Lopez, Joshua Albert T. Engr. Alonica R. Villanueva

CPE41S3 January 17, 2022

Design of Laboratory Activity

**Network Programmability and Testing**

Final Case Study | Network Automation and Programmability

Objectives:

* Configure CSR1000v routers using python scripts with RESTCONF
* Discover and use the URI endpoints for configuring routers
* Implement the URI endpoints in the python scripts
* Perform requests with the python scripts through RESTCONF

*Note:*

*The activity is not a mere simulation of the network using GNS3, instead it is the actual connection utilized in the activity. These are through one locally connected CSR1000v router, and two sandboxes provided in DevNet Sandbox. This is because the device (laptop) being used in performing the case study (activity) has limited processing power, thus, even with expanded memory, it was not able to run the CSR1000v through GNS3. Different combinations of configurations within GNS3 have been done, but still did not work. Thus, as a workaround, the case study (activity) is done purely with VirtualBox.*

Resources:

* Oracle VM VirtualBox Manager
* DEVASC-LABVM virtual machine
* CSR1000v virtual machine
* Cisco DevNet Account
* Internet Connection

Procedures:

**Part 1: Network Programmability with RESTCONF**

1. **Examine how the devices will be accessed, and interconnect to one another. This Step primarily focuses on verifying CSR1000v sandbox and CSR1000v local connectivity.**
   1. The given topology represents the network that the activity uses. It can be observed that there are two routers connected through the cloud. These are the CSR1000v routers deployed through the CISCO DevNet sandboxes. These allow for free modification of configuration and interfaces, as if it were your own. One CSR1000v is deployed locally and is connected to the DEVASC-LABVM virtual machine. There is no need to rebuild the topology as it will be the actual network in the activity. Below this are some of the available links for the CSR1000v CISCO DevNet sandboxes that do not require reservation.

Diagram

Description automatically generated

Links:

IOS XE on CSR Recommended Code AlwaysOn

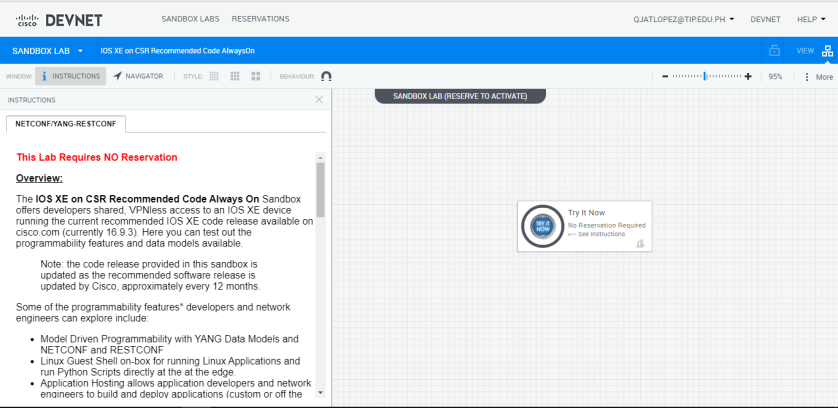
<https://devnetsandbox.cisco.com/RM/Diagram/Index/27d9747a-db48-4565-8d44-df318fce37ad?diagramType=Topology>

IOS XE on CSR Latest Code AlwaysOn1

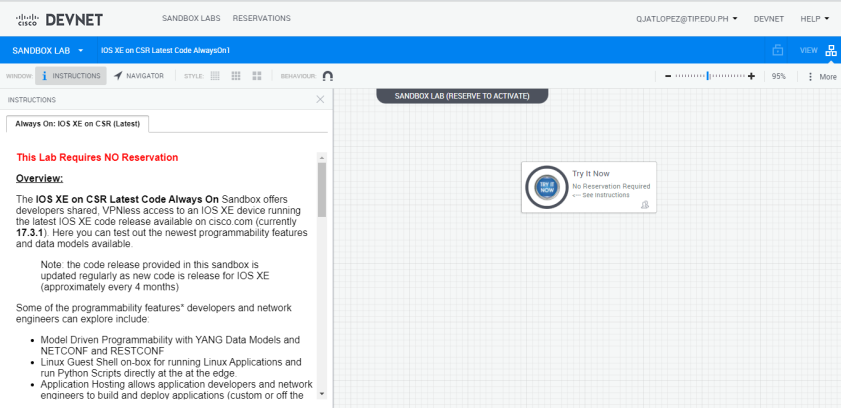
<https://devnetsandbox.cisco.com/RM/Diagram/Index/7b4d4209-a17c-4bc3-9b38-f15184e53a94?diagramType=Topology>

Note: To access, simply login to the CISCO DevNet account, or other accounts affiliated with Cisco such as Google account, and GitHub account. After logging in, the following screens should show up:

IOS XE on CSR Recommended Code AlwaysOn

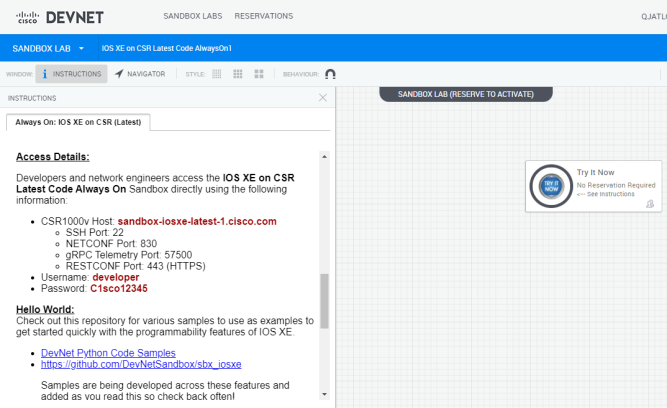


IOS XE on CSR Latest Code AlwaysOn1

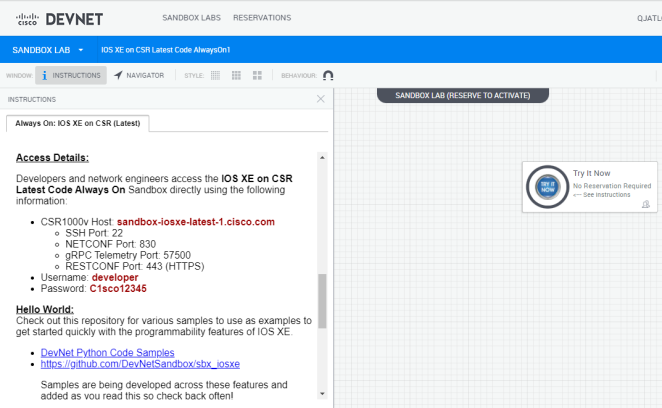


* 1. Check the access credentials for each sandbox by scrolling down to the middle part of each sandbox. Each sandbox has their own section called “Access Details” which shows the CSR1