



# INSTITUTO POLITÉCNICO NACIONAL ESCUELA SUPERIOR DE CÓMPUTO

## Práctica 3

*Integrantes:*

López Ayala Eric Alejandro

López Romero Joel

*Profesora:*

Pérez de Los Santos Mondragón

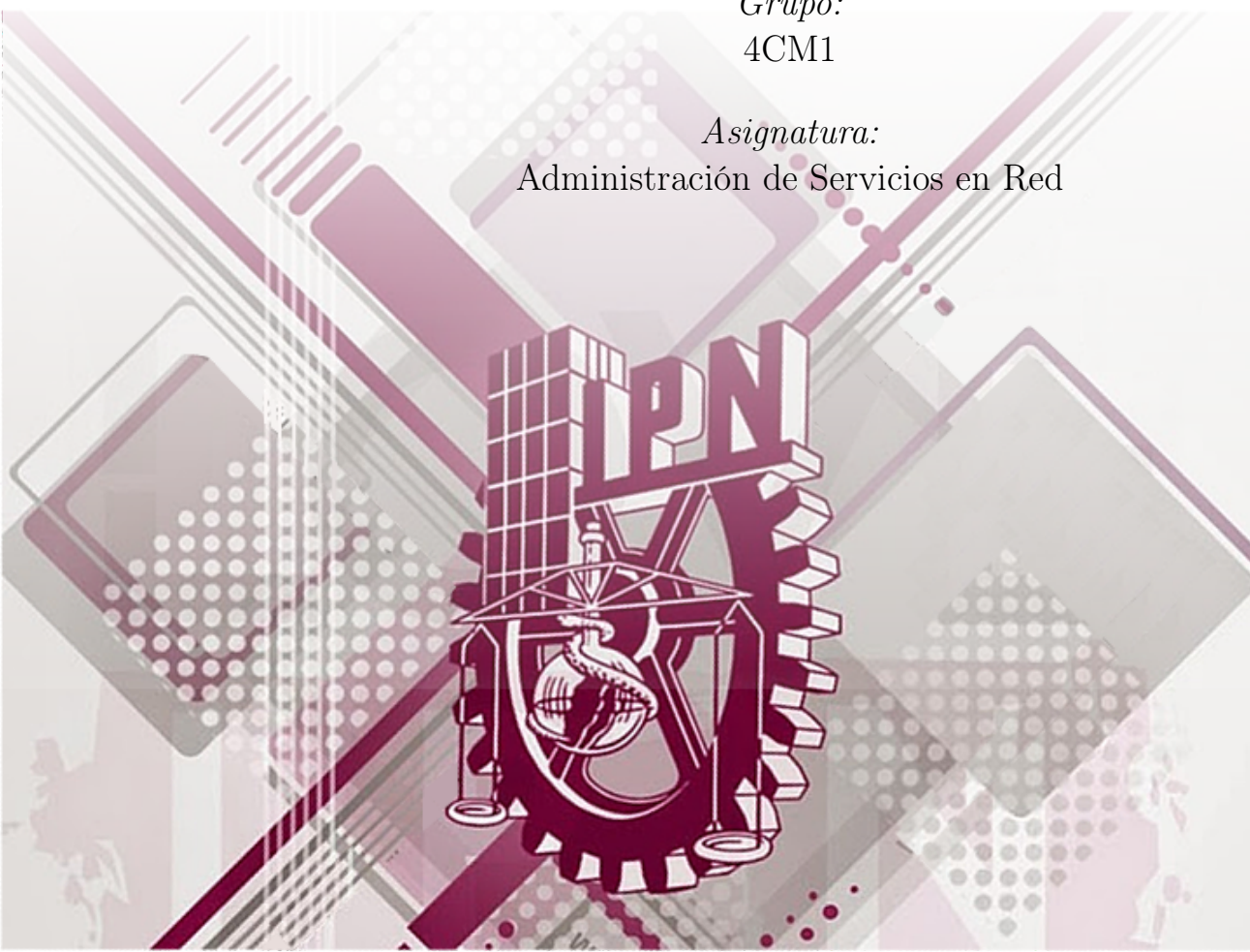
Tanibet

*Grupo:*

4CM1

*Asignatura:*

Administración de Servicios en Red



# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Marco teórico. Administración de configuración . . . . .	1
1.1.1. Flujo de proceso de alto nivel para la administración de la configuración. . .	2
1.1.2. Características de los sistemas de gestión de configuración de red. . . . .	2
1.1.3. Herramientas de configuración de red. . . . .	3
1.2. Introducción al problema 3. . . . .	3
<b>2. Configuración de dispositivos IP.</b>	<b>4</b>
2.1. Credenciales. . . . .	4
2.2. Persistencia. . . . .	4
2.3. Modificación de Nombre. . . . .	4
2.4. Asignación de IPs. . . . .	5
2.5. Enrutamiento estático. . . . .	5
2.6. Enrutamiento RIP. . . . .	5
2.7. Enrutamiento OSPF. . . . .	5
<b>3. Supervisión de Servidores.</b>	<b>6</b>
3.1. Supervisión de los servidores de correo electrónico. . . . .	6
3.1.1. Procedimiento . . . . .	6
3.2. Supervisión de los servidores web. . . . .	13
3.2.1. Procedimiento . . . . .	13
3.3. Supervisión de servidores de archivos. . . . .	16
3.3.1. Procedimiento . . . . .	16
3.4. Supervisión de impresoras. . . . .	21
3.4.1. Procedimiento . . . . .	21
3.5. Supervisión de acceso remoto. . . . .	26
3.5.1. Procedimiento . . . . .	26
<b>4. Recopilar la configuración de los dispositivos de manera periódica.</b>	<b>30</b>
4.1. Administración del archivo de configuración. . . . .	30
<b>5. Conclusiones</b>	<b>36</b>
5.1. López Ayala Eric Alejandro . . . . .	36
5.2. López Romero Joel . . . . .	36

Referencias . . . . .	37
-----------------------	----

# 1 | Introducción

## 1.1. Marco teórico. Administración de configuración

Es una recolección de procesos y herramientas que fomentan la consistencia de la red, realizan un seguimiento del cambio de red y proporcionan documentación y visibilidad de redes actualizadas. Si se crean y mantienen prácticas recomendadas de administración de la configuración, las ventajas podrían ser, una mejor disponibilidad de red y menores costos. Estos incluyen:

- Costos bajos de servicio técnico debido a una disminución de los problemas de soporte reactivo.
- Costos más bajos de red debido al uso de herramientas de seguimiento de dispositivo, circuito y usuario, y procesos que identifican componentes no usados de red.
- Disponibilidad de red mejorada debido a una disminución en los costos del soporte reactivo y tiempo optimizado para resolver problemas.

Los siguientes problemas surgen de una mala administración de la configuración.

- Incapacidad para determinar el impacto de los cambios de la red en el usuario.
- Más problemas de soporte reactivo y menor disponibilidad
- Mayor tiempo para resolver problemas
- Mayores costos de red debido a componentes de red no utilizados

[1]

La *administración de configuración* de red es el proceso de organizar y mantener información sobre todos los componentes de una red informática. Cuando una red necesita reparación, modificación, expansión o actualización, el administrador se refiere a la base de datos de gestión de configuración de red para determinar el mejor curso de acción. Esta base de datos contiene las ubicaciones y la dirección IP o la dirección de red de todos los dispositivos de hardware, así como información sobre la configuración predeterminada, los programas, las versiones y las actualizaciones instaladas en las computadoras de la red. [1]

### 1.1.1. Flujo de proceso de alto nivel para la administración de la configuración.

A continuación en la *Fig 1.1* se ilustra el modo de utilizar los factores esenciales del éxito seguidos de indicadores de rendimiento a fin de implementar un plan de administración de configuración exitosa. [1]

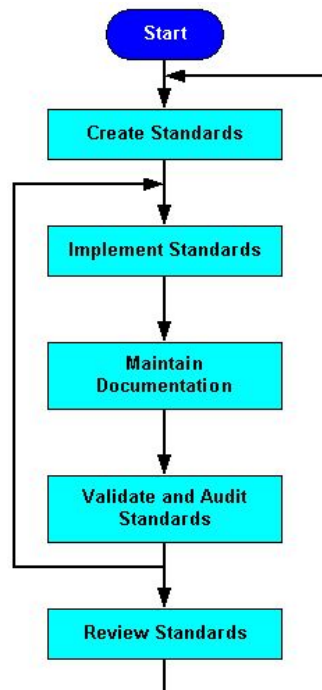


Figura 1.1: Diagrama

### 1.1.2. Características de los sistemas de gestión de configuración de red.

Una característica principal de la administración de configuración de red es su capacidad de reemplazar las funciones de un dispositivo de red en caso de falla. Los diferentes dispositivos de red guardan las configuraciones en diferentes formatos, y encontrar información de configuración puede ser difícil cuando un dispositivo en particular tiene que ser reemplazado. Con un sistema de gestión de la configuración de red instalado, la información de configuración se almacenará en un servidor ubicado centralmente, donde las configuraciones del dispositivo se pueden descargar fácilmente.

