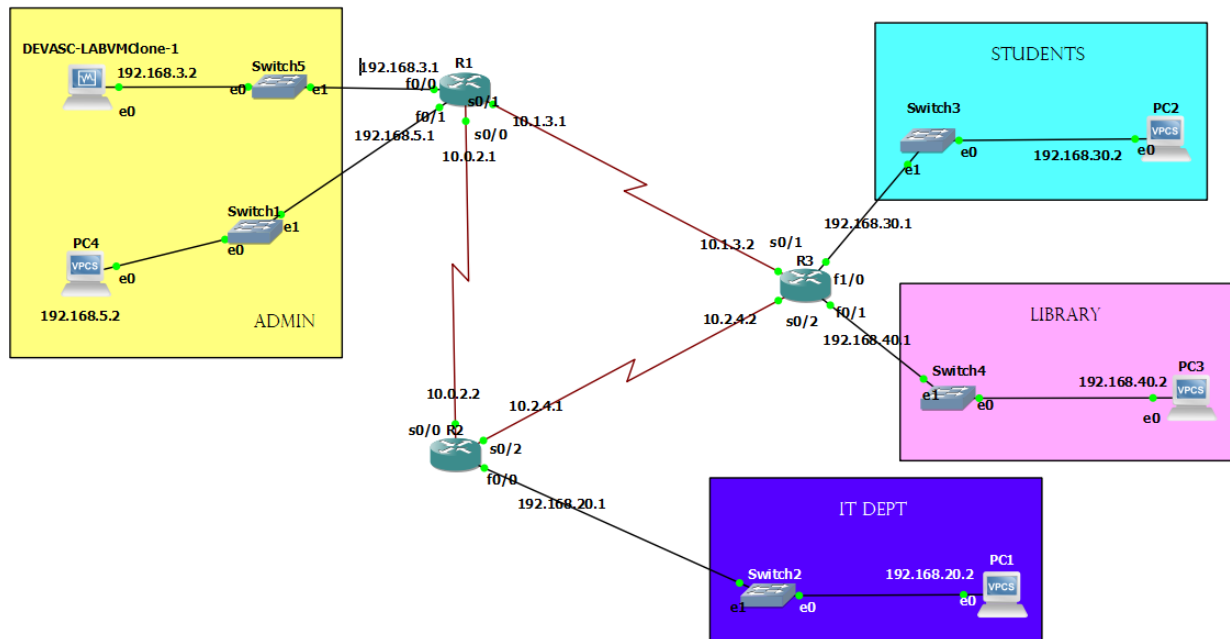


Final Case Study



Addressing Table

Device	Interface	IP Address	Subnet Mask
R1	S2/0	10.0.2.1	255.255.255.252
	S0/1	10.1.3.1	255.255.255.252
	F0/0	192.168.3.1	255.255.255.0
	F0/1	192.168.5.1	255.255.255.0
R2	S0/0	10.0.2.2	255.255.255.252
	S0/2	10.2.4.2	255.255.255.252
	F0/0	192.168.20.1	255.255.255.0
R3	S0/1	10.1.3.2	255.255.255.252
	S0/2	10.2.4.1	255.255.255.252
	F0/1	192.168.40.1	255.255.255.0
	F1/0	192.168.30.1	255.255.255.0
IT Dept	NIC	192.168.20.2	255.255.255.0
Students	NIC	192.168.30.2	255.255.255.0

Library	NIC	192.168.40.2	255.255.255.0
Admin1	NIC	192.168.3.2	255.255.255.0
Admin2	NIC	192.168.5.2	255.255.255.0

Objectives

Part 1: Launch the GNS3

Part 2: Cable the Network and Execute Basic Configuration

Part 3: Launch the DEVASC VM

Part 4: Configure Open Shortest Path First (OSPF)

Part 5: Configure Access Control Lists (ACL)

Part 6: Configure Authentication, Authorization and Accounting (AAA)

Part 7: YAML Codes Combination

Part 8: pyATS Testing

Part 9: Uploading to GitHub

Background / Scenario

In this activity, you will need to design a laboratory activity that discusses three different network topics, specifically AAA, ACL, OSPF and use it with ansible as application-deployment tool. To test the network, pyATS will be needed.

The devices in the topology are initially configured with:

