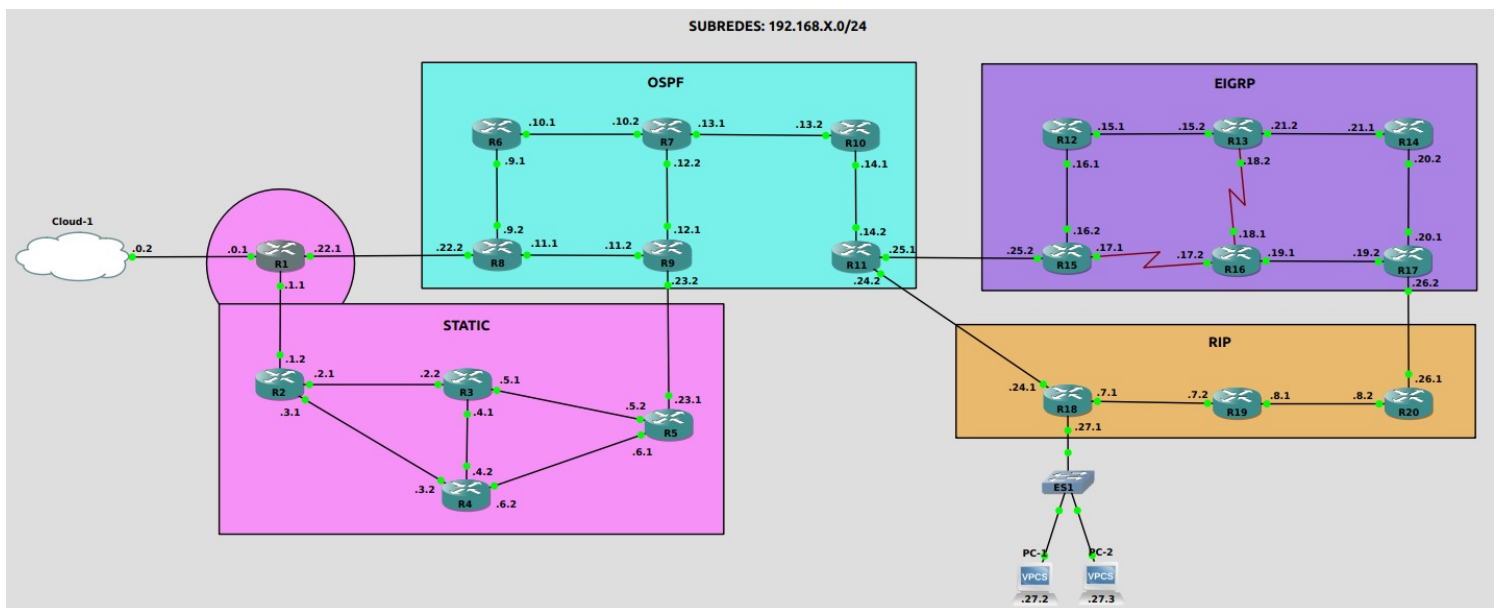
1. Ping Puller.

2. Traps.

3. Syslog.

2. Topología

A continuación, creamos la topología con los elementos que necesitamos para el desarrollo de nuestra práctica.



3. Desarrollo

A continuación mostramos la tabla Cisco, de la cual nos basaremos para resolver las preguntas presentadas.

Fault Management Self-Assessment

Table 3. Fault Management Self-Assessment

Question	Yes/No	Score
Does the organization have a ping poller which results in a fault being raised when a device fails to respond to a ping?		
Are SNMP traps monitored, and specific traps result in a fault being raised?		
Are syslog events monitored, and specific syslogs result in a fault being raised?		
Is device hardware status monitored and faults raised for environmental exceptions, including power supply failure, redundant system failover, device temperatures, etc?		
Are faults sent to the service desk and managed using an incident management process?		
Total		

1. Ping Puller

Explicando la pregunta de la tabla anterior respecto a este apartado, lo que se busca es que se mande un ping a toda la topología y que se encuentren todos los dispositivos que estén conectados, incluso si son nuevos, es decir, que se agreguen nuevos.