Los dispositivos que dependen de la interfaz de línea de comando necesitan un administrador confiable que use un protocolo estándar como el Protocolo de Transferencia Segura de Archivos para obtener la información necesaria, pero las copias de seguridad también se pueden automatizar, a menudo a través de un tercero. [2]

### 1.1.3. Herramientas de configuración de red.

Las herramientas de configuración de red pueden ser independientes del proveedor o específicas del proveedor. Las herramientas independientes del proveedor, por lejos las más comunes, están diseñadas para redes que contienen hardware y programas de múltiples proveedores. Las herramientas específicas del proveedor generalmente funcionan solo con los productos de un único proveedor y pueden ofrecer un mejor rendimiento en las redes donde ese proveedor domina el mercado.

Las herramientas de configuración se pueden utilizar para algo más que simplemente relanzar un dispositivo después de que falla. Algunas herramientas de administración de configuración rastrean los datos de configuración a diario para detectar cualquier cambio en los archivos de configuración, lo que podría revelar amenazas cibernéticas y posibles fallas. Las herramientas de configuración de red se pueden usar para crear cambios masivos. Por ejemplo, una empresa podría implementar rápidamente un cambio de contraseña si se filtran los datos de la contraseña. Además de realizar cambios en conjunto, las herramientas de administración de configuración se pueden usar para auditar e informar. Aunque no muestran información como la memoria o el rendimiento de la CPU, se pueden usar para mostrar reglas de firewall o parámetros VPN exactos.

Las herramientas de administración de configuración de red también tienen capacidades de informes, lo que permite al personal del sistema rastrear fácilmente la información sobre los componentes de la red. [2]

## 1.2. Introducción al problema 3.

De acuerdo a lo visto en clase, las especificaciones dadas para esta tercer evaluación, además de la información investigada. Dada una topología, se debe realizar la configuración por un protocolo en específico para tener conectividad entre los distintos segmentos de red. Para ello es necesario el uso de GNS3, una poderosa herramienta que permite desarrollar topologías de red complejas y poner simulaciones en marcha en estas topologías. Además del uso de RCP100 para simular routers. El objetivo de esta tercer evaluación es desarrollar e implementar una herramienta para la administración de configuración sobre una red y monitorizar ciertos aspectos en una máquina virtual con 5 servidores configurados (SMTP - IMAP, HTTP, FTP, Impresión, SSH). A continuación se presentan las distintas tareas realizadas para cumplir con el objetivo de la evaluación.

## 2 | Configuración de dispositivos IP.

A continuación se muestran varios comandos de configuración para RCP100.

### 2.1. Credenciales.

```
1   rcp login:  rcp
2   password:  rcp
```

### 2.2. Persistencia.

```
1   root      -> alt + f5
2   crear particion
3           -> fdisk /dev/sda
4           -> command: n
5           -> select: p
6           -> command: w
7   crear sistemas de archivos
8           -> mkfs.ext4 /dev/sda1
9           -> reboot
10          -> persist.sh
11          -> opcion: 1
12          -> reboot
```

### 2.3. Modificación de Nombre.

```
1   enable      ->   en
2   configure    ->   conf
3   nombreHost   ->   hostname nombre
4   guardar      ->   copy run start
```

## 2.4. Asignación de IPs.

```
1      enable      ->    en
2      configure   ->    conf
3      interfaz     ->    interface ethernet nombreInterfaz
4      asignar IP   ->    ip address direccionIP/bitsDeMascara
5      No Shutdown ->    no sh
6      guardar      ->    copy run start
7      estado Interfaces -> show interface
8      salir        ->    exit
```

## 2.5. Enrrutamiento estático.

```
1  configuracion -> conf
2  enrrutamiento -> ip route direccionIpDestino/bitsMask direccionIpSigSalto
3  mostrar tabla de enrrutamiento -> show ip route
```

## 2.6. Enrrutamiento RIP.

```
1      enable      ->    en
2      configure   ->    conf
3      RIP          ->    router rip
4      asignarIP    ->    network DireccionRedIp/bitsMaks
5      Distribuir ruta -> redistribute connected
6      mostrar Ips RIP -> show ip rip
```

## 2.7. Enrrutamiento OSPF.

```
1      enable      ->    en
2      configure   ->    conf
3      OSPF         ->    router ospf
4      asginar ip   ->    network direccionRedIp/mask area 0
5      distribuir   ->    redistribute connected
6      mostrar osp  ->    show ip ospf
```

[3]



## 3 | Supervisión de Servidores.

Todos los administradores de red deben supervisar la funcionalidad de los servidores que se encuentran disponibles dentro de la topología que se está supervisando.

### 3.1. Supervisión de los servidores de correo electrónico.

Todo administrador de red debe supervisar la funcionalidad de sus servidores de correo electrónico. Esto significa supervisar:

- Disponibilidad.
- Rendimiento.
- Entrega de los correo electrónicos sin errores.

#### 3.1.1. Procedimiento

1. El primer paso es la instalación y configuración del servidor de correo electrónico, para ello se hizo uso del servidor de correo electrónico **hMailServer**, el cual nos ofrece servicios que implementan los protocolos: **IMAP**, **POP3** y **SMTP**.

#### Instalación y Configuración

- Lo primero es descargar el archivo ejecutable de instalación del servidor de la página: <https://www.hmailserver.com/download>, (Nota: El servidor solo puede ser instalado en OS Windows o usando la herramienta Wine en Linux).
  - Se procede con el proceso de instalación, tal y como lo indica el wizard.
  - Por último una vez instalado, se define una contraseña para poder acceder a la configuración del servidor, esta contraseña será solicitada cada vez que se cierre el proceso del servidor.
2. Una vez instalado y configurado el servidor, ya podemos realizar la supervisión del mismo; para la supervisión se implemento un *Sensor SMTP & IMAP Round Trip*. Este sensor hace

uso de los protocolos **SMTP** e **IMAP** para supervisar la entrega de correo electrónico de extremo a extremo.

El sensor de ida y vuelta de SMTP e IMAP supervisa el tiempo que tarda un correo electrónico en llegar a un buzón del Protocolo de acceso a mensajes de Internet (IMAP) después de enviarse mediante el Protocolo simple de transferencia de correo (SMTP). Se envía un correo electrónico utilizando el dispositivo principal como servidor SMTP y luego escanea un buzón IMAP dedicado hasta que llega este correo electrónico.

## Implementación

El primer elemento a implementar para poder hacer el Round-Trip es el sensor SMTP, el cuál es el encargado de enviar el correo electrónico de prueba y sensar la respuesta del servidor SMTP. A continuación se muestra el código que implementa este comportamiento:

### smtp sensor

```
1 import smtplib
2 from email.mime.multipart import MIMEMultipart
3 from email.mime.text import MIMEText
4 import time
5
6 #Establish configuration parameters
7 hostname = "10.10.10.3"
8 username = "administrador@mailserver.asr"
9 destination = "servidorsmtp@mailserver.asr"
10 password = "123456"
11
12 def open_connection(verbose):
13     #Connect to the server
14     if verbose: print('Connecting to', hostname)
15     connection = smtplib.SMTP(hostname, 25)
16
17     # Login to our account
18     if verbose: print('Logging in as', username)
19     connection.login(username, password)
20     return connection
```

### main smtp

```
1 def smtpSensor():
2     start = time()
3     conn = open_connection(verbose=False)
```

```

4
5     # Assembling a email basic header
6     fromaddr = "administrador@mailserver.asr"
7     toaddr = "servidoresmtpt@mailserver.asr"
8     msg = MIMEMultipart()
9     msg['From'] = fromaddr
10    msg['To'] = toaddr
11    msg['Subject'] = "Correo de prueba del equipo # del grupo 4CM1"
12
13    # Now we attach the body of the email to the MIME message
14    body = "Eric Alejandro Lopez Ayala\nJoel Lopez Romero"
15    msg.attach(MIMEText(body, 'plain'))
16    text = msg.as_string()
17
18    conn.sendmail(fromaddr, toaddr, text)
19    end = time()
20
21    logging.debug("=====")
22    logging.debug("SMTP SENSOR")
23    logging.debug('Send message ::::::::::::::::::::')
24    logging.debug(text)
25    logging.debug("SMTP response time took ("
26    + str(end - start)
27    + " seconds passed)")

```

El segundo elemento requerido el Round-Trip es el sensor IMAP, el cual es el encargado de leer la bandeja de entrada de correo electrónico del email destino e identificar los correos de prueba que llegan desde el servidor SMTP, así mismo se encarga de sensar la respuesta del servidor IMAP. A continuación se muestra el código que implementa este comportamiento:

### imap sensor

```

1  import imaplib
2  import email
3  import time
4  from itertools import chain
5
6  def search_string(uid_max, criteria):
7      c = list(map(lambda t: (t[0],
8          "'"+str(t[1])+"'",
9          criteria.items()))
10     + [( 'UID', '%d:*' % (uid_max+1))])
11     return '(%s)' % ' '.join(chain(*c))
12     # Produce search string in IMAP format:
13     # e.g. (FROM "me@gmail.com" SUBJECT "abcde" BODY "123456789" UID 9999:*)
14

```

```
15 def get_first_text_block(msg):
16     type = msg.get_content_maintype()
17
18     if type == 'multipart':
19         for part in msg.get_payload():
20             if part.get_content_maintype() == 'text':
21                 return part.get_payload()
22     elif type == 'text':
23         return msg.get_payload()
24
25 def open_connection(verbose):
26     # Stablish configuration parameters
27     username = "servidorimap@mailserver.asr"
28     password = "123456"
29     hostname = "10.10.10.3"
30
31     # Connect to the server
32     if verbose: print('Connecting to', hostname)
33     connection = imaplib.IMAP4(hostname)
34
35     # Login to our account
36     username = username
37     password = password
38     if verbose: print('Logging in as', username)
39     connection.login(username, password)
40     return connection
```