- Console password: **cisco123**
- Enable password: **class**

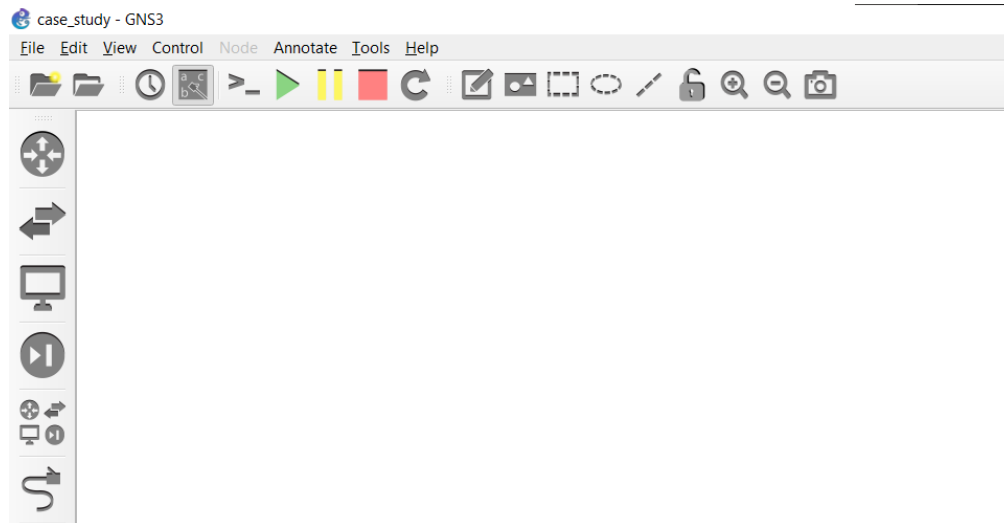
Required Resources

- 1 PC with operating system of your choice
- GNS3
- Virtual Box or VMWare
- DEVASC Virtual Machine

Instructions

Part 1: Launch the GNS3

Network Automation



Part 2: Cable the Network and Execute Basic Configuration

Step 1: Create the network as shown in the topology. Attach the devices as shown in the topology diagram and cable as necessary.

Step 2: Configure basic settings on R1, R2, and R3.

```
R1> en
R1# conf t
R1(config)# hostname R1
R1(config)# username cisco password cisco123
R1(config)# enable secret class
R1(config)# service password-encryption
R1(config)# banner motd "Unauthorized Access is Prohibited"
R1(config)# ip domain-name www.abc.com
R1(config)# crypto key gen rsa
```

```
R2> en
R2# conf t
R2(config)# hostname R1
R2(config)# username cisco password cisco123
R2(config)# enable secret class
R2(config)# service password-encryption
R2(config)# banner motd "Unauthorized Access is Prohibited"
R2(config)# ip domain-name www.abc.com
R2(config)# crypto key gen rsa
```

```
R3> en
R3# conf t
R3(config)# hostname R1
R3(config)# username cisco password cisco123
R3(config)# enable secret class
R3(config)# service password-encryption
R3(config)# banner motd "Unauthorized Access is Prohibited"
R3(config)# ip domain-name www.abc.com
R3(config)# crypto key gen rsa
```

Network Automation

Step 3: Configure IP address on each router.

```
R1(config)#int f0/0
R1(config-if)#ip add 192.168.30.1 255.255.255.0
R1(config-if)#no shut
R1(config-if)#
*Mar 1 00:05:48.171: %LINK-3-UPDOWN: Interface FastEthernet0/0, changed state to up
*Mar 1 00:05:49.171: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to up
R1(config-if)#int s0/0
R1(config-if)#ip add 10.0.2.1 255.255.255.252
R1(config-if)#no shut
R1(config-if)#
*Mar 1 00:11:04.119: %LINK-3-UPDOWN: Interface Serial0/0, changed state to up
R1(config-if)#
*Mar 1 00:11:05.123: %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/0, changed state to up
R1(config-if)#
*Mar 1 00:11:33.047: %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/0, changed state to down
R1(config-if)#int s0/1
R1(config-if)#ip add 10.1.3.1 255.255.255.252
R1(config-if)#no shut
R1(config-if)#
*Mar 1 00:12:46.543: %LINK-3-UPDOWN: Interface Serial0/1, changed state to up
R1(config-if)#
*Mar 1 00:12:47.547: %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/1, changed state to up
```

```
R2(config)#int s0/0
R2(config-if)#ip add 10.0.2.2 255.255.255.192
R2(config-if)#no shut
R2(config-if)#
*Mar 1 00:01:14.731: %LINK-3-UPDOWN: Interface Serial0/0, changed state to up
R2(config-if)#
*Mar 1 00:01:15.735: %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/0, changed state to up
R2(config-if)#int s0/2
R2(config-if)#ip add 10.2.4.2 255.255.255.192
R2(config-if)#no shut
R2(config-if)#
*Mar 1 00:01:39.831: %LINK-3-UPDOWN: Interface Serial0/2, changed state to up
R2(config-if)#
*Mar 1 00:01:40.835: %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/2, changed state to up
R2(config-if)#int f0/0
R2(config-if)#ip add 192.168.20.1 255.255.255.0
R2(config-if)#no shut
R2(config-if)#
*Mar 1 00:03:18.243: %LINK-3-UPDOWN: Interface FastEthernet0/0, changed state to up
*Mar 1 00:03:19.243: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to up
R2(config-if)#exit
R2(config)#ip route 0.0.0.0 0.0.0.0 s0/0
R2(config)#ip route 0.0.0.0 0.0.0.0 s0/2
R2(config)#ip route 0.0.0.0 0.0.0.0 f0/0
```

```
R3(config-if)#int s0/1
R3(config-if)#ip add 10.1.3.2 255.255.255.252
R3(config-if)#no shut
R3(config-if)#
*Mar 1 00:01:50.043: %LINK-3-UPDOWN: Interface Serial0/1, changed state to up
R3(config-if)#
*Mar 1 00:01:51.047: %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/1, changed state to up
R3(config-if)#int f1/0
R3(config-if)#ip add 192.168.30.1 255.255.255.0
R3(config-if)#no shut
R3(config-if)#
*Mar 1 00:02:14.519: %LINK-3-UPDOWN: Interface FastEthernet1/0, changed state to up
*Mar 1 00:02:15.519: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet1/0, changed state to up
R3(config-if)#int f0/1
R3(config-if)#ip add 192.168.40.1 255.255.255.0
R3(config-if)#no shut
R3(config-if)#
*Mar 1 00:02:33.115: %LINK-3-UPDOWN: Interface FastEthernet0/1, changed state to up
*Mar 1 00:02:34.115: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/1, changed state to up
R3(config-if)#exit
R3(config)#ip route 0.0.0.0 0.0.0.0 s0/1
R3(config)#ip route 0.0.0.0 0.0.0.0 s0/2
R3(config)#ip route 0.0.0.0 0.0.0.0 f1/0
R3(config)#ip route 0.0.0.0 0.0.0.0 f0/1
```