Para ello, lo primero que hicimos fue conectarnos vía Telnet para poder ingresar al Router, a continuación mostramos parte del Script en donde hacemos la conexión y sus validaciones, por si el usuario o contraseña, no son correctos.

```

def conectar_telnet(ip, passwordTelnet, passwordEnable = None):
    tn = None
    try:
        #Conectar Telnet
        print("Telnet: Conectandose con " + ip + " ...")
        tn = telnetlib.Telnet(ip)
        #tn.set_debuglevel(9)
        tn.read_until(b'Password: ', 5)
        tn.write(passwordTelnet.encode('utf-8') + b'\n')
        respuesta = tn.read_until(b'>', 1)
        #Comprobar que se establecio conexion
        if respuesta.find(b'Password: ') != -1:
            print('Telnet: Contraseña incorrecta')
            tn.close()
            return None, 'Telnet: Contraseña incorrecta.'
        print("Telnet: Conexion exitosa.")

        if passwordEnable != None :
            #Habilitar modo enable
            print("Telnet: Habilitando modo Enable.")
            tn.write(b'enable\n')
            tn.read_until(b'Password: ', 1)
            tn.write(passwordEnable.encode('utf-8') + b'\n')
            respuesta = tn.read_until(b'#', 1)
            if respuesta.find(b'Password: ') != -1:
                print('Telnet(enable): Contraseña incorrecta')
                tn.close()
                return None, 'Telnet(enable): Contraseña incorrecta.'
            print("Telnet(enable): Modo Enable habilitado.")

    except OSError as oserr:
        #Para No route to host 113
        print(oserr)
        return None, 'Interrupcion: ' + ip

    except EOFError as eoferror:
        #Conexion terminada
        print(eoferror)
        return None, 'Interrupcion.'

    return tn, None

```

Una vez que ya podemos mandarle comandos a nuestro Router, iniciamos con hacer un ping y leer todas las subredes que nos llegan.

```
def ping_thread(subnet):
    #Obtener informacion de la subred
    idSubred, strIdSubred, strMascaraSubred, strBroadcast, listHosts = info_subred(subnet)
    subred = Subred(idSubred, strIdSubred, strMascaraSubred, strBroadcast)
    for host in listHosts:
        #Ejecutar pings a la subred
        FNULL = open(os.devnull, 'w')
        command = ['ping', '-c', '1', '-W', '3', host]
        if subprocess.call(command, stdout = FNULL, stderr = subprocess.STDOUT) == 0:
            print(host + ' conectado')
            subred.addHost(host)
        else:
            print(host + ' no accesible')
    return subred

def ping_puller(subnets):
    threads = list()
    current_threads = 0
    total_treads = 0
    q = Queue()
    subredes = []

    for subnet in subnets:
        current_threads += 1
        total_treads += 1
        print("leyendo subred: " + subnet + ". current_thread: " + str(total_treads))
        #Ejecutar hilos para realizar los pings a las subredes
        t = threading.Thread(target = lambda q, arg1: q.put(ping_thread(arg1)), args = (q, subnet))
        threads.append(t)
        t.start()
        if current_threads == 10 or total_treads == len(subnets):
            current_threads = 0
            for t in threads:
                t.join()
            while not q.empty():
                subredes.append(q.get())
            threads.clear()
            q = Queue()

    return subredes
```