### main imap

```
1 def imapSensor():
2     conn = open_connection(verbose=False)
3     try:
4         logging.debug(conn)
5
6         # Restrict mail search. Be very specific.
7         # Machine should be very selective to receive messages.
8         criteria = {
9             'FROM': 'administrador@mailserver.asr' # ,
10             # 'SUBJECT': 'Correo de prueba del equipo # del grupo 4CM1',
11             # 'BODY': 'Eric Alejandro Lopez Ayala\nJoel Lopez Romero',
12         }
13
14         # We stablish the max uid from mailbox
15         uid_max = 0
16
17         # We select the objects that are ar inbox
18         conn.select('INBOX')
```

```
19     result, data = conn.uid('search', None, search_string(uid_max, criteria))
20
21     uids = [int(s) for s in data[0].split()]
22     if uids:
23         uid_max = max(uids)
24     # Initialize `uid_max`. Any UID less than or equal to `uid_max`
25     # will be ignored subsequently.
26
27     # We close the connection
28     conn.logout()
29
30     # Keep checking messages ...
31     logging.debug("=====")
32     logging.debug("IMAP SENSOR")
33     while True:
34         start = time()
35         end = 0
36         # Have to login/logout each time because that's
37         # the only way to get fresh results.
38         conn = open_connection(verbose=False)
39         conn.select('INBOX')
40
41         result, data = conn.uid('search',
42                                 None,
43                                 search_string(uid_max, criteria))
44
45         uids = [int(s) for s in data[0].split()]
46
47         for uid in uids:
48             # Have to check again in case the UID criterion is not obeyed
49             if uid > uid_max:
50                 result, data = conn.uid('fetch', str(uid), '(RFC822)')
51                 # Fetch entire message
52                 msg = email.message_from_string(str(data[0][1],
53                                                     encoding="utf-8"))
54
55                 uid_max = uid
56                 # print(uid_max)
57
58                 text = get_first_text_block(msg)
59                 logging.debug('New message ::::::::::::::::::::')
60                 logging.debug("Message Body: " + text)
61                 end = time()
62                 logging.debug("IMAP response time took: ("
63                               + str(end - start)
64                               + " seconds passed)" )
65         conn.logout()
```

```

66         time.sleep(1)
67         logging.debug("=====")
68     finally:
69         conn.logout()

```

3. Una vez que se han implementado, se procede a realizar el sensado del servidor de correos, el sensor muestra la siguiente información:

- Tiempo de respuesta del servidor SMTP.
- Contenido del correo electrónico de prueba enviado.
- Tiempo de respuesta del servidor IMAP.
- Contenido del correo electrónico de prueba recibido.

A continuación se muestran las capturas de pantalla que reflejan el comportamiento de este sensor, las pruebas se realizaron al servidor de correos con la IP: 10.10.10.2:

## Pruebas

```

(venv) ESCOM@Linux:~/Documents/Monitoring/Sensors$ python monitoring_sensors.py
=====
Administration monitor sensor
Traceback (most recent call last):
  File "monitoring_sensors.py", line 199, in <module>
    ssh_thread = threading.Thread(sssh_sensor(), name="SSH Sensor", args=[])
NameError: name 'sssh_sensor' is not defined
[DEBUG] - SMTP Sensor : =====
[DEBUG] - SMTP Sensor : SMTP SENSOR
[DEBUG] - SMTP Sensor : Send message ::::::::::::::::::::
[DEBUG] - SMTP Sensor : Content-Type: multipart/mixed; boundary="=====8431881314442361020=="
MIME-Version: 1.0
From: administrador@mailserver.asr
To: servidoressmtp@mailserver.asr
Subject: Correo de prueba del equipo # del grupo 4CM1

--=====8431881314442361020==
Content-Type: text/plain; charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit

Eric Alejandro Lopez Ayala
Joel Lopez Romero
--=====8431881314442361020==--

[DEBUG] - SMTP Sensor : SMTP response time took (0.6284189224243164 seconds passed)
[DEBUG] - SMTP Sensor : =====

```

Figura 3.1: Prueba Sensor SMTP.

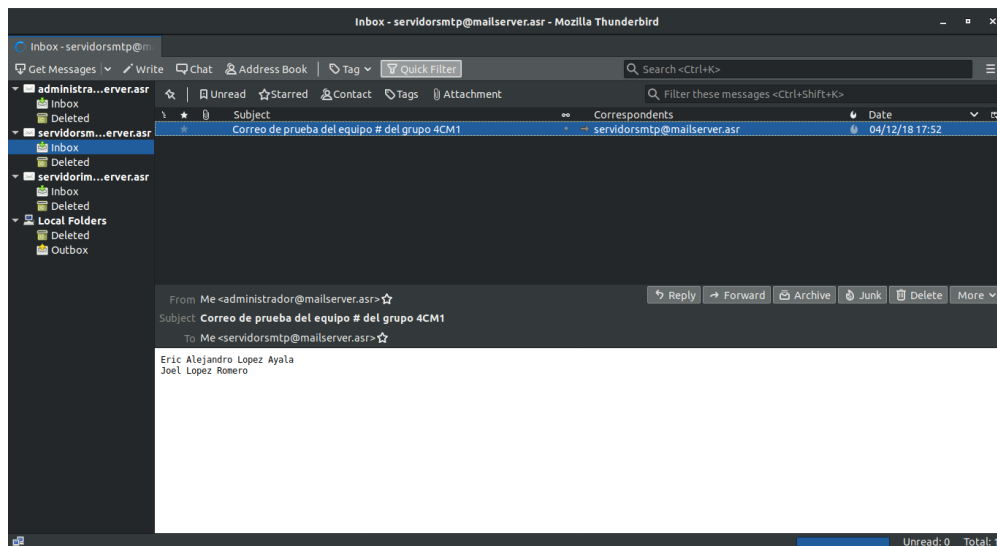


Figura 3.2: Bandeja de correo electrónico SMTP.

```
(venv) ESCOM@linux:~/Documents/Monitoring/Sensors/IMAP$ python imap_sensor.py
<imaplib.IMAP4 object at 0x7ff308f360f0>
New message ::::::::::::::::::::
Eric Alejandro Lopez Ayala
Joel Lopez Romero
IMAP response time took: (0.69402623 seconds passed):
```

Figura 3.3: Prueba Sensor SMTP.

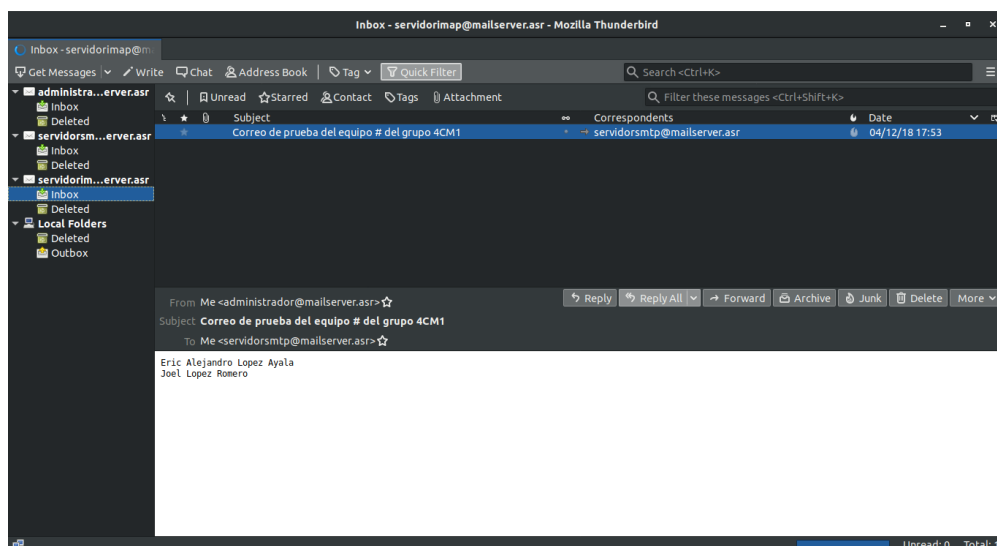


Figura 3.4: Bandeja de correo electrónico IMAP.

## 3.2. Supervisión de los servidores web.

El rendimiento un sitio web es un factor decisivo para muchas empresas, mucho más cuando ellas ofrecen productos y servicios a través de sus sitios web. Esto significa supervisar:

- Tiempo de respuesta (carga) de una solicitud HTTP.
- Ancho de banda de la conexión.
- Bytes recibidos/enviados durante la petición.

### 3.2.1. Procedimiento

1. El primer paso es la instalación y configuración del servidor HTTP, para ello se hizo uso de un built-in que viene incorporado dentro de python nativo del OS Linux.

**Instalación y Configuración** En OS Linux que poseen python 2.7 en adelante poseen un paquete que permite implementar un servidor HTTP simple mediante una línea de comando de shell, para dar de alta el servidor HTTP se ejecuta el siguiente comando:

```
1 python -m SimpleHTTPServer 8000
```

Con esta línea de comando montamos un servidor HTTP simple en el puerto 8000, tal y como se muestra en la siguiente pantalla:

2. Una vez instalado y configurado el servidor, ya podemos realizar la supervisión del mismo; para la supervisión se implementó un *Sensor HTTP*. Este sensor implementa el protocolo HTTP, el cual le permite recibir peticiones **GET**, **POST**, **DELETE** y **UPDATE**.