Network Automation

Step 4: Configure an IP address on each PC.

```
PC1> ip 192.168.20.2 255.255.255.0 192.168.20.1
Checking for duplicate address...
PC1 : 192.168.20.2 255.255.255.0 gateway 192.168.20.1
```

```
PC2> ip 192.168.8.2 255.255.255.0 192.168.8.1
Checking for duplicate address...
PC1 : 192.168.8.2 255.255.255.0 gateway 192.168.8.1
```

```
PC3> ip 192.168.7.2 255.255.255.0 192.168.7.1
Checking for duplicate address...
PC1 : 192.168.7.2 255.255.255.0 gateway 192.168.7.1
```

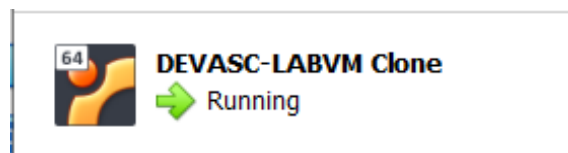
```
PC4> ip 192.168.5.2 255.255.255.0 192.168.5.1
Checking for duplicate address...
PC1 : 192.168.5.2 255.255.255.0 gateway 192.168.5.1
```

```
PC1> ip 192.168.40.2 255.255.255.0 192.168.40.1
Checking for duplicate address...
PC1 : 192.168.40.2 255.255.255.0 gateway 192.168.40.1
```

Part 3: Launch the DEVASC VM

Step 1: If you have not already completed the **Lab - Install the Virtual Machine Lab Environment**, do so now. If you have already completed that lab, launch the DEVASC VM now.

Step 2: In this part, you must clone your DEVASC VM to avoid IP changes.



Step 3: Ping to check connectivity in GNS3.

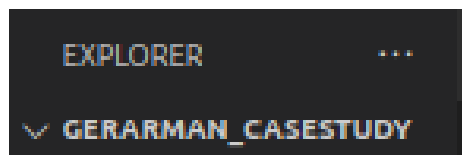
```
devasc@labvm:~$ ping 10.0.2.1
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.
64 bytes from 10.0.2.1: icmp_seq=1 ttl=255 time=7.75 ms
64 bytes from 10.0.2.1: icmp_seq=2 ttl=255 time=3.53 ms
64 bytes from 10.0.2.1: icmp_seq=3 ttl=255 time=2.56 ms
64 bytes from 10.0.2.1: icmp_seq=4 ttl=255 time=12.1 ms
64 bytes from 10.0.2.1: icmp_seq=5 ttl=255 time=2.29 ms
^Z
[2]+  Stopped                  ping 10.0.2.1
```

Step 4: Check SSH connection using DEVASC. This will allow you to check connectivity between each interface.

```
devasc@labvm:~$ ssh cisco@192.168.5.1
Warning: Permanently added '192.168.5.1' (RSA) to the list of known hosts.
Password:
Unauthorized Access is Prohibited
R1>logout
Connection to 192.168.5.1 closed.
```

***You must need to do this on other interfaces.

Step 5: Create a folder named **<lastname>_casestudy** inside the devnet-src/ansible folder.



Step 6: After creating the folder, add ansible.cfg file and copy the following code.

```
[defaults]
host_key_checking = False
```

Network Automation

Step 7: After creating the ansible.cfg, create a hosts file containing the following code.

```
[routers]
192.168.3.1
192.168.5.1
192.168.20.1
192.168.30.1
192.168.40.1
10.0.2.1
10.0.2.2
10.1.3.1
10.1.3.2
10.2.4.1
10.2.4.2

[routers:vars]
ansible_user=cisco
ansible_password=cisco123
ansible_network_os=ios
ansible_connection=network_cli]
```

Step 8: Once the ansible.cfg and hosts was created, check the connectivity using ansible.

```
devasc@labvm:/etc/ansible$ ansible routers -m ping
192.168.5.2 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
192.168.5.1 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
192.168.3.1 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
192.168.20.1 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
```

Part 4: Applying Open Shortest Path First (OSPF)

Before starting the application of OSPF in each router, you must have already finished **Part 3: Launch the DEVASC VM**.