Ahora debemos separar la información obtenida, pues si la mostráramos como la recibimos, no se entendería claramente. A continuación eliminaremos las redes locales.

```
def obtener_subredes_telnet(tn):
    routeTable = b""
    #Expresion regular de una subred
    patron = re.compile('\\d{1,3}\\d{1,3}\\d{1,3}\\d{1,3}/\\d{1,2}')
    tn.write(b'show ip route list subnets | exclude L      \n')
    while True:
        routeTable += tn.read_until(b' --More-- ', 1)
        if routeTable.find(b' --More-- ') != -1:
            routeTable = routeTable.replace(b' --More-- ', b'')
            tn.write(b' ')
            continue
        break
    #considerar >
    routeTable = routeTable.decode('utf8')
    return set(patron.findall(routeTable))
```

Ahora, separaremos específicamente entre subred, máscara y broadcast.

```
def intIp_to_string(ip):
    strIp = str((ip >> 24) & 255)
    strIp += '.' + str((ip >> 16) & 255)
    strIp += '.' + str((ip >> 8) & 255)
    strIp += '.' + str(ip & 255)
    return strIp

def info_subred(cidr):
    aux = cidr.split('/')
    subnet = aux[0].split('.')

    idSubred = 0 | int(subnet[0])
    idSubred = (idSubred << 8) | int(subnet[1])
    idSubred = (idSubred << 8) | int(subnet[2])
    idSubred = (idSubred << 8) | int(subnet[3])
    mascaraSubred = (~0) << (32 - int(aux[1]))
    broadcast = (idSubred | (~mascaraSubred))

    strIdSubred = intIp_to_string(idSubred)
    strMascaraSubred = intIp_to_string(mascaraSubred)
    strBroadcast = intIp_to_string(broadcast)

    #Obtener todos los host de la subred
    listHosts = []
    host = idSubred + 1
    while host < broadcast:
        listHosts.append(intIp_to_string(host))
        host += 1
    n = 0
    return idSubred, strIdSubred, strMascaraSubred, strBroadcast, listHosts
```


