Network Management and Automation

Lab 4
Automation II

University of Colorado Boulder
Network Engineering Program

Professor Levi Perigo, Ph.D.

Summary

The aim of network automation is to minimize the effort required and decrease the chance of human error which is one of the leading causes of network downtime.

While using information from configuration files and deploying routine configurations onto multiple network devices is a step towards automation, this approach can be made more dynamic. Creating an interface that automates configuration from minimal user input simplifies the process, does not require the end user to know vendor-specific CLI commands, and ultimately reduces the possibility of misconfigurations.

Objectives

- Learn how to create user friendly web interface using Flask.
- Learn how to automate dynamic network configuration.

Objective 1:
Problem Statement:

In your previous lab, you were assigned the task of configuring iBGP between the routers in your data center. For iBGP to work, you will need underlay IGP (OSPF) connectivity within your AS. Your next task as a network engineer is to configure inter-area OSPF within the AS through a user-friendly web-based application.

For the given topology (Figure 1), automate the process of configuring inter-area OSPF between the routers through a front-end web application (Flask).

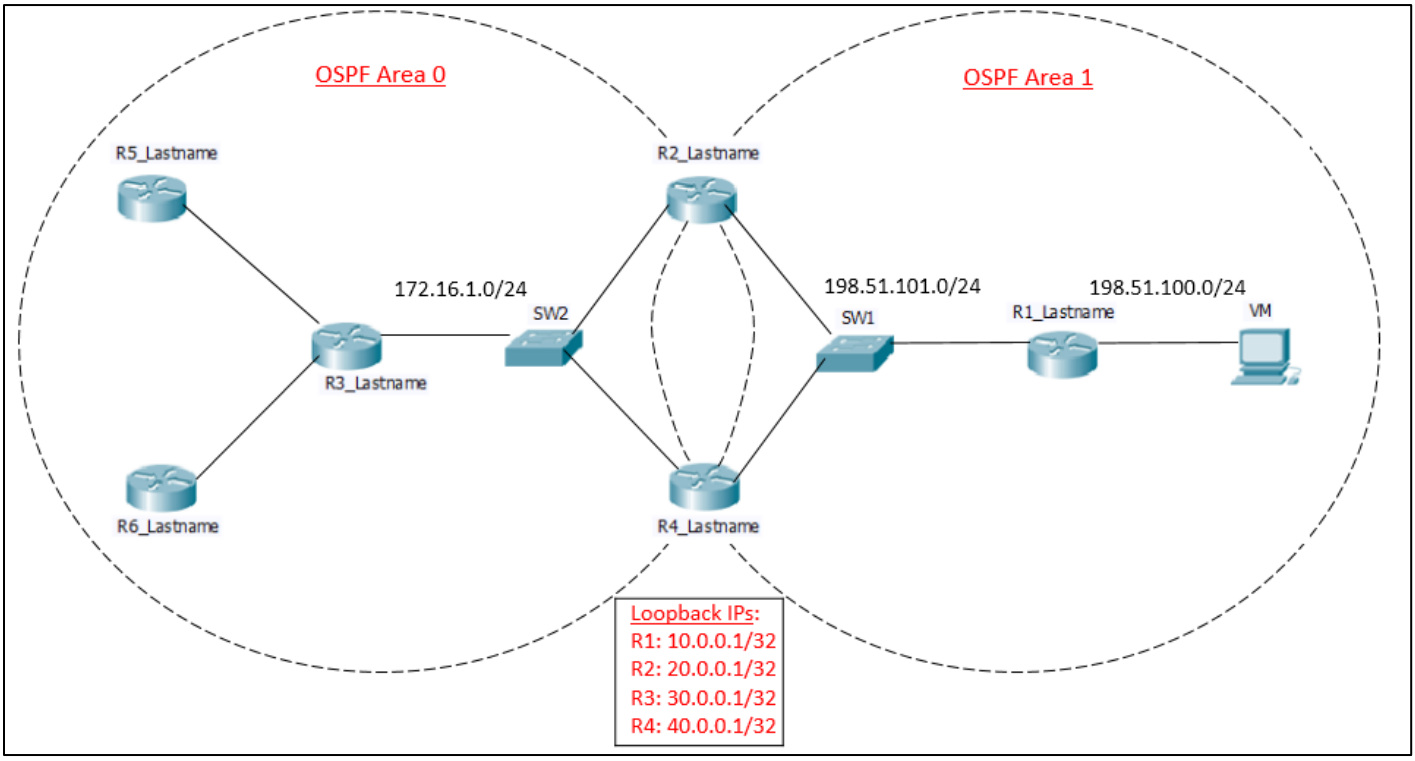


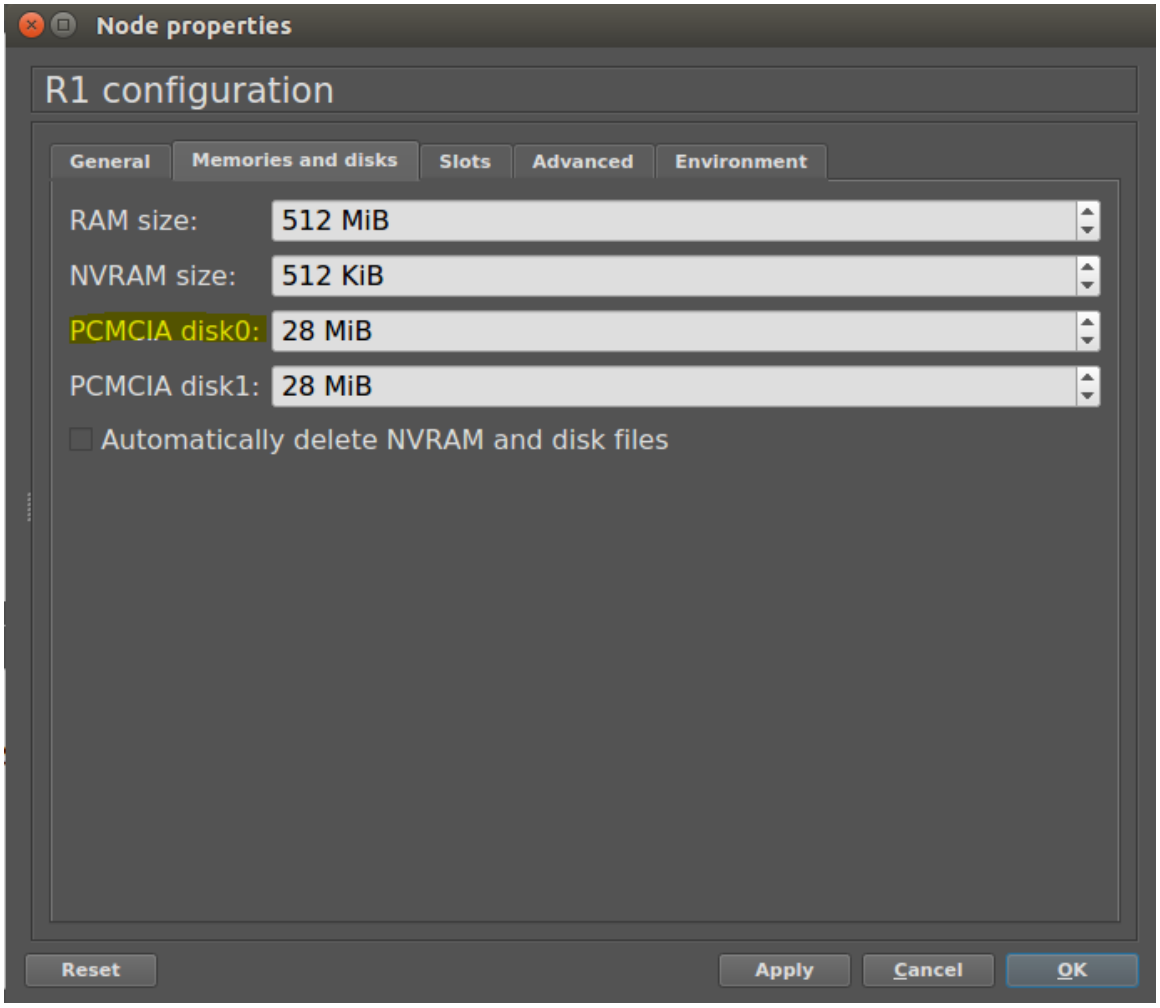
Figure 1

Guidelines:

1. Create the above topology in GNS3 on the NetMan VM. Manually configure the loopbacks and interface IPs on the routers. Follow the IP addressing scheme as mentioned in the above topology.
2. In the above topology you do NOT need to configure anything on R5 and R6.