Step 1: Use the following requirements to configure OSPF routing on all routers:

Network Automation

- Process ID 1
- Router ID for each router: R1 = 1.1.1.1; R2 = 2.2.2.2; R3 = 3.3.3.3
- Network address for each interface
- LAN interface set to passive

```
! ospf.yml > Cloud Code > [ ] tasks > { } ios_config > [ ] lines
1  ---
2
3  - hosts: routers
4    gather_facts: false
5    connection: network_cli
6    become_method: enable
7
8    tasks:
9      - name: Router 1 OSPF configuration
10        when: ansible_host == "10.0.2.1"
11        ios_config:
12          parents: router ospf 1
13          lines:
14            - router-id 1.1.1.1
15            - network 192.168.3.0 0.0.0.255 area 0
16            - network 192.168.5.0 0.0.0.255 area 0
17            - network 10.1.3.0 0.0.0.3 area 0
18            - network 10.0.2.0 0.0.0.3 area 0
19            - passive-interface FastEthernet0/0
20            - passive-interface FastEthernet0/1
21
22      - name: Router 2 OSPF configuration
23        when: ansible_host == "10.2.4.1"
24        ios_config:
25          parents: router ospf 1
26          lines:
27            - router-id 2.2.2.2
28            - network 10.0.2.0 0.0.0.3 area 0
29            - network 192.168.20.0 0.0.0.255 area 0
30            - network 10.2.4.0 0.0.0.3 area 0
31            - passive-interface FastEthernet0/0
32            - default-information originate
33
34      - name: Router 3 OSPF configuration
35        when: ansible_host == "10.2.4.2"
36        ios_config:
37          parents: router ospf 1
38          lines:
39            - router-id 3.3.3.3
40            - network 10.2.4.0 0.0.0.3 area 0
41            - network 10.1.3.0 0.0.0.3 area 0
42            - network 192.168.30.0 0.0.0.255 area 0
43            - network 192.168.40.0 0.0.0.255 area 0
44            - passive-interface FastEthernet0/1
45            - passive-interface FastEthernet1/0
```

***Additional: You can run the script and check for errors.

Part 5: Applying Access Control Lists (ACL)

Before starting the application of OSPF in each router, you must have already finished **Part 4: Applying Open Shortest Path First (OSPF)**.

Step 1: Create a file named `acl.yml`. This is where you will write your ACL configuration. After creating the yml file, copy the following codes.

```
! acl.yml > Cloud Code > [ ]tasks > {}ios_config
1  ---
2  - hosts: routers
3    gather_facts: false
4    connection: network_cli
5    become_method: enable
6
7    tasks:
8      - name: IT Department Access List
9        when: ansible_host == "192.168.20.1"
10       ios_config:
11         lines:
12           - deny 192.168.30.0 0.0.0.255
13           - permit any
14         parents: ip access-list standard Employee
15
16      - name: Interface F0/1 access group
17        when: ansible_host == "192.168.20.1"
18       ios_config:
19         lines:
20           - int f0/0
21           - ip access-group Employee out
```

***Additional: You can run the script and check for errors.

Part 6: Applying Authentication, Authorization and Accounting (AAA)

Step 1: Create a yml file named `aaa.yml` to configure local AAA authentication.

Network Automation

```
! aaa.yml > Cloud Code > [ ] tasks > [ ] name
1  ---
2
3  - hosts: routers
4    gather_facts: no
5    connection: network_cli
6    become_method: enable
7
8    tasks:
9      - name: Configure local AAA authentication for console access
10        ios_config:
11          commands:
12            - aaa new-model
13            - aaa authentication login default local
14
15      - name: Configure the line console to use the defined AAA authentication method
16        ios_config:
17          commands:
18            - login authentication default
19          parents: line console 0
20
21      - name: Configure a named list AAA authentication method for the vty lines
22        ios_config:
23          commands:
24            - aaa authentication login SSH-LOGIN local
25
26      - name: Configure the vty lines to use the defined AAA authentication method
27        ios_config:
28          commands:
29            - login authentication SSH-LOGIN
30          parents: line vty 0 4
```

***Additional: You can run the script and check for errors.

Part 7: YAML Codes

The given code is the compilation of three network topics: AAA, OSPF, and ACL.

```
---
- hosts: routers
  gather_facts: no
  connection: network_cli
  become_method: enable

  tasks:
    - name: Configure local AAA authentication for console access
      ios_config:
        commands:
          - aaa new-model
          - aaa authentication login default local

    - name: Configure the line console to use the defined AAA authentication method
      ios_config:
        commands:
          - login authentication default
        parents: line console 0

    - name: Configure a named list AAA authentication method for the vty lines
      ios_config:
        commands:
          - aaa authentication login SSH-LOGIN local
```

Network Automation

- name: Configure the vty lines to use the defined AAA authentication method
ios_config:
 - commands:
 - login authentication SSH-LOGIN
 - parents: line vty 0 4

- name: Router 1 OSPF configuration
when: ansible_host == "10.0.2.1"
ios_config:
 - parents: router ospf 1
 - lines:
 - router-id 1.1.1.1
 - network 192.168.3.0 0.0.0.255 area 0
 - network 192.168.5.0 0.0.0.255 area 0
 - network 10.1.3.0 0.0.0.3 area 0
 - network 10.0.2.0 0.0.0.3 area 0
 - passive-interface FastEthernet0/0
 - passive-interface FastEthernet0/1

- name: Router 2 OSPF configuration
when: ansible_host == "10.2.4.1"
ios_config:
 - parents: router ospf 1
 - lines:
 - router-id 2.2.2.2
 - network 10.0.2.0 0.0.0.3 area 0
 - network 192.168.20.0 0.0.0.255 area 0
 - network 10.2.4.0 0.0.0.3 area 0
 - passive-interface FastEthernet0/0
 - default-information originate