Por último, creamos un método para imprimir la información obtenida y clasificada, para así poder mandarla en un archivo JSON a nuestro servidor web.

```
class Subred:
    idSubred = 0
    strIdSubred = ''
    strMascara = ''
    strBroadcast = ''
    hosts = []

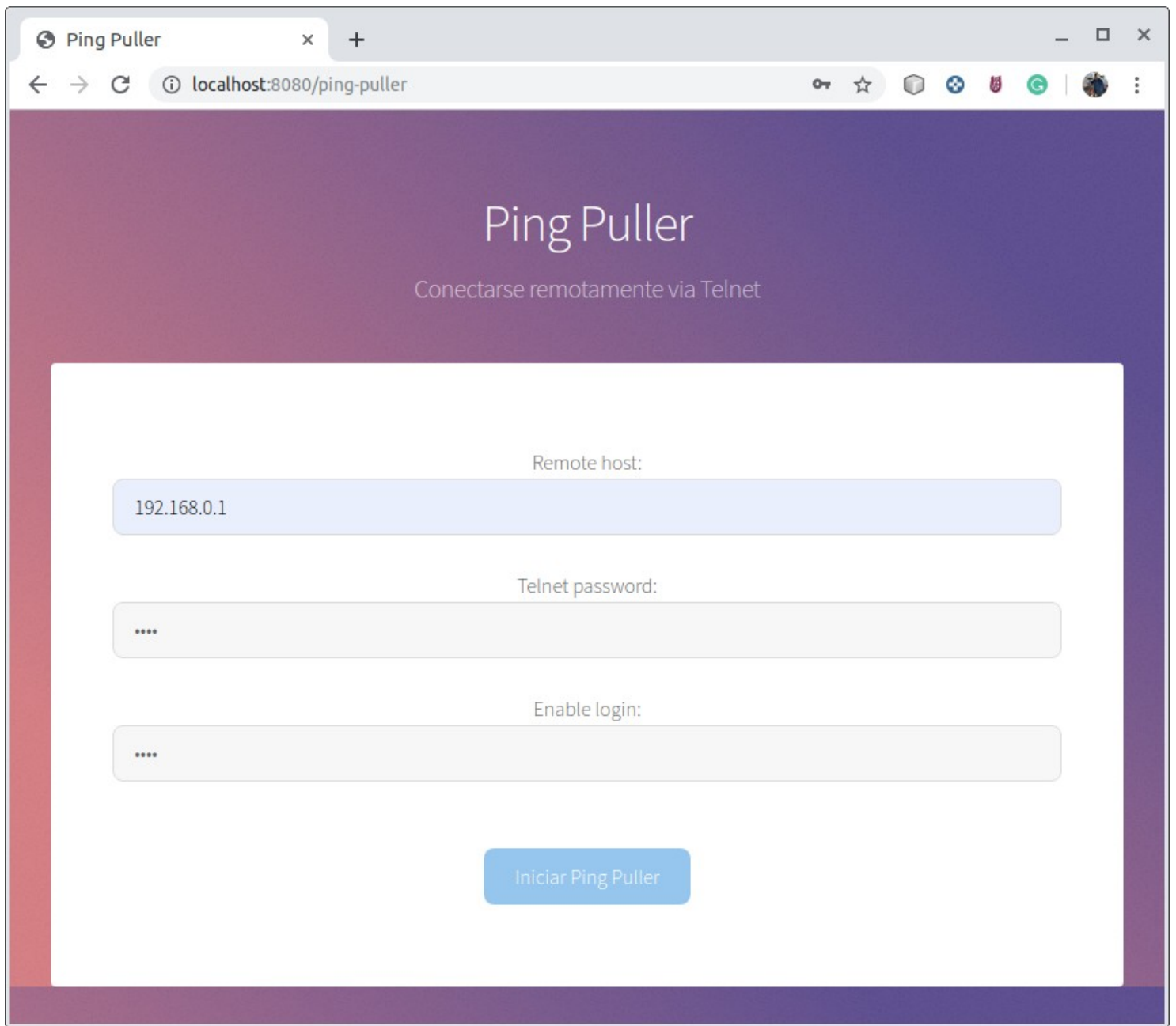
    def __init__(self, idSubred, strIdSubred, strMascara, strBroadcast):
        self.idSubred = idSubred
        self.strIdSubred = strIdSubred
        self.strMascara = strMascara
        self.strBroadcast = strBroadcast
        self.hosts = []

    def addHost(self, host):
        self.hosts.append(host)

    def formatoJson(self, data):
        hostData = []
        for host in self.hosts:
            hostData.append(host)

        data['subred'].append({
            'id': self.strIdSubred,
            'mascara': self.strMascara,
            'broadcast': self.strBroadcast,
            'host': hostData
        })
```

Por último, mostramos en la interfaz la conexión al router por vía Telnet, y los resultados que serían todas nuestras subredes ya clasificadas.



Subredes:

Subred: 192.168.0.0 (Mascara: 255.255.255.0, Broadcast: 192.168.0.255)

192.168.0.1

192.168.0.2

Subred: 192.168.1.0 (Mascara: 255.255.255.0, Broadcast: 192.168.1.255)

192.168.1.1

192.168.1.2

Subred: 192.168.2.0 (Mascara: 255.255.255.0, Broadcast: 192.168.2.255)

192.168.2.1

192.168.2.2

Subred: 192.168.3.0 (Mascara: 255.255.255.0, Broadcast: 192.168.3.255)

192.168.3.1

192.168.3.2

Subred: 192.168.4.0 (Mascara: 255.255.255.0, Broadcast: 192.168.4.255)

192.168.4.1

192.168.4.2

Subred: 192.168.5.0 (Mascara: 255.255.255.0, Broadcast: 192.168.5.255)

192.168.5.1

192.168.5.2

Subred: 192.168.6.0 (Mascara: 255.255.255.0, Broadcast: 192.168.6.255)

192.168.6.1

192.168.6.2

Subred: 192.168.7.0 (Mascara: 255.255.255.0, Broadcast: 192.168.7.255)

192.168.7.1

192.168.7.2

2. Traps

Para configurar el envío de traps, es necesario configurar el servidor SNMP en la versión 3, cabe mencionar que esto debe ser en cada uno de los routers que se encuentren en la topología a trabajar.