### Implementación

El sensor HTTP se encarga de realizar una petición POST al servidor, en la cual se envían un conjunto de datos *payload*, y recibe los datos de sensado como respuesta a la petición. A continuación se muestra el código que implementa este comportamiento:

#### http sensor

```
1 # Here we import 2 libraries required for the http sensor
2 # time: Required for measuring time of the request
3 # request: API required handle HTTP requests and responses
4 import time
5 import requests
6
7 def sense(url):
8     try:
9         # This is the code used to do an HTTP request and obtaining the time in
10         start_time = time.time()
```



```
11     # with eventlet.Timeout(100):
12     payload = {'Equipo': '10', 'Grupo': '4CM1'}
13     response = requests.post(url, payload)
14     end_time = time.time()
15
16     roundtrip = end_time - start_time
17
18     # Then from the HTTP response we read and calculate the data per response
19     # The for each and sum are in case that the response
20     # data it is too big so we cut them into chunks
21     with response as r:
22         size = sum(len(chunk) for chunk in response.iter_content(8196))
23         # The size of the chunk is 8kbytes
24
25     # Then finally we calculate the HTTP download rate,
26     # this is done by using the total request time and its size
27     # First we do a conversion from bytes to bits
28     size_in_bits = size * 8
29
30     # Then we do a division rate to obtain the total number of bits per second
31     download_rate = size_in_bits / roundtrip
32
33     # Finally we print the response code of the request
34     response_code = response.status_code
35
36     return roundtrip, size, size_in_bits, download_rate, response_code
37
38 except requests.exceptions.RequestException as e:
39     # Here we handle the exception
40     print(e)
41     # We return an all 0 list
42     return [0, 0, 0]
```

### main http

```
1 def httpSensor():
2     test_web_pages = ['http://10.10.10.2:8000/']
3     # Hay que cambiar la IP de aqui xd
4
5     logging.debug("=====")
6     logging.debug("HTTP SENSOR")
7
8     for web_page in test_web_pages:
9         logging.debug("*****")
10        logging.debug("Doing test to: " + web_page)
11        roundtrip, size, size_in_bits, download_rate, response_code = sense(web_page)
12
```

```

13     logging.debug('Total time: ' + str(roundtrip))
14     # Finally we print the result time in seconds.
15     logging.debug('Total recieved bytes: ' + str(size))
16     logging.debug('Total recieved bits: ' + str(size_in_bits))
17     logging.debug('Download rate in bits/s: ' + str(int(download_rate)))
18     logging.debug('Response code:' + str(response_code))
19     logging.debug("*****")
20
21     logging.debug("=====")

```

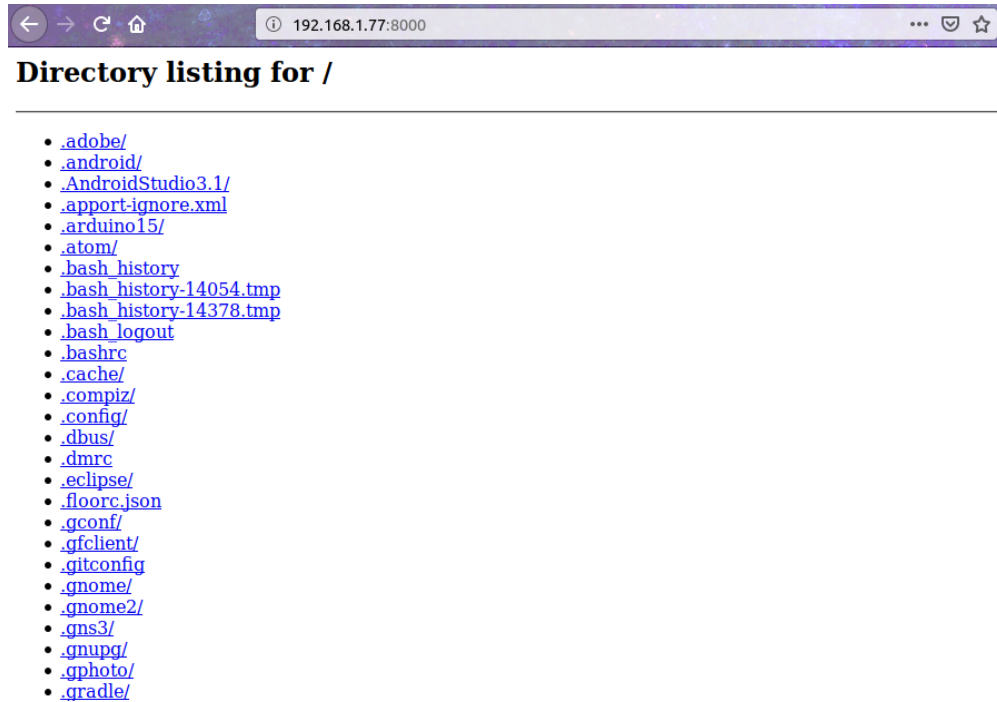


Figura 3.5: Servidor HTTP Ejecutando.

3. Una vez que se ha implementado el sensor, se procede realizar el sensado del servidor HTTP, el sensor realiza una petición POST al servidor, con lo cual muestra la siguiente información:
  - Tiempo de respuesta del servidor HTTP.
  - Número e bytes que ha recibido de la petición.
  - Velocidad de ancho de descarga de la petición.
  - Código de respuesta del servidor.

A continuación se muestra la captura de pantalla que refleja el comportamiento de este sensor, las pruebas se realizarán al servidor HTTP con la IP: 10.10.10.2:

## Pruebas

```

=====
Administration monitor sensor
[DEBUG] - HTTP Sensor : =====
=====
[DEBUG] - HTTP Sensor : HTTP SENSOR
[DEBUG] - HTTP Sensor : *****
[DEBUG] - HTTP Sensor : Doing test to: http://10.10.10.2:8000/
[DEBUG] - HTTP Sensor : Starting new HTTP connection (1): 10.10.10.2:8000
[DEBUG] - HTTP Sensor : http://10.10.10.2:8000 "POST / HTTP/1.1" 501 497
[DEBUG] - HTTP Sensor : Total time: 0.11972522735595703
[DEBUG] - HTTP Sensor : Total recieved bytes: 497
[DEBUG] - HTTP Sensor : Total recieved bits: 3976
[DEBUG] - HTTP Sensor : Download rate in bits/s: 33209
[DEBUG] - HTTP Sensor : Response code:501
[DEBUG] - HTTP Sensor : *****
[DEBUG] - HTTP Sensor : =====

```

Figura 3.6: Prueba Sensor HTTP.

### 3.3. Supervisión de servidores de archivos.

Los servidores FTP son centros de descargas que proporcionan datos, aplicaciones, controladores y actualizaciones de software a sus clientes y compañeros de trabajo.

Si las empresas no pueden acceder a la información entonces nada funciona. Por lo tanto, una de las principales tareas de los administradores es asegurar que el servidor de archivos está disponible y que funciona sin ningún tipo de problema. Esto significa supervisar:

- Disponibilidad.
- Rendimiento.
- Capacidad de subir y descargar archivos.

#### 3.3.1. Procedimiento

1. El primer paso es la instalación y configuración del servidor FTP, para ello se hizo uso del servidor vsFTPD, el cual será el encargado de proporcionar todos los servicios que establece el protocolo **FTP**.

#### Instalación y Configuración

- Para instalar el vsFTPD en un SO Linux, abrimos una terminal y ejecutamos el siguiente comando:

```
1          sudo apt-get install vsftpd
```

El comando anterior instalará e inicializará el servidor FTP.

```
1          Setting up vsftpd (2.3.5-1ubuntu2) ...  
2          vsftpd start/running, process 1891
```

- Después una vez instalado, vamos al archivo de configuración de la ruta `/etc/vsftpd.conf` y modificamos las siguientes líneas:

```
1          listen = YES  
2          local_enable=YES  
3          write_enable=YES  
4          userlist_enable=YES  
5          userlist_deny=NO
```

- Por último guardamos los cambios en el archivo y procedemos a reiniciar el servicio:

```
1          sudo service vsftpd restart
```

Una vez terminada la instalación y configuración del servidor FTP, podemos realizar conexiones al mismo, tal y como se muestra en la pantalla:

```
$ ftp localhost  
Connected to localhost.  
220 (vsFTPd 2.3.5)  
Name (localhost:root): anonymous  
331 Please specify the password.  
Password:  
230 Login successful.  
Remote system type is UNIX.  
Using binary mode to transfer files.  
ftp>  
ftp> quit  
221 Goodbye.
```

Figura 3.7: Servidor FTP Ejecutando.

2. Una vez instalado y configurado el servidor, ya podemos realizar la supervisión del mismo: para la supervisión se implemento un *Sensor FTP*. Este sensor implementa el protocolo FTP, el cual permite realizar operaciones de **READ** y **WRITE**.