- name: Router 3 OSPF configuration
when: ansible_host == "10.2.4.2"
ios_config:
 - parents: router ospf 1
 - lines:
 - router-id 3.3.3.3
 - network 10.2.4.0 0.0.0.3 area 0
 - network 10.1.3.0 0.0.0.3 area 0
 - network 192.168.30.0 0.0.0.255 area 0
 - network 192.168.40.0 0.0.0.255 area 0
 - passive-interface FastEthernet0/1
 - passive-interface FastEthernet1/0

- name: IT Department Access List
when: ansible_host == "192.168.20.1"
ios_config:
 - lines:
 - deny 192.168.30.0 0.0.0.255
 - permit any
 - parents: ip access-list standard Students

- name: Interface F0/1 access group
when: ansible_host == "192.168.20.1"
ios_config:
 - lines:
 - int f0/0

Network Automation

```
- ip access-group Students out
```

Run the codes and check for errors. If there are no errors, the output should be like this:

PLAY	RECAP	*****						
10.0.2.1	:	ok=5	changed=1	unreachable=0	failed=0	skipped=4	rescued=0	ignored=0
10.0.2.2	:	ok=4	changed=1	unreachable=0	failed=0	skipped=5	rescued=0	ignored=0
10.1.3.1	:	ok=4	changed=1	unreachable=0	failed=0	skipped=5	rescued=0	ignored=0
10.1.3.2	:	ok=4	changed=1	unreachable=0	failed=0	skipped=5	rescued=0	ignored=0
10.2.4.1	:	ok=5	changed=1	unreachable=0	failed=0	skipped=4	rescued=0	ignored=0
10.2.4.2	:	ok=5	changed=1	unreachable=0	failed=0	skipped=4	rescued=0	ignored=0
192.168.20.1	:	ok=6	changed=3	unreachable=0	failed=0	skipped=3	rescued=0	ignored=0
192.168.3.1	:	ok=4	changed=1	unreachable=0	failed=0	skipped=5	rescued=0	ignored=0
192.168.30.1	:	ok=4	changed=1	unreachable=0	failed=0	skipped=5	rescued=0	ignored=0
192.168.40.1	:	ok=4	changed=1	unreachable=0	failed=0	skipped=5	rescued=0	ignored=0
192.168.5.1	:	ok=4	changed=1	unreachable=0	failed=0	skipped=5	rescued=0	ignored=0

Part 8: pyATS Testing

Step 1: To create your testbed YAML file, enter the command below. The --output parameter will create a testbed.yml file in a directory named yaml. The directory will be automatically created. The --encode-password parameter will encode the passwords in the YAML file. The parameter interactive means you will be asked a series of questions. Answer no to the first three questions. And then provide the following answers to create the testbed.yml file. Enter the code: **genie create testbed interactive --output yaml/testbed.yml --encode-password**. The output should be like this:

```
Start creating Testbed yaml file ..
Do all of the devices have the same username? [y/n] y
Common Username: cisco

Do all of the devices have the same default password? [y/n] y
Common Default Password (leave blank if you want to enter on demand):

Do all of the devices have the same enable password? [y/n] y
Common Enable Password (leave blank if you want to enter on demand):

Device hostname: R1
  IP (ip, or ip:port): 10.0.2.1
  Protocol (ssh, telnet, ...): ssh
  OS (iosxr, iosxe, ios, nxos, linux, ...): ios
More devices to add ? [y/n] y

Device hostname: R2
  IP (ip, or ip:port): 10.0.2.2
  Protocol (ssh, telnet, ...): ssh
  OS (iosxr, iosxe, ios, nxos, linux, ...): ios
More devices to add ? [y/n] y

Device hostname: R3
  IP (ip, or ip:port): 10.1.3.2
  Protocol (ssh, telnet, ...): ssh
  OS (iosxr, iosxe, ios, nxos, linux, ...): ios
More devices to add ? [y/n] n
Testbed file generated:
yaml/testbed.yml
```

Step 2: Using your testbed YAML file, invoke Genie to parse unstructured output from the **show ip interface brief** command into structured JSON. This command includes the IOS command to be parsed (**show ip interface brief**), the YAML testbed file (**testbed.yml**). The code is **genie parse "show ip interface brief" --testbed-file yaml/testbed.yml --devices > yaml/log.txt** and the output should be like this:

```
100%|██████████| 1/1 [00:01<00:00, 1.62s/it]
100%|██████████| 1/1 [00:00<00:00, 2.60it/s]
```

Network Automation

```
100%|███████████| 1/1 [00:00<00:00, 2.61it/s]
```

Step 3: To show the configuration of the ACL, we can use the given codes: **pyats learn acl --testbed-file yam1/testbed.yml --output yam1/acl/**.

```
devasc@labvm:~/labs/devnet-src/ansible/gerarman_casestudy$ pyats learn acl --testbed-file yaml/testbed.yml --output yaml/acl/
```

```
Learning '['acl']' on devices '['R1', 'R2', 'R3']'
```

```
100%|██████████████████████████████████████████████████████████████████████████| 678 B / 678 B
```

```
+=====+  
| Genie Learn Summary for device R1 |  
+=====+
```

```
| Connected to R1  
- Log: yaml/acl//connection_R1.txt  
-----+  
| Learnt feature 'acl'  
- Ops structure: yaml/acl//acl_ios_R1_ops.txt  
- Device Console: yaml/acl//acl_ios_R1_console.txt  
-----+  
+=====+
```

```
+=====+  
| Genie Learn Summary for device R2 |  
+=====+
```

```
| Connected to R2  
- Log: yaml/acl//connection_R2.txt  
-----+  
| Learnt feature 'acl'  
- Ops structure: yaml/acl//acl_ios_R2_ops.txt  
- Device Console: yaml/acl//acl_ios_R2_console.txt  
-----+  
+=====+
```

```
+=====+  
| Genie Learn Summary for device R3 |  
+=====+
```

```
| Connected to R3  
- Log: yaml/acl//connection_R3.txt  
-----+  
| Learnt feature 'acl'  
- Ops structure: yaml/acl//acl_ios_R3_ops.txt  
- Device Console: yaml/acl//acl_ios_R3_console.txt  
-----+  
+=====+
```