3. Once the above topology has been created, ensure that you're able to SSH into all routers (R1, R2, R3 & R4) from your NetMan VM.
4. Ensure that the Flask application is up and running on your VM.

   Reference Link: http://flask.pocoo.org/docs/0.10/installation/
5. Use only Napalm for device configuration.
6. You can add webpages if needed as per your requirement.
7. For the routers in your topology, make sure you initialize a disk0 memory slot as shown below:



8. In GNS3, after configuring SSH; save the configs manually first ('write memory') and allow the NVRAM to be modified. Else the napalm library will not be able to save configs on your router.

Code Requirements:

- For this lab you will have to create modules (.py files) for each of the three Python files (getconfig.py, ospfconfig.py, and diffconfig.py) you write, which can be imported into a lab7main.py file.

Code Part – I "Web Front-end"

Lab6main.py -

- The homepage of the web application should present three options (Get config, OSPF config, and Diff config) to the user which navigate to the appropriate webpages.

Code Part – II "Get Device Configuration"

getconfig.py-

- When the user selects **Get config**, your script should fetch the current running configurations of routers R1, R2, R3, and R4, and save them locally as – RouterName_ISO8601timestamp.txt [example: R1_ 2018-03-10T16:56:41Z.txt]. The saved filenames should be displayed on the webpage. [**40 points**]

Code Part – III "Configure OSPF and Create Database"

ospfconfig.py –

Requirements

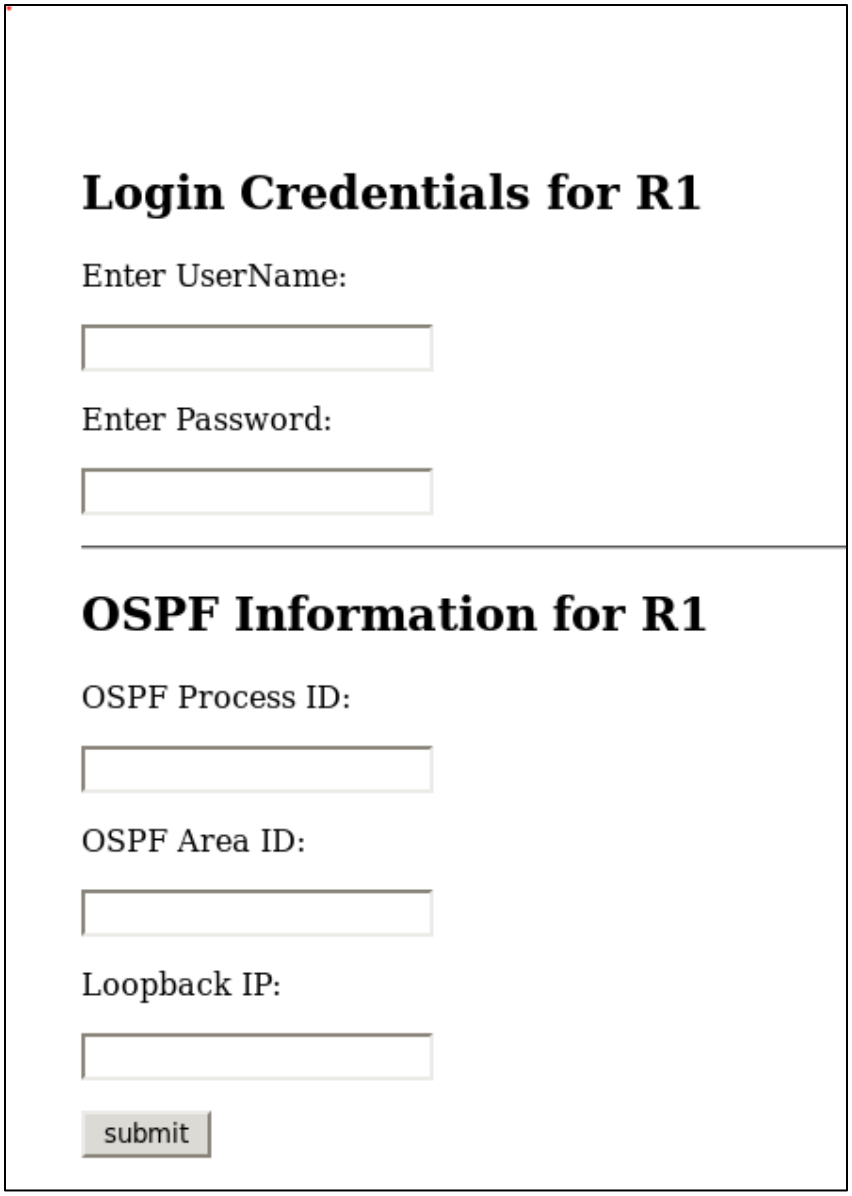
Part 1:

1. When "OSPF config" is selected, the user should be presented with a HTML page for each router (R1, R2, R3 & R4) to collect relevant data for SSH login and OSPF configuration from the user. **[2O POINTS]**
2. Routers R2 and R4 should be configured to do OSPF equal-cost load balancing. **[10 POINTS]**

Part 2 (Database creation):

1. Create a database using Sqlite3, MySQL etc. for storing the values obtained from the HTML page for each Router. Feel free to create your own schema.
3. Fire database queries to fetch the relevant values to configure inter-area OSPF between the routers and display the response. **[5 POINTS]**

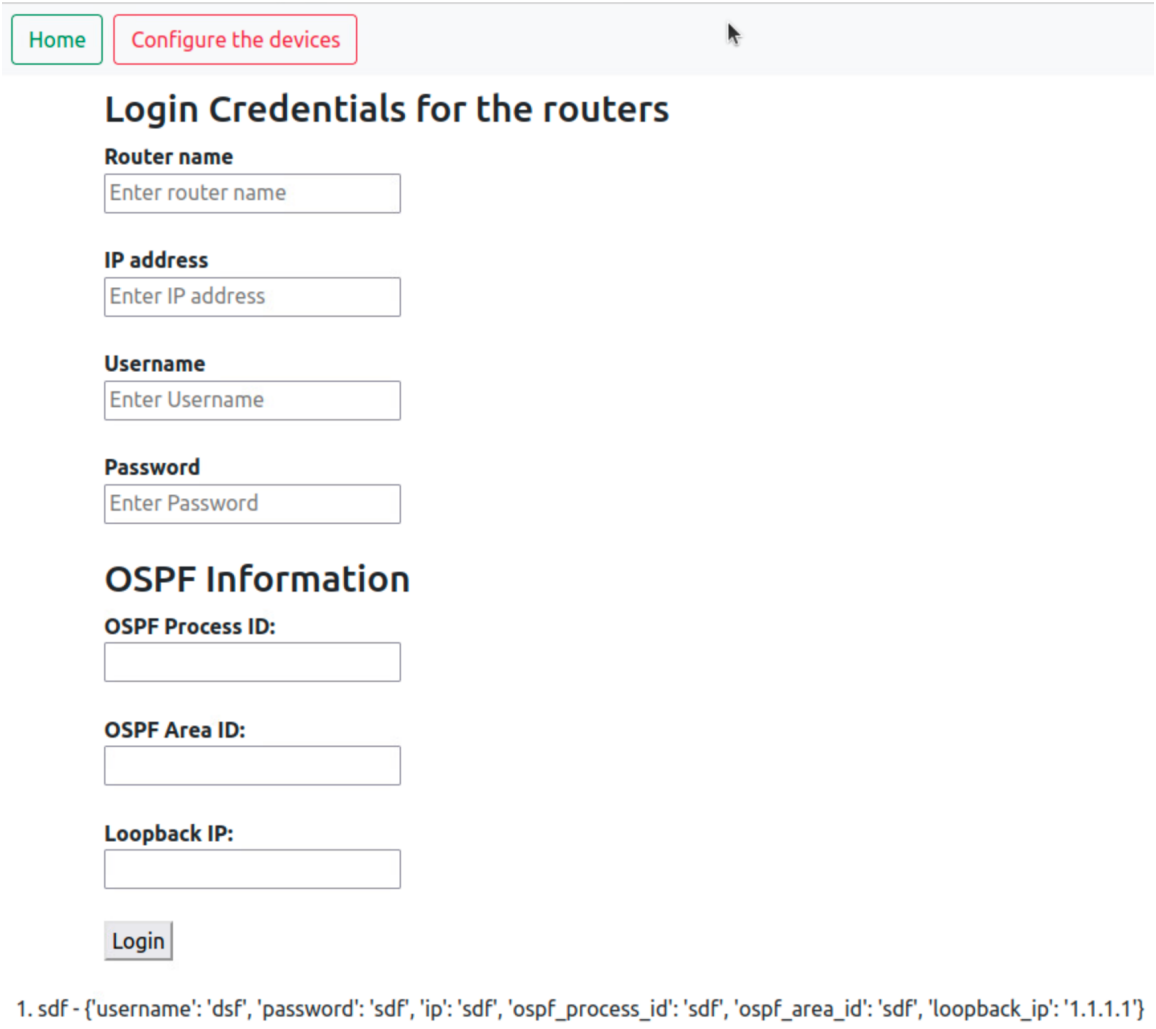
Below image shows the sample data required for Router R1. You are free to add fields, modify the layout, and create the HTML page of any style.