## Implementación

El sensor FTP se encarga de realizar la conexión con el servidor y realizar operaciones de lectura y escritura al mismo, con ello puede sensar datos como la velocidad de conexión y disponibilidad del servidor. A continuación se muestra el código que implementara este comportamiento:

**ftp sensor**

```
1 import ftplib
2 from time import time
3
4 FTP_HOST = "10.10.10.2"
5 FTP_USERNAME = "servidores"
6 FTP_PASSWORD = "12345678"
7
8 ftp = None
9 TEST_FILE = "configuracion.txt"
10
11 def getFile(ftp, file):
12     try:
13         ftp_file = open(file, 'wb')
14         ftp.retrbinary('RETR ' + file, ftp_file.write)
15         ftp.quit()
16         return 0
17     except Exception as e:
18         print(e)
19         return -1
20
21 def setFile(ftp):
22     try:
23         ftp_file = open(TEST_FILE, 'rb')
24         ftp.storbinary('STOR ' + TEST_FILE, ftp_file)
25         return 0
26     except Exception as e:
27         print(e)
28         return -1
29
30 def doSubScanning(ftp, dir_path):
31     try:
32         ftp.cwd(dir_path)
33         subdirectory_content = ftp.nlst()
34         files = len(subdirectory_content)
35         file_scan = ""
36
37         if (files > 0):
38             for directory in subdirectory_content:
39                 file_scan = file_scan
40                 + "Scanning directory/file: "
41                 + directory + "\n"
42                 if "." not in directory:
43                     file_scan = file_scan
44                     + doSubScanning(ftp, dir_path + "/" + directory + "/")
45             return file_scan
46     except Exception as e:
47         print(e)
```

```

48         return -1
49
50 def doServerScanning(ftp):
51     try:
52         root_directory_content = ftp.nlst()
53         file_scan = ""
54         file_scan = file_scan + "Scanning: Server Root Directory\n"
55         for directory in root_directory_content:
56             file_scan = file_scan + "Scanning directory/file: " + directory + "\n"
57             #if "." not in directory:
58                 #file_scan = file_scan + doSubScanning(ftp, "/" + directory + "/"
59         return file_scan
60     except Exception as e:
61         print(e)
62         return -1

```

### main ftp

```

1 def ftpSensor():
2     # Connect to host, default port
3     ftp = ftplib.FTP(FTP_HOST)
4     # Do login
5     ftp.login(FTP_USERNAME, FTP_PASSWORD)
6
7     logging.debug("=====")
8     logging.debug("FTP SENSOR")
9     # Get welcome message from server
10    logging.debug(ftp.getwelcome())
11    start = time()
12    # Doing FTP File Upload
13    result = setFile(ftp)
14    logging.debug("Upload of file sucessful")
15    end = time()
16    logging.debug("FTP response time: (%.2f seconds)" %(end - start))
17    # Doing Sensor FTP Server File Count
18    logging.debug(doServerScanning(ftp))
19    logging.debug("=====")
20    # Quitting FTP Server Connection
21    ftp.quit()

```

- Una vez que se ha implementado el sensor, se procede a realizar el sensado del servidor FTP, el sensor realiza un handshake con el servidor para verificar la disponibilidad del mismo, sube un archivo al servidor y por último realiza un recorrido sobre su directorio raíz, mostrando la siguiente información:

- Tiempo de respueta del servidor.

- Archivo de carga al servidor.
- Contenido del directorio raíz del servidor.

A continuación se muestra la captura de pantalla que refleja el comportamiento de este sensor, las pruebas se realizaron al servidor FTP con la IP: 10.10.10.2

## Pruebas

```
[DEBUG] - FTP Sensor : =====  
[DEBUG] - FTP Sensor : FTP SENSOR  
[DEBUG] - FTP Sensor : 220 (vsFTPd 3.0.3)  
[DEBUG] - FTP Sensor : Upload of file sucessful  
[DEBUG] - FTP Sensor : FTP response time: (0.08 seconds)  
[DEBUG] - FTP Sensor : Scanning: Server Root Directory  
Scanning directory/file: Descargas  
Scanning directory/file: Desktop  
Scanning directory/file: Documentos  
Scanning directory/file: Imágenes  
Scanning directory/file: Música  
Scanning directory/file: Plantillas  
Scanning directory/file: Público  
Scanning directory/file: Vídeos  
Scanning directory/file: archivo.pcap  
Scanning directory/file: archivo1.py  
Scanning directory/file: archivoPrueba.txt  
Scanning directory/file: configuracion.txt  
Scanning directory/file: http.pcap  
Scanning directory/file: prueba2MB  
Scanning directory/file: ssh.pcap  
  
[DEBUG] - FTP Sensor : =====
```

Figura 3.8: Prueba Sensor FTP.

## 3.4. Supervisión de impresoras.

Las impresoras deben trabajar y no interrumpir los flujos de trabajo. Los empleados no deben tener que considerar si la impresora está lista para operar.

El servidor CUPS nos permite administrar la configuración de las impresoras dentro de una red de computadora, por lo cual nos permite supervisar el funcionamiento de cada una de ellas.

### 3.4.1. Procedimiento

1. El primer paso es la instalación y configuración del servidor de Impresoras, para ello se hizo uso del servidor **CUPS** que viene por defecto que viene en los OS Linux.

#### Instalación y Configuración

- Por defecto cups ya viene preinstalado los OS Linux por lo cual abrimos nuestro navegador y en la barra de direcciones insertamos el siguiente enlace: <http://localhost:631/>; en caso de estar habilitado se mostrará la siguiente pantalla:

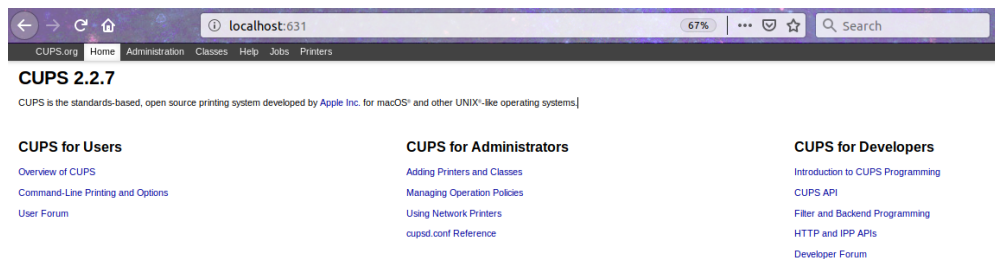


Figura 3.9: Servidor CUPS Ejecutandose.

- Después nos vamos al módulo de administración y agregamos una nueva impresora, tal y como se muestra a continuación:
  - Seleccionamos la impresora y que tipo de red estará conectada.
  - Configuramos la información de descripción de la impresora.
  - Por último seleccionamos el tipo y modelo de la impresora de un catálogo definido.
2. Una vez instalado y configurado el servidor, ya podemos realizar la supervisión del mismo, para la supervisión se implemento un *Sensor CUPS*.

#### Implementación

El sensor de impresión supervisa varios tipos d impresoras usando CUPS. Este sensor implementa las directrices que requiere cups, con ello se puede sensar datos como la descripción



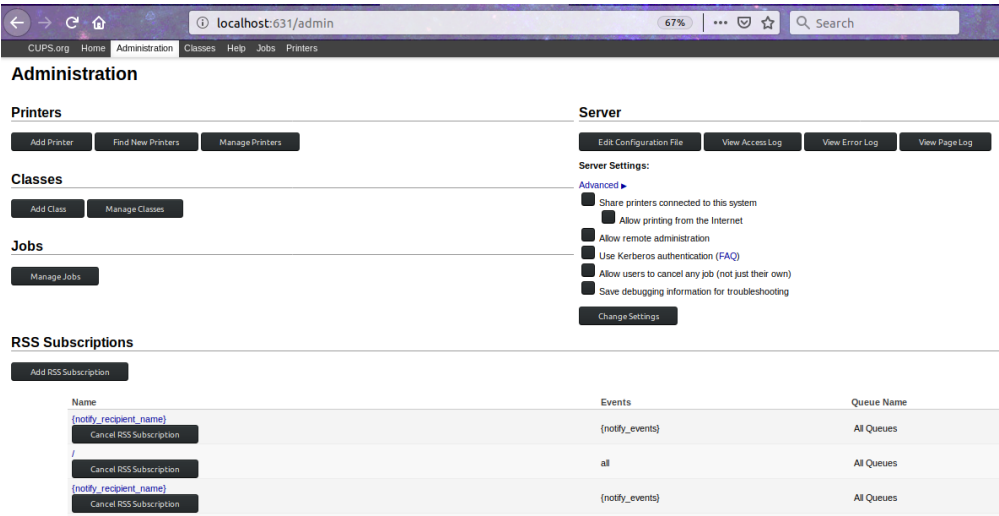


Figura 3.10: Agregar nueva impresora.