Step 4: For us to look for the information about the OSPF, use the codes given: **pyats learn ospf --testbedfile yaml/testbed.yml --output yaml/ospf/.**

```
devascs@labvm:~/labs/devnet-src/ansible/gerarman_casestudy$ pyats learn ospf --testbed-file yaml/testbed.yml --output yaml/ospf/
Learning '['ospf']' on devices '['R1', 'R2', 'R3']'
100%|#####|

=====+
| Genie Learn Summary for device R1                                     |
=====+
| Connected to R1                                                       |
|   - Log: yaml/ospf//connection_R1.txt                                |
|-----+
| Could not learn feature 'ospf'                                        |
|   - Exception:    yaml/ospf//ospf_ios_R1_exception.txt               |
|   - Ops structure: yaml/ospf//ospf_ios_R1_ops.txt                   |
|   - Device Console: yaml/ospf//ospf_ios_R1_console.txt              |
|-----+
|
=====+

=====+
| Genie Learn Summary for device R2                                     |
=====+
| Connected to R2                                                       |
|   - Log: yaml/ospf//connection_R2.txt                                |
|-----+
| Could not learn feature 'ospf'                                        |
|   - Exception:    yaml/ospf//ospf_ios_R2_exception.txt               |
|   - Ops structure: yaml/ospf//ospf_ios_R2_ops.txt                   |
|   - Device Console: yaml/ospf//ospf_ios_R2_console.txt              |
|-----+
|
=====+

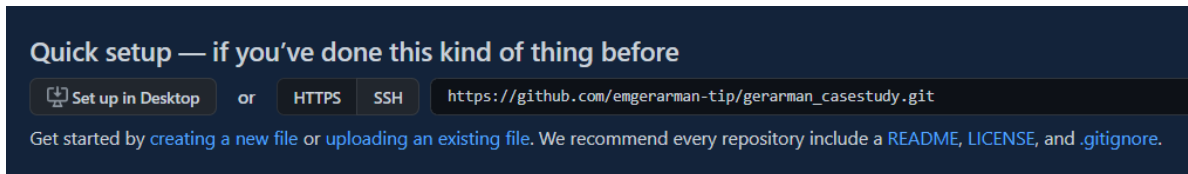
=====+
| Genie Learn Summary for device R3                                     |
=====+
| Connected to R3                                                       |
|   - Log: yaml/ospf//connection_R3.txt                                |
|-----+
|
```

Part 9: Uploading to GitHub

Network Automation

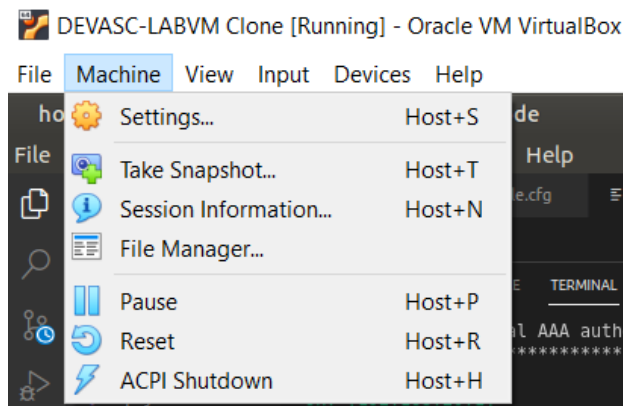
We can now upload the files to a GitHub Repository after validating and testing the network to see whether the configurations we applied were successful. To continue, make sure you have previously created a GitHub account.

Step 1: Create a GitHub repository having the same name with the directory you are working on. Get the link of the repository.