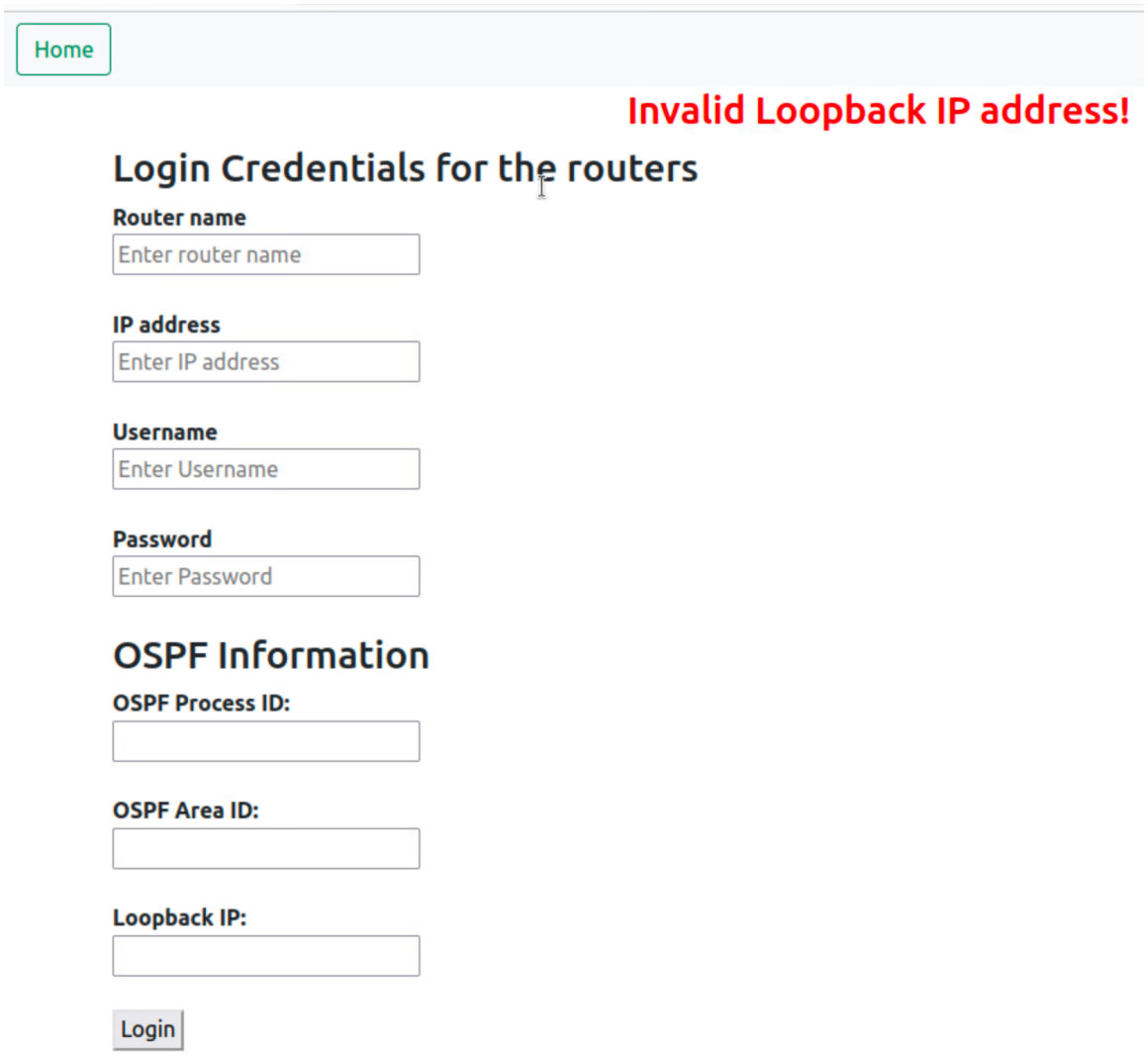
NOTE: For Routers R2 and R4, you may require some additional input from the user based on the above network topology.