En este caso, nosotros asignamos como Host la dirección IP **192.168.0.2** y como usuario **RXUSER**, utilizando como contraseña **password** para la encriptación SHA y AES.

Si el puerto no se especifica, se utilizará el 162 que viene por default.

```
> conf t
> ip access-list standard PERMIT-ADMIN
> permit 192.168.0.2 0.0.0.255
> exit
> snmp-server view SNMP-RO iso included
> snmp-server group ADMIN v3 priv read SNMP-RO access
    PERMIT-ADMIN
> snmp-server user R1USER ADMIN v3 auth sha password
    pri aes 128 password
> snmp-server host 192.168.0.2 version 3 priv R1USER
> snmp-server enable traps
```

Para poder recibir las traps en nuestro anfitrión, se hizo uso de un demonio llamado “**snmptrapd**” que es una aplicación de Net-SNMP; para su instalación y configuración inicial se utilizaron los siguientes comandos:

```
> sudo apt-get install snmptrapd
> sudo systemctl daemon-reload
> sudo systemctl enable snmptrapd
> systemctl status snmptrapd.service
```

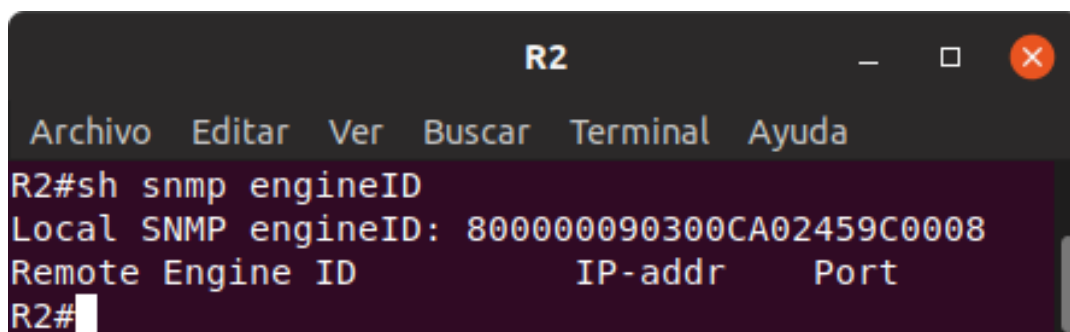
Para configurar la recepción de los traps, se configuró el archivo de configuración ***etc/snmp/snmptrapd.conf*** con las siguientes instrucciones:

```
createUser -e 0x8000000090300CA01458D0008 USER1 SHA "password" AES "password"  
authUser log,USER1  
traphandle default /usr/sbin/snmpthandler
```

Ahora, para registrar un usuario se agregó lo siguiente:

```
> createUser -e 0x8000000090300CA01458D0008 RXUSER  
SHA "password" AES "password"  
authUser log,execute,net RXUSER
```

Donde **-e 0x8000000090300CA01458D0008** corresponde al Engine ID del router que estemos registrando, para obtener dicho ID, debemos ingresar el comando **show snmp engineID**.



```
R2  
Archivo  Editar  Ver  Buscar  Terminal  Ayuda  
R2#sh snmp engineID  
Local SNMP engineID: 8000000090300CA02459C0008  
Remote Engine ID      IP-addr      Port  
R2#
```