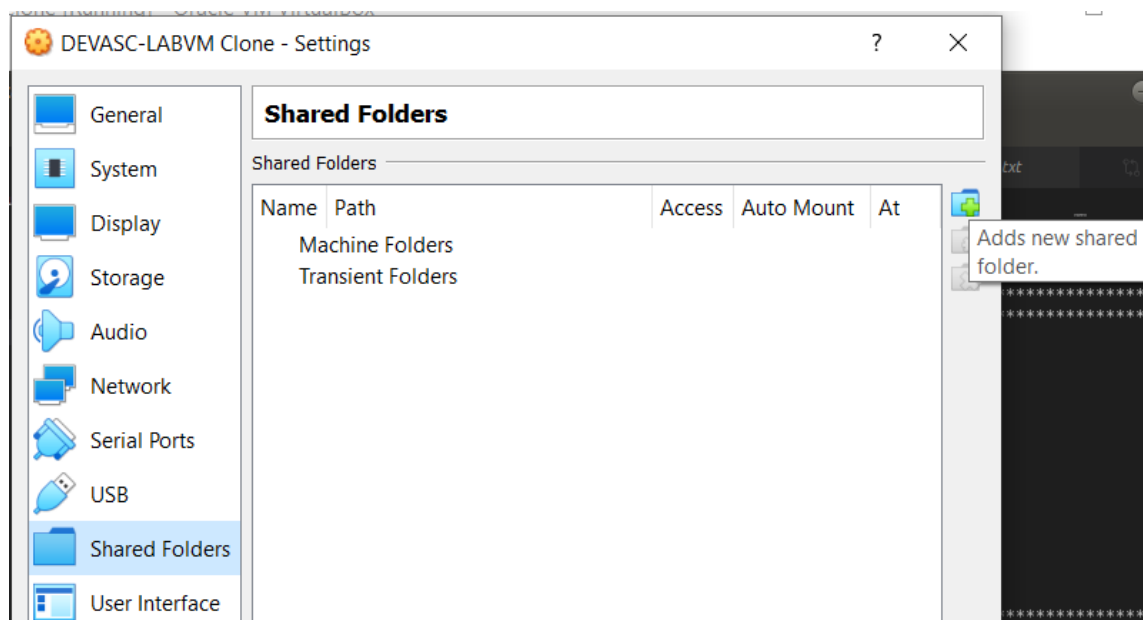


Step 2: Transfer your files from VM to your host device.

- Click the Machine button and look for Settings

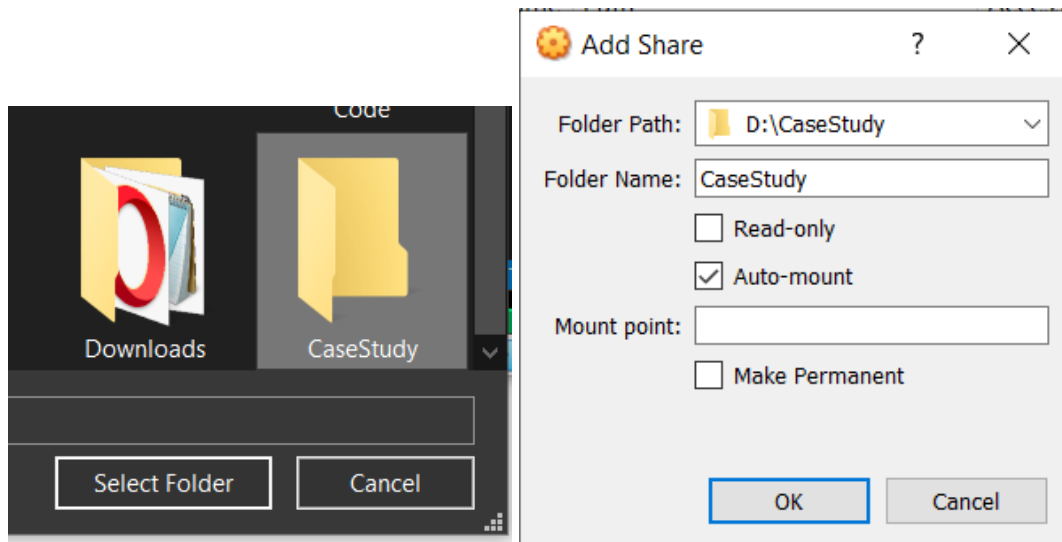


- Look for the Shared Folders tab and click the "Adds new shared folder".

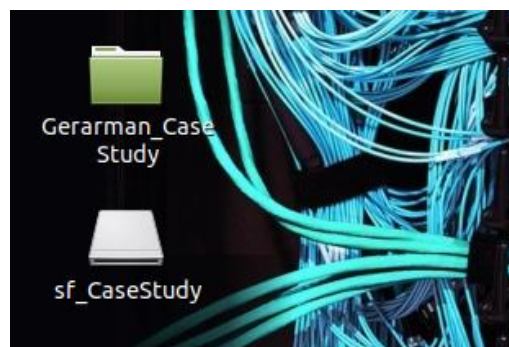


- Create a new folder in any path of your choice. The newly created folder should be selected and click "Auto-mount".

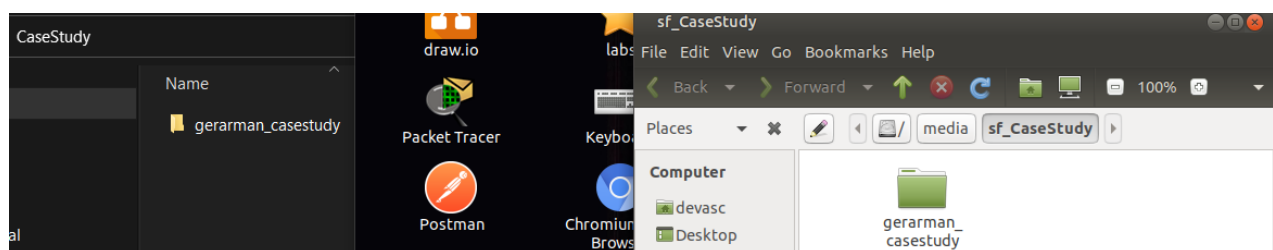
Network Automation



- After selecting your folder, you can see that there will be a new folder in your desktop in DEVASC VM.