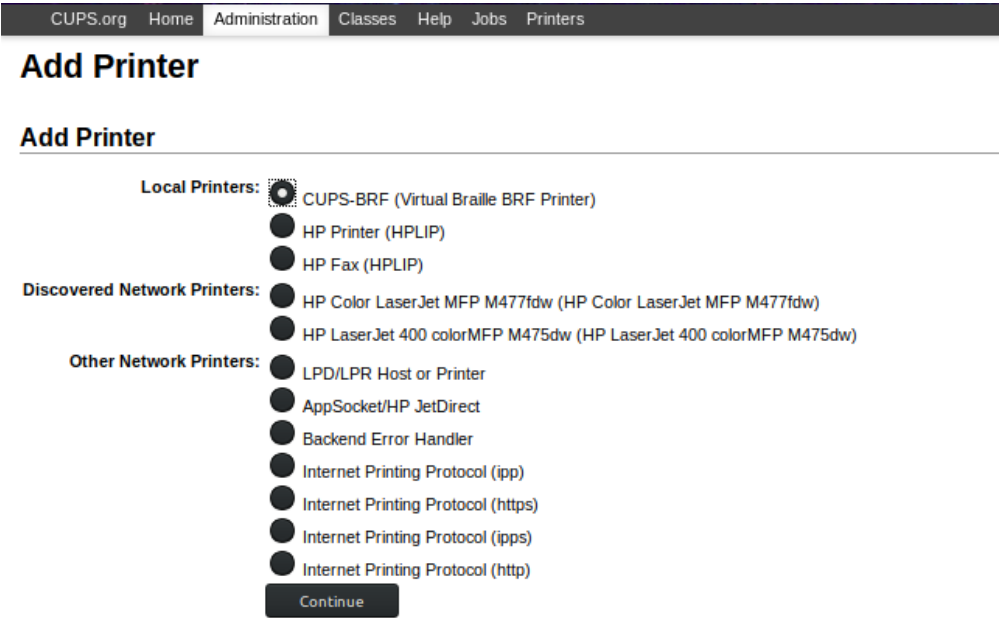
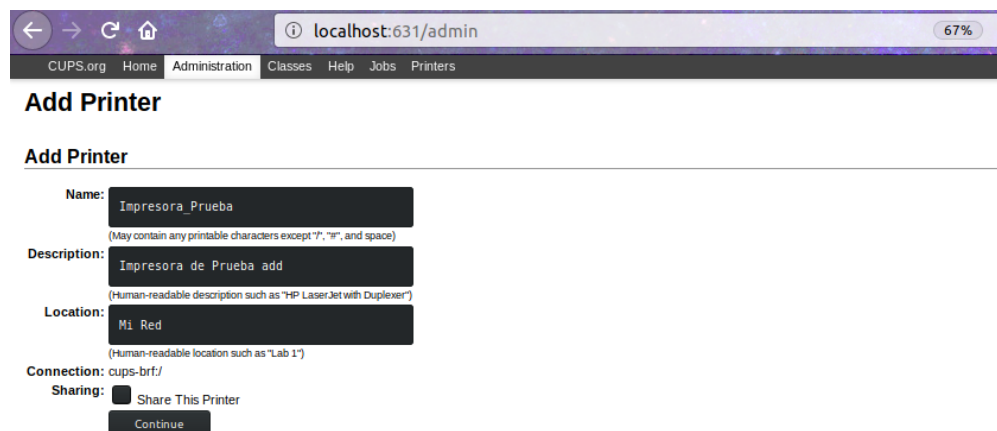


Figura 3.11: Seleccionar impresora.

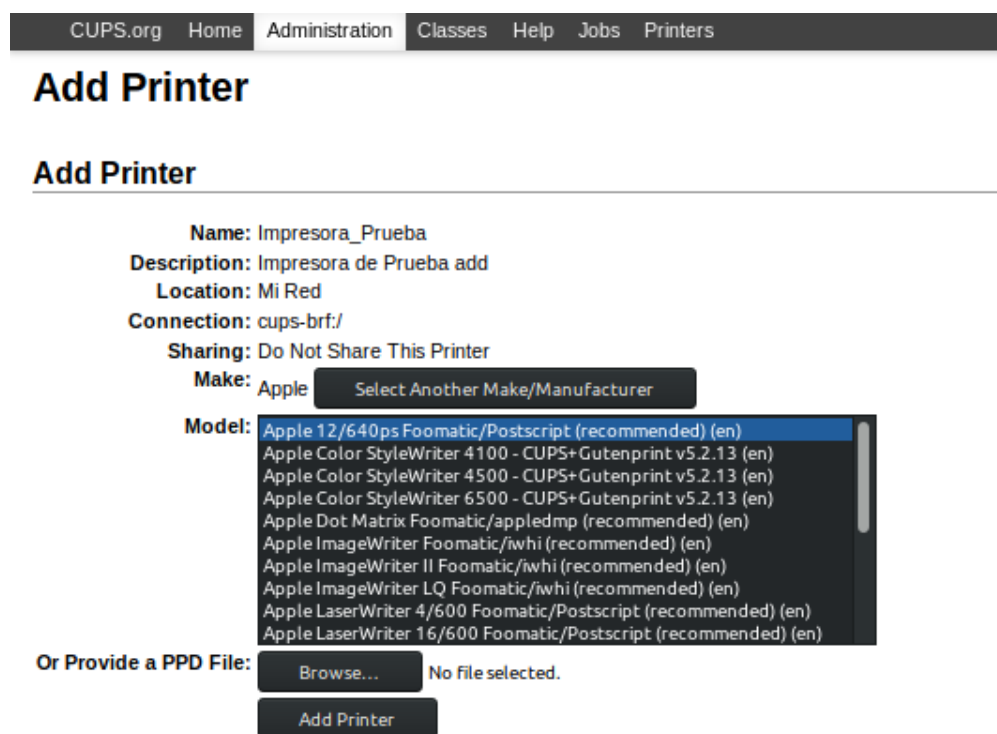


The screenshot shows the CUPS.org Administration interface at localhost:631/admin. The 'Add Printer' form is displayed with the following fields:

- Name:** Impresora\_Prueba
- Description:** Impresora de Prueba add
- Location:** Mi Red
- Connection:** cups-brf:/
- Sharing:** ☐ Share This Printer

A 'Continue' button is visible at the bottom of the form.

Figura 3.12: Configuración de información.



The screenshot shows the CUPS.org Administration interface at localhost:631/admin. The 'Add Printer' form is displayed with the following fields:

- Name:** Impresora\_Prueba
- Description:** Impresora de Prueba add
- Location:** Mi Red
- Connection:** cups-brf:/
- Sharing:** Do Not Share This Printer
- Make:** Apple (with a dropdown menu to 'Select Another Make/Manufacturer')
- Model:** A list of printer models is shown, with 'Apple 12/640ps Foomatic/Postscript (recommended) (en)' selected.

Below the model list, there is a section 'Or Provide a PPD File:' with a 'Browse...' button and the text 'No file selected.' An 'Add Printer' button is at the bottom.

Figura 3.13: Seleccionar tipo de impresora.

que la impresora ofrece. A continuación se muestra el código que implementara este comportamiento:

### **cups sensor**

```
1 import spur
2
3 def doLogin():
4     # Spur command shell execution
5     shell = spur.SshShell(
6         hostname="10.10.10.2",
7         username="servidores",
8         password="12345678",
9         port = "22",
10        missing_host_key = spur.ssh.MissingHostKey.accept
11    )
12    return shell
```

### **main cups**

```
1 def cupsSensor():
2     #We do login and obtain an instance of the shell
3     shell = doLogin()
4
5     #We run the command prompt from the shell
6     with shell:
7         result = shell.run(["lpstat", "-p"])
8
9     #Then we parse the result and we process the text data
10    if result.return_code == 0:
11        lines = str(result.output, encoding="utf-8").split("\n")
12        logging.debug("=====")
13        logging.debug("CUPS SENSOR")
14        for line in lines:
15            if (line != ""):
16                data = str(line).split(" ")
17
18                logging.debug("*****")
19                printer_name = data[1]
20                printer_last_conn = " ".join(data[5:12])
21
22                logging.debug("Printer name: " + str(printer_name))
23                logging.debug("First enabled configuration: "
24                    + str(printer_last_conn))
25                logging.debug("*****")
26
27        logging.debug("=====")
```

```

28         logging.debug("Total number of printers: "
29         + str(len(lines) - 1))
30     else:
31         str(result.stderr_output, encoding="utf-8")
32         # prints the stderr output code

```

- Una vez que se ha implementado el sensor, se procede a realizar el sensado del servidor CUPS, el sensor se conecta mediante SSH al servidor y ejecuta un comando de shell para realizar la consulta al servidor. Esta consulta se procesa y se obtiene la siguiente información:

- Nombre de la impresora.
- Cuando fue habilitada por primera vez.
- Estado de la impresora.

A continuación se muestra la captura de pantalla que refleja el comportamiento de este sensor, las pruebas se realizaron al servidor CUPS con la IP: 10.10.10.2

## Pruebas

```

[INFO] - Thread-1 : Authentication (password) successful!
[DEBUG] - CUPS Sensor : [chan 0] Max packet in: 32768 bytes
[DEBUG] - Thread-1 : Received global request "hostkeys-00@openssh.com"
[DEBUG] - Thread-1 : Rejecting "hostkeys-00@openssh.com" global request from server.
[DEBUG] - Thread-1 : [chan 0] Max packet out: 32768 bytes
[DEBUG] - Thread-1 : Secsh channel 0 opened.
[DEBUG] - Thread-1 : [chan 0] Secsh channel 0 request ok
[DEBUG] - Thread-1 : [chan 0] EOF received (0)
[DEBUG] - Thread-1 : EOF in transport thread
[DEBUG] - CUPS Sensor : =====
[DEBUG] - CUPS Sensor : CUPS SENSOR
[DEBUG] - CUPS Sensor : =====
[DEBUG] - CUPS Sensor : Printer name: impresora
[DEBUG] - CUPS Sensor : First enabled configuration: activada desde mié 05 dic 2018
[DEBUG] - CUPS Sensor : =====
[DEBUG] - CUPS Sensor : =====
[DEBUG] - CUPS Sensor : Printer name: impresora
[DEBUG] - CUPS Sensor : First enabled configuration: activada desde dom 25 nov 2018
[DEBUG] - CUPS Sensor : =====
[DEBUG] - CUPS Sensor : =====
[DEBUG] - CUPS Sensor : Printer name: impresora
[DEBUG] - CUPS Sensor : First enabled configuration: activada desde sáb 01 dic 2018
[DEBUG] - CUPS Sensor : =====
[DEBUG] - CUPS Sensor : =====
[DEBUG] - CUPS Sensor : Total number of printers: 3
(venv) ESCOM@Linux:~/Documents/Monitoring/Sensors$ █

```

Figura 3.14: Prueba Sensor CUPS.