Your **script** should fulfill the below requirements. Paste relevant supporting screenshots.

- Use Flask to fetch the values entered in the above HTML pages. **[30 Points]**



- After receiving IP addresses verify that the IP address are valid and are configured in your network. Print out the IP addresses and their Interfaces using PrettyTables. **[10 points]**



- Using the above data, configure inter-area OSPF for the Routers (R1, R2, R3 & R4) using Napalm. Make sure that you advertise the respective router loopbacks while configuring OSPF. **[30 Points]**

```
R1#show ip ospf neighbor

Neighbor ID     Pri   State           Dead Time   Address         Interface
20.0.0.1          1   FULL/BDR        00:00:31    198.51.101.2    FastEthernet1/1
40.0.0.1          1   FULL/DROTHER    00:00:30    198.51.101.4    FastEthernet1/1
```

```
R2#show ip ospf neighbor

Neighbor ID     Pri   State           Dead Time   Address         Interface
30.0.0.1          1   FULL/DROTHER    00:00:30    172.16.1.3      FastEthernet1/1
40.0.0.1          1   FULL/BDR        00:00:35    172.16.1.4      FastEthernet1/1
10.0.0.1          1   FULL/DR         00:00:36    198.51.101.1    FastEthernet1/0
40.0.0.1          1   FULL/DROTHER    00:00:34    198.51.101.4    FastEthernet1/0
```

```
R3#show ip ospf neighbor

Neighbor ID     Pri   State           Dead Time   Address         Interface
20.0.0.1          1   FULL/DR         00:00:33    172.16.1.2      FastEthernet1/0
40.0.0.1          1   FULL/BDR        00:00:37    172.16.1.4      FastEthernet1/0
```

```
R4#show ip ospf neighbor

Neighbor ID     Pri   State           Dead Time   Address         Interface
20.0.0.1          1   FULL/DR         00:00:35    172.16.1.2      FastEthernet1/1
30.0.0.1          1   FULL/DROTHER    00:00:33    172.16.1.3      FastEthernet1/1
10.0.0.1          1   FULL/DR         00:00:30    198.51.101.1    FastEthernet1/0
20.0.0.1          1   FULL/BDR        00:00:38    198.51.101.2    FastEthernet1/0
```

- Ping all the loopbacks from R1 and display the results on the webpage.

  [20 Points]

Code Part – IV "Difference of Device Configuration"

diffconfig.py -

- When the user selects Diff config, your script should fetch the current running configurations of routers R1, R2, R3 and R4, compare them with the previously saved router configuration files and print out only the difference in configuration for each router. [**40 points**]

Objective 2: Migration [20 points]

- In this scenario, the router R4 needs to be taken out of production to implement an upgrade with no impact on the traffic flow.
- For this objective, you will have to create another Python module **migration.py,** and add an option of **Migration** on the homepage (along with Get config, OSPF config and Diff config).
- During the entire execution of this module, a continuous ping should be running between R1 and R3 to check if there are any packet drops.
- Your script should first check if there is any traffic traversing currently on the link between R4 and SW2. After ensuring that there is no traffic on that link, proceed to shutdown the R4 interface connected to SW2, configure a banner motd on the router R4 '**Change made for migration in Lab 6**' and bring it back in the network.

- Your script should have appropriate print statements to indicate the execution steps.
- The goal is that traffic should flow via R2 for the duration when R4 is out of production, and the routers should start load balancing once R4 is back in the network. Once this process is complete, display the below message on the webpage-

  '**Migration completed successfully**'

Total Points _____ / 225