- Copy your files in the new folder in your DEVASC VM. Check your host device for the same folder you've created. It should contain the same files you copied in your DEVASC.



Step 3: This time, you will be needing a Git Bash. If you haven't installed the Git Bash, follow this link <https://www.educative.io/edpresso/how-to-install-git-bash-in-windows> for the installation guide.

Network Automation

```
MINGW64:/d

Acer@LAPTOP-9SH9NJ05 MINGW64 /d
$
```

Step 4: Go to the shared folder and change your directory.

```
Acer@LAPTOP-9SH9NJ05 MINGW64 /d
$ cd CaseStudy
```

Step 5: The commands below is for you to connect to the repository you created.

```
Acer@LAPTOP-9SH9NJ05 MINGW64 /d/CaseStudy
$ git init
Initialized empty Git repository in D:/CaseStudy/.git/

Acer@LAPTOP-9SH9NJ05 MINGW64 /d/CaseStudy (master)
$ git remote add origin https://github.com/emgerarman-tip/gerarman_casestudy.git
```

Step 6: Verify that you have linked your local machine to the remote repository.

```
Acer@LAPTOP-9SH9NJ05 MINGW64 /d/CaseStudy (master)
$ git remote -v
origin https://github.com/emgerarman-tip/gerarman_casestudy.git (fetch)
origin https://github.com/emgerarman-tip/gerarman_casestudy.git (push)
```

Step 7: Add the changes in the working directory to the staging area. Also, it is needed to create a snapshot of the staged changes along a timeline of a Git projects history using git commit.

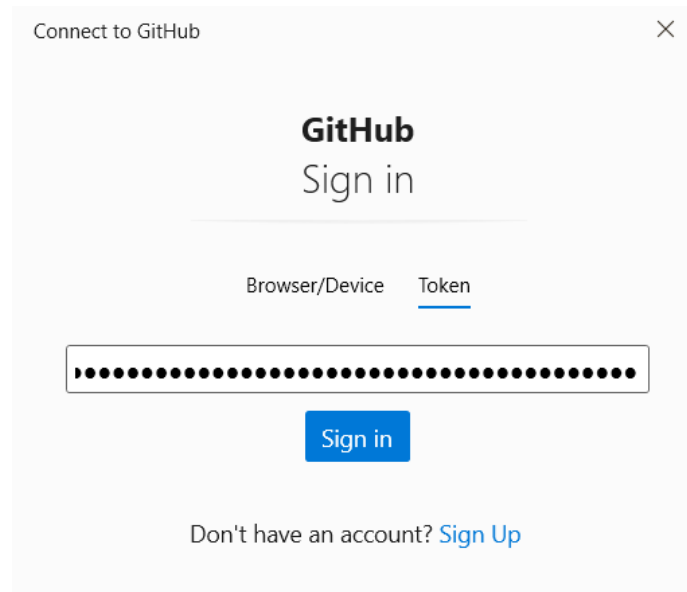
```
Acer@LAPTOP-9SH9NJ05 MINGW64 /d/CaseStudy (master)
$ git add .
warning: LF will be replaced by CRLF in gerarman_casestudy/aaa.yml.
The file will have its original line endings in your working directory
```

```
Acer@LAPTOP-9SH9NJ05 MINGW64 /d/CaseStudy (master)
$ git commit -m "Added files from Case Study"
[master (root-commit) 0e6b91a] Added files from Case Study
29 files changed, 2178 insertions(+)
```

Step 8: Push the changes from your current local branch to the remote server identified in your git config.

```
Acer@LAPTOP-9SH9NJ05 MINGW64 /d/CaseStudy (master)
$ git push -u origin master
fatal: Response status code does not indicate success: 401 (Unauthorized).
error: unable to read askpass response from 'D:/School/Git/mingw64/bin/git-askpass.exe'
Password for 'https://emgerarman-tip@github.com':
remote: Invalid username or password.
fatal: Authentication failed for 'https://github.com/emgerarman-tip/gerarman_casestudy.git/'
```

Step 9: The GitHub will require you to login for you to successfully push the changes in your repository.

A screenshot of the GitHub 'Connect to GitHub' dialog box. The dialog has a title bar with a close button (X). The main content area has the GitHub logo and the text 'Sign in'. Below this, there are two tabs: 'Browser/Device' and 'Token', with 'Token' being the active tab. Under the 'Token' tab, there is a text input field containing a series of dots, representing a masked token. Below the input field is a blue 'Sign in' button. At the bottom of the dialog, there is a link that says 'Don't have an account? Sign Up'.

```
Acer@LAPTOP-9SH9NJ05 MINGW64 /d/CaseStudy (master)
$ git push -u origin master
Enumerating objects: 30, done.
Counting objects: 100% (30/30), done.
Delta compression using up to 8 threads
Compressing objects: 100% (28/28), done.
Writing objects: 100% (30/30), 8.81 KiB | 1.26 MiB/s, done.
Total 30 (delta 13), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (13/13), done.
To https://github.com/emgerarman-tip/gerarman_casestudy.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

Conclusion

In the end of this activity, I was able to design a laboratory activity that discusses three different network topics, specifically Open Shortest Path First (OSPF), Access Control Lists (ACL), and Authentication, Authorization and Accounting (AAA) and use it with ansible as application-deployment tool. To test the network, pyATS was used. I can say that learning these topics again is challenging yet very exciting to do.