## 3.5. Supervisión de acceso remoto.

El acceso remoto debe ser controlado en todo momento. Un administrador de red debe controlar las conexiones de acceso remoto a su red.

El servidor SSH nos permite generar usuarios y sesiones para que se puedan conectar de forma remota a un dispositivo y tomar control de el.

### 3.5.1. Procedimiento

1. El primer paso es la instalación y configuración del servidor SSH, para ello se hizo uso del servidor *openssh-server*, el cual implementa todas las funcionalidades de un servidor SSH.

#### Instalación y Configuración

- Abrimos una terminal de nuestro OS Linux y ejecutamos el siguiente comando para instalar el servidor:

```
1 sudo apt-get install openssh-server
```

- Una vez instalado el servidor se procede a realizar la configuración, la configuración se realiza modificando el archivo configuración de la ruta */etc/ssh/sshd\_config* y modificamos las siguientes lineas:

```
1 PORT 22
2 PermitRootLogin yes
```

- Por último guardamos los cambios en el archivo y procedemos a reiniciar el servicio:

```
1 sudo service sshd restart
```

Una vez terminada la instalación y configuración del servidor SSH, podemos realizar conexiones al mismo, tal y como se muestra en la pantalla:

2. Una vez instalado y configurado el servidor, ya podemos realizar la supervisión del mismo, para la supervisión se implemento un *Sensor SSH*.

#### Implementación

El sensor SSH supervisa varios aspecto de una conexión remota SSH. Este sensor implementa el protocolo SSH, con ello puede sensar los datos de una sesión SSH y mediante sniffing supervisa los datos que se envia por el canal de comunicación al puerto del servidor. A continuación se muestra el código que implementara este comportamiento:

**ssh sensor**

```
ESCOM@linux:~$ ssh localhost
ESCOM@localhost's password:
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-42-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

113 packages can be updated.
0 updates are security updates.

Last login: Wed Dec  5 15:13:03 2018 from 127.0.0.1
$
```

Figura 3.15: Servidor SSH Ejecutando.

```
1 import spur
2 import re
3 import pyshark
4
5 from Sensors.FTP.sensor_ftp import *
6
7 def obtainBytesActivity(ssh_ip):
8     capture = pyshark.LiveCapture(interface='en0',
9                                   bpf_filter='ip and tcp port 22')
10    capture.sniff(timeout=1)
11    print(capture)
12    return getTotal(capture, ssh_ip)
13
14 def getTotal(capture, ssh_ip):
15     total_send = 0
16     total_recv = 0
17
18     try:
19         for packet in capture:
20             if packet.destination == ssh_ip:
21                 total_recv += int(packet.length)
22             elif packet.source == ssh_ip:
23                 total_send += int(packet.length)
24     except pyshark.capture.capture.TSharkCrashException as e:
25         print(e)
26     return total_send, total_recv
27
28
29 def doLogin():
30     try:
```

```
31         # Spur command shell execution
32         shell = spur.SshShell(
33             hostname="127.0.0.1",
34             username="root",
35             password="root",
36             port="22",
37             missing_host_key=spur.ssh.MissingHostKey.accept
38         )
39         return shell
40     except Exception as e:
41         print(e)
42         return -1
```

### main ssh

```
1 def sshSensor():
2     # We do login and obtain an instance of the shell
3     shell = doLogin()
4     # We run the command prompt from the shell
5     with shell:
6         result = shell.run(["sh",
7                             "-c",
8                             "netstat -apt | grep 'ESTABLISHED.*ssh '"])
9
10    # Then we parse the result and we process the text data
11    if result.return_code == 0:
12        lines = str(result.output, encoding="utf-8").split("\n")
13        for line in lines:
14            if (line != ""):
15                data = re.findall(r'\S+', line)
16
17                ssh_pid = int(str(data[6]).split("/")[0])
18                ssh_conn = " ".join(data[4:])
19                print("=====")
20                print("Process PID: " + str(ssh_pid))
21                print("Connection: " + str(ssh_conn))
22
23            # We do login and obtain an instance of the shell
24            shell = doLogin()
25
26            # We run the command prompt from the shell
27            with shell:
28                result = shell.run(["sh",
29                                    "-c",
30                                    "ps -o etime -p "
31                                    + str(ssh_pid)])
32
```

```

33         ssh_conn_time = " ".join(re.findall(r'\S+'
34         str(result.output, encoding="utf-8")))
35         print("Time: " + str(ssh_conn_time))
36
37         # Connect to host, default port
38         ftp = ftplib.FTP(FTP_HOST)
39         #Do login
40         ftp.login(FTP_USERNAME, FTP_PASSWORD)
41
42         print("Input/Output Traffic: " +
43         obtainBytesActivity("10.10.10.2"))
44         print("=====")
45
46         print("Total number of SSH connections: " + str(len(lines) - 1))
47
48     else:
49         str(result.stderr_output, encoding="utf-8")
50         # prints the stderr output code

```

3. Una vez que se ha implementado el sensor, se procede a realizar el sensado del servidor SSH, el sensor se conecta mediante una API que implementa el protocolo SSH al servidor y ejecuta comandos shell y hace sniffing del puerto 22 para obtener la siguiente información:

- Número de conexiones activas.
- Tráfico de datos enviados y recibidos.
- Tiempo de actividad de las conexiones.
- Usuario SSH conectado.

A continuación se muestra la captura de pantalla que refleja el comportamiento de este sensor, las pruebas se realizarán al servidor SSH con la IP: 10.10.10.2

## Pruebas

```

(venv) ESCOM@linux:~/Documents/Monitoring/Sensors/SSH$ python ssh_sensor.py
=====
Process PID: 9287
Connection: equipo34cm1@10.10.10.2:ssh ESTABLISHED 9287/ssh
Time: ELAPSED 06:46
Input/Output Traffic (bytes):[11266,90128]
=====
Total number of SSH connections: 1

```

Figura 3.16: Prueba Sensor SSH.



## 4 | Recopilar la configuración de los dispositivos de manera periódica.

### 4.1. Administración del archivo de configuración.

Para el desarrollo de este módulo de administración fue necesario el uso del protocolo de transferencia de archivos FTP, ya que es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP.

Para esto fue necesario configurar (ya se había configurado en la supervisión de servidor de archivos) FTP en el host que tiene la herramienta y solo habilitar el servicio en el router específico con el comando *service ftp*. Como este módulo emplea el protocolo FTP, programamos un pequeño script en Shell para poder extraer e importar los archivos de configuración de un router. A continuación se presenta el código, además de evidencias de su funcionamiento.

```
1  #!/bin/sh
2
3  clear
4
5  #-----Credenciales-----
6  USER=rcp
7  pass=rcp
8  #-----
9
10 echo "-----Administracion del archivo de configuracion-----"
11 echo "Lopez Ayala Eric Alejandro"
12 echo "Lopez Romero Joel"
13 echo
14 echo "Ingrese direccion ip"
15 read -p "-> " direc
16 echo "Direccion: $direc "
17 echo
18
19 while true
20 do
21
22  clear
```

```
23 echo "Direccion ip: $direc "
24 echo
25
26 echo "Seleccione alguna opcion"
27 echo "1. Extraer archivo de configuracion"
28 echo "2. Importar archivo de configuracion"
29 echo "3. Cambiar direccion ip"
30 echo "4. Salir"
31 read -p "-> " opcion
32 echo
33
34 case $opcion in
35     1 ) echo "Extraer archivo"
36         read -p "Guardar archivo del $direc como: " nombre
37         echo
38         ftp -inv $direc <<-EOF
39         user $USER $pass
40         get startup-config $nombre
41         bye
42         EOF
43         ;;
44     2 ) echo "Importar archivo"
45         read -p "Archivo a enviar a $direc: " nom
46         echo
47         ftp -inv $direc <<-EOF
48         user $USER $pass
49         put $nom startup-config
50         bye
51         EOF
52         ;;
53     3 ) echo "Ingrese direccion ip"
54         read -p "-> " direc
55         echo "Direccion: $direc "
56         echo
57         ;;
58     4 ) echo "Adios"
59         exit 2
60         ;;
61     * ) echo "Escoje alguna opcion valida"
62         ;;
63
64 esac
65
66 done
```



Figura 4.1: Topología

```
-----Administracion del archivo de configuracion-----  
Lopez Ayala Eric Alejandro  
Lopez Romero Joel  
  
Ingresa direccion ip  
-> 192.168.122.1  
Direccion: 192.168.122.1
```

Figura 4.2: Dirección ip startup-config

```
Direccion ip: 192.168.122.1  
  
Seleccione alguna opcion  
1. Extraer archivo de configuracion  
2. Importar archivo de configuracion  
3. Cambiar direccion ip  
4. Salir  
-> 1  
  
Extraer archivo  
Guardar archivo del 192.168.122.1 como: startup-configR1  
  
Connected to 192.168.122.1.  
220 (vsFTPD 3.0.3)  
331 Please specify the password.  
230 Login successful.  
Remote system type is UNIX.  
Using binary mode to transfer files.  
250 Directory successfully changed.  
local: startup-configR1 remote: startup-config  
200 PORT command successful. Consider using PASV.  
150 Opening BINARY mode data connection for startup-config (28 bytes).  
226 Transfer complete.  
28 bytes received in 0.00 secs (38.6757 kB/s)  
221 Goodbye.
```

Figura 4.3: Extracción startup-config

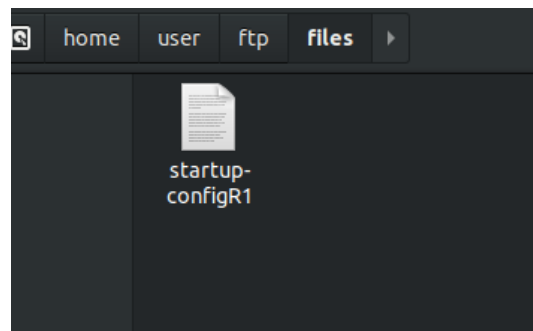


Figura 4.4: Extracción startup-config dir

```
Direccion ip: 192.168.122.1

Seleccione alguna opcion
1. Extraer archivo de configuracion
2. Importar archivo de configuracion
3. Cambiar direccion ip
4. Salir
-> 2

Importar archivo
Archivo a enviar a 192.168.122.1: startup-configR1

Connected to 192.168.122.1.
220 (vsFTPD 3.0.3)
331 Please specify the password.
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
250 Directory successfully changed.
local: startup-configR1 remote: startup-config
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
28 bytes sent in 0.00 secs (427.2461 kB/s)
221 Goodbye.
Direccion ip: 192.168.122.1
```

Figura 4.5: Importación startup-configR1

```

rcp(config)#dir
.:
total 12
-rw-r--r-- 1 rcp rcp 699 Dec 6 14:51 startup-config
drwxr-xr-x 2 rcp rcp 60 Dec 6 14:44 tftpboot
-rw----- 1 rcp rcp 7266 Dec 6 14:45 xmlstats.xml

./tftpboot:
total 4
-rw-r--r-- 1 rcp rcp 6 Dec 6 14:44 testfile

rcp(config)#dir
.:
total 16
-rw-r--r-- 1 rcp rcp 699 Dec 6 14:51 startup-config
-rw-r--r-- 1 rcp rcp 699 Dec 6 14:53 startup-configR1
drwxr-xr-x 2 rcp rcp 60 Dec 6 14:44 tftpboot
-rw----- 1 rcp rcp 7266 Dec 6 14:45 xmlstats.xml

./tftpboot:
total 4
-rw-r--r-- 1 rcp rcp 6 Dec 6 14:44 testfile

rcp(config)#_

```

Figura 4.6: Importación startup-configR1 dir

```

Direccion ip: 192.168.122.1

Seleccione alguna opcion
1. Extraer archivo de configuracion
2. Importar archivo de configuracion
3. Cambiar direccion ip
4. Salir
-> 1

Extraer archivo
Guardar archivo del 192.168.122.1 como: startup-configR2

Connected to 192.168.122.1.
220 (vsFTPD 3.0.3)
331 Please specify the password.
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
250 Directory successfully changed.
local: startup-configR2 remote: startup-config
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for startup-config (28 bytes).
226 Transfer complete.
28 bytes received in 0.00 secs (196.7176 kB/s)
221 Goodbye.

```

Figura 4.7: Extracción startup-config

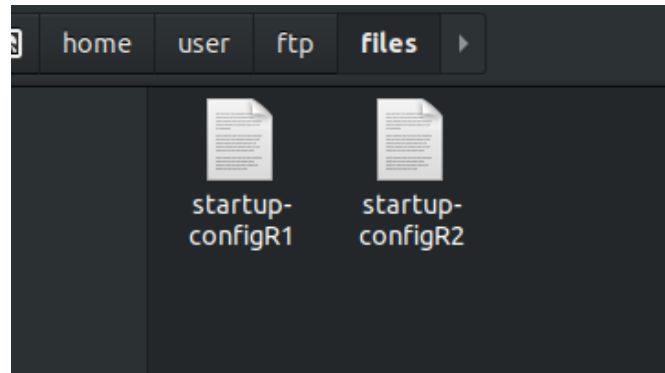


Figura 4.8: Extracción startup-config dir

Como se mostró, este módulo realiza la recolección de archivos de configuración de un router, pudiendo así administrar estos archivos, mediante la comparación, eliminación de configuraciones viejas, etc. Estos últimos nos fue difícil de implementar debido la organización de tiempos.

## 5 | Conclusiones

### 5.1. López Ayala Eric Alejandro

Para poder realizar una correcta configuración de los enrutadores para crear una topología, es importante conocer los comandos de RCP100 además de practicarlos, ya que al mezclar varios protocolos en un enrutador, se deben redistribuir sus tablas de enrutamiento de cada protocolo, para ello se emplearon los siguientes comandos: `redistribute rip`, `redistribute ospf`, `redistribute static`. Esta topología se logro gracias a GNS3.

El desarrollo e implementación de los sensores resultó ser una actividad desafiante durante el desarrollo del administrador de monitoreo, ya que dependiendo de las distintas variables y parámetros a sensar se requerian de distintas configuraciones aplicadas a los servidores, los cuales dependen de la topología de la red y su configuración definida. Los sensores son necesarios para que un administrador de red sea completo y requirió de aplicar todos los conocimientos adquiridos durante el curso.

El módulo de administración del archivo de configuración, lo realizamos en shell, debido a que solo eran peticiones simples de FTP (GET y PUT), como se presentó en el punto 4, esto fue fácil de implementar. Este pequeño modulo recolecta los archivos de configuración y los exporta a los enrutadores que pertenezcan a la red, por motivos de tiempo, nos fue imposible desarrollar la parte administrativa de este módulo, la cual consta en comparar los archivos de onfiguración, eliminación y copiar la configuración de inicio a la configuración en ejecución.

### 5.2. López Romero Joel

Tras el desarrollo de esta evaluación, fue importante el manejo de los comandos de configuración de RCP100, ya que este lo usamos en GNS3 para poder virtualizar routers y así poder implementar una topología mediante la configuración de estos. El manejo de estos comandos nos ayudaron demasiado, ya que al configurar algún router con 3 protocolos de enrutamiento, es necesario distribuir sus tablas de enrutamiento, para que estos se conozcan puedan definir una ruta para llegar a su destino, por ello fue necesario el empleo del comando *redistribute* tanto para RIP, OSPF y static. La implementación de una topología solo es la base, para probar el funcionamiento de nuestra herramienta desarrollada a lo largo de este parcial.

El desarrollo de los sensores resultó ser una actividad que complementó los conocimientos adquiridos durante el curso, ya que hicimos uso de ciertas herramientas que se ocuparon previamente y se complementarón con las nuevas desarrolladas durante este periodo. Así mismo nos ayudo a entender como se lleva la configuración de servidores que implementan los protocolos más comunes en una red.

El desarrollo del módulo de administración del archivo de configuración fue en shell, ya que gracias al uso del protocolo de transferencia de archivos FTP, solo fue necesario realizar peticiones con GET y PUT para extraer e importar los archivos de configuración. Este pequeño módulo quedo incompleto, ya que por falta de tiempo no nos fue posible desarrollar la comparación, eliminación de configuraciones viejas y copiar la configuración de inicio a la configuración en ejecución.

A lo largo de este semestre, aprendí demasiado en esta unidad de aprendizaje, ya que desarrollamos distintas herramientas las cuales nos ayudan a monitorizar y así poder administrar ciertas partes en una red. Desde monitorizar el comportamiento de un agente, realizar predicciones de acuerdo al comportamiento de un agente, detección de aberraciones, sensado de 5 servidores, hasta la administración de archivos de configuración de un enrutador. Para desarrollar estas herramientas usamos distintas librerías en Python, Django, Celery, Shell, protocolos como SNMP, RIP, OSPF. Fue difícil el desarrollo de estas herramientas, pero con la teoría vista en clase, investigaciones realizadas por nosotros, fue posible.



# Bibliografía

- [1] Cisco. Administración de la configuración informe oficial de mejores prácticas. [Online]. Available: [https://www.cisco.com/c/es\\_mx/support/docs/availability/high-availability/15111-configmgmt.html](https://www.cisco.com/c/es_mx/support/docs/availability/high-availability/15111-configmgmt.html)
- [2] M. Rouse. Gestión de la configuración de red, ncm. [Online]. Available: <https://searchdatacenter.techtarget.com/es/definicion/Gestion-de-la-configuracion-de-red-NCM>
- [3] R. 100. Documentation. [Online]. Available: <http://rcp100.sourceforge.net/cmdref.html>
- [4] G. Trubetskoy. Holt-winters forecasting for dummies - part ii. [Online]. Available: <https://grisha.org/blog/2016/02/16/triple-exponential-smoothing-forecasting-part-ii/>
- [5] ——. Holt-winters forecasting for dummies - part iii. [Online]. Available: <https://grisha.org/blog/2016/02/16/triple-exponential-smoothing-forecasting-part-iii/>