Para procesar las traps recibidas, se utilizaron los siguientes comandos:

```
> traphandle iso.3.6.1.6.3.1.1.5.3
/usr/sbin/trap-senders/trap-sender-links.py
#OID Interface down
> traphandle iso.3.6.1.4.1.9.9.41.2.0.1 /usr/sbin/trap-
senders/trap-sender-mssg.py
#OID Interface changed
> state to up
> traphandle iso.3.6.1.6.3.1.1.5.4
/usr/sbin/trap-senders/trap-sender-links.py
#OID Interface up
> traphandle iso.3.6.1.4.1.9.9.43.2.0.1 /usr/sbin/trap-
senders/trap-sender-config.py
#OID Configure
> terminal or write
> traphandle iso.3.6.1.6.3.1.1.5.1
/usr/sbin/trap-senders/trap-sender-mssg.py
#OID Reboot
> traphandle iso.3.6.1.4.1.9.0.0 /usr/sbin/trap-senders/trap-
sender-mssg.py
#OID Reload
```

Donde traphandle es el comando que nos ayuda a manipular las traps, filtrándolas con el OID que se quiera recibir para así después ingresar la instrucción que se quiera realizar con esa notificación de trap.

A continuación se muestra el código del script en Python, el cual procesa las traps recibidas para después enviarlas al servidor web.

```
File Edit Selection Find View Goto Tools Project Preferences Help
trap-sender.py x
1  #!/usr/bin/env python3
2  import fileinput
3  import socket
4  import sys
5
6  data = ""
7  for line in fileinput.input():
8      data += line
9
10 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
11 server_address = ('localhost', 8081)
12
13 try:
14     # Send data
15     sent = sock.sendto(data.encode('utf-8'), server_address)
16
17 finally:
18     print('closing socket')
19     sock.close()
```

Line 19, Column 17 Tab Size: 4 Python

Como podemos observar, creamos un socket en el puerto 8081, que es a donde enviaremos las traps obtenidas de los routers.

Para recibir las traps, se utilizó el siguiente código.

```
31 def trap_receiver():
32     #Delete traps file
33     try:
34         remove(filenameTraps)
35     except:
36         pass
37
38     # Create a UDP socket
39     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
40     # Bind the socket to the port
41     server_address = ('', 8081)
42     print('starting trap service up on {} port {}'.format(*server_address))
43     sock.bind(server_address)
44     while True:
45         mssg, address = sock.recvfrom(4096)
46         print('Trap-service:received {} bytes from {}'.format(len(mssg), address))
47         print(mssg.decode('utf-8'))
48         fooTraps(mssg.decode('utf-8'))
```

3. Syslog

Para la configuración de las notificaciones Syslog, se necesitó configurar los routers con los siguientes comandos:

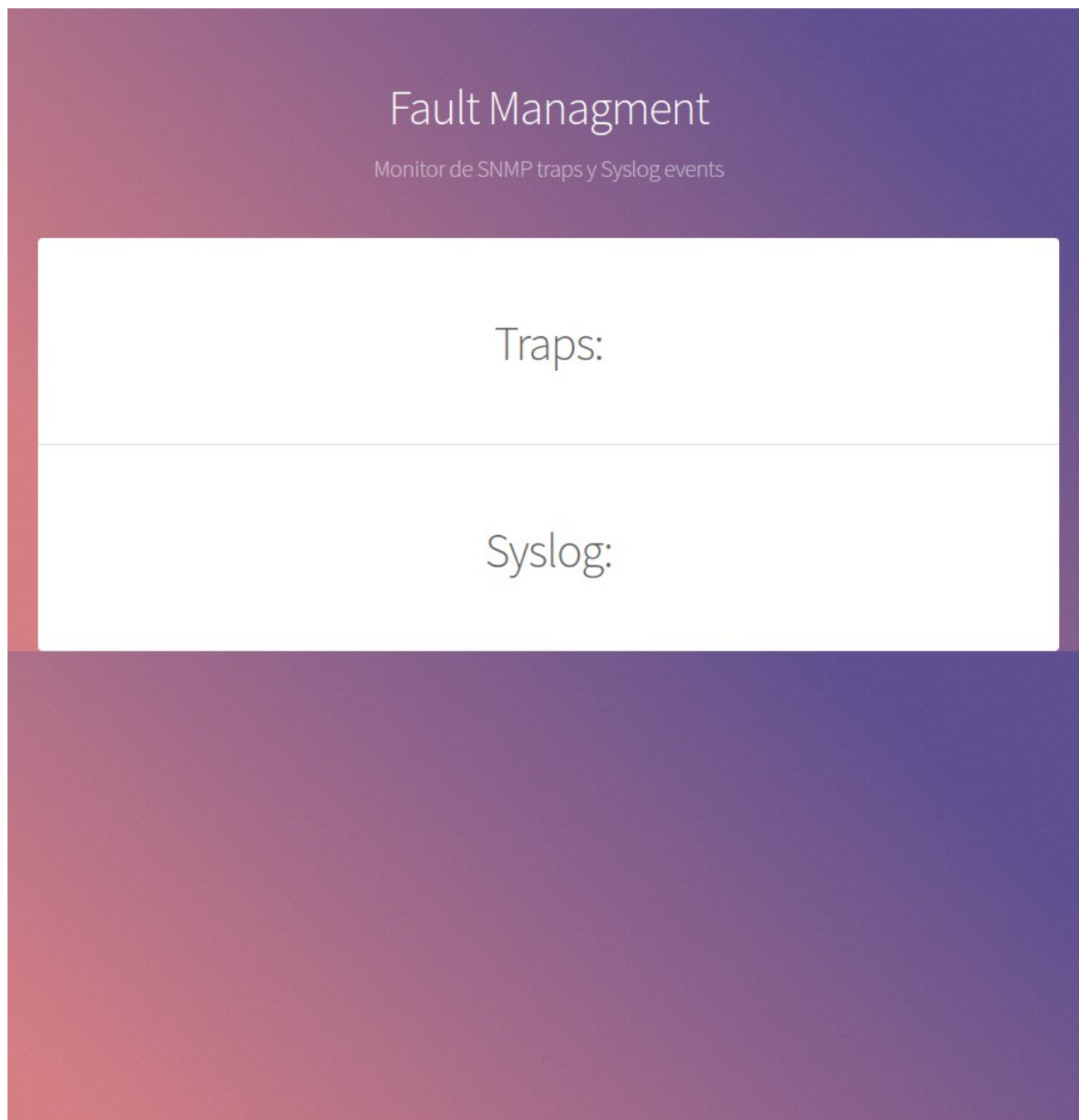
```
> logging host 192.168.0.2 transport udp port 8082
    > logging trap notifications
    > service timestamps log datetime
    > logging on
```

Donde podemos observar que se utilizó el puerto 8082 para el envío de notificaciones Syslog y además se configuró el envío de traps con un nivel de seguridad 5, el cual es Notifications, es decir, que nos llegarán todas las notificaciones nivel 1 a 5.

Para recibir las notificaciones en el servidor, se utilizó el siguiente código:

```
71 def syslog_receiver():
72     #Delete logging file
73     try:
74         remove(filenameSyslog)
75     except:
76         pass
77
78     # Create a UDP socket
79     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
80     # Bind the socket to the port
81     server_address = ('', 8082)
82     print('starting trap service up on {} port {}'.format(*server_address))
83     sock.bind(server_address)
84     while True:
85         mssg, address = sock.recvfrom(4096)
86         print('Trap-service:received {} bytes from {}'.format(len(mssg), address))
87         print(mssg.decode('utf-8'))
88         fooSyslog(mssg.decode('utf-8'), address)
```


Por último, mostramos las imágenes de nuestro servidor Web ya funcionando, mostrando las notificaciones Trap y Syslog generadas de cada router de nuestra topología.



Fault Managment

Monitor de SNMP traps y Syslog events

Traps:

```
> 2019-10-18 13:41:01.762356
  #Host[192.168.3.2]: Terminal configurada
> 2019-10-18 13:40:58.052215
  #Host[192.168.3.2]: unknown reload cause - suspect boot_data[BOOT_COUNT] 0x0, BOOT_COUNT 0, BOOTDATA 19
> 2019-10-18 13:40:58.380748
  #Host[192.168.3.2]: unknown reload cause - suspect boot_data[BOOT_COUNT] 0x0, BOOT_COUNT 0, BOOTDATA 19
> 2019-10-18 13:37:39.277525
  #Host[192.168.24.1]: FastEthernet1/0, up
> 2019-10-18 13:37:38.015714
  #Host[192.168.24.1]: LINK, UPDOWN, Interface FastEthernet1/0, changed state to up
> 2019-10-18 13:37:30.463449
  #Host[192.168.24.1]: FastEthernet1/0, administratively down
> 2019-10-18 13:37:19.635065
  #Host[192.168.24.1]: Terminal configurada
> 2019-10-18 13:37:04.534302
  #Host[192.168.8.2]: LINK, UPDOWN, Interface FastEthernet3/0, changed state to up
> 2019-10-18 13:37:05.677684
  #Host[192.168.8.2]: FastEthernet3/0, up
> 2019-10-18 13:36:54.349733
  #Host[192.168.8.2]: Terminal configurada
> 2019-10-18 13:36:46.522892
  #Host[192.168.8.2]: FastEthernet3/0, administratively down
> 2019-10-18 13:35:58.197530
  #Host[192.168.26.1]: FastEthernet3/0, administratively down
> 2019-10-18 13:35:58.744764
  #Host[192.168.26.1]: LINK, UPDOWN, Interface FastEthernet3/0, changed state to up
> 2019-10-18 13:35:58.990498
  #Host[192.168.26.1]: FastEthernet3/0, up
> 2019-10-18 13:35:20.039640
  #Host[192.168.26.1]: Terminal configurada
> 2019-10-18 13:35:04.506261
  #Host[192.168.26.1]: Terminal configurada
```

Syslog:

```
> Host[192.168.3.2,63499]:  
  #<189>39: *Oct 18 18:41:18: %SYS-5-CONFIG_I: Configured from console by console  
> Host[192.168.3.2,63499]:  
  #<189>38: *Oct 18 18:39:56: %SNMP-5-COLDSTART: SNMP agent on host R4 is undergoing a cold start  
> Host[192.168.24.1,60497]:  
  #<187>45: *Oct 18 18:31:51: %LINK-3-UPDOWN: Interface FastEthernet1/0, changed state to up  
> Host[192.168.24.1,60497]:  
  #<189>46: *Oct 18 18:31:52: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet1/0, changed state to up  
> Host[192.168.24.1,60497]:  
  #<189>43: *Oct 18 18:31:42: %LINK-5-CHANGED: Interface FastEthernet1/0, changed state to administratively down  
> Host[192.168.24.1,60497]:  
  #<189>44: *Oct 18 18:31:43: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet1/0, changed state to down  
> Host[192.168.8.2,60636]:  
  #<187>51: *Oct 18 18:37:03: %LINK-3-UPDOWN: Interface FastEthernet3/0, changed state to up  
> Host[192.168.8.2,60636]:  
  #<189>52: *Oct 18 18:37:04: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet3/0, changed state to up  
> Host[192.168.8.2,60636]:  
  #<189>50: *Oct 18 18:36:37: %SYS-5-CONFIG_I: Configured from console by console  
> Host[192.168.26.1,60636]:  
  #<187>46: *Oct 18 18:35:52: %LINK-3-UPDOWN: Interface FastEthernet3/0, changed state to up  
> Host[192.168.26.1,60636]:  
  #<189>47: *Oct 18 18:35:53: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet3/0, changed state to up  
> Host[192.168.26.1,60636]:  
  #<189>43: *Oct 18 18:35:07: %SYS-5-CONFIG_I: Configured from console by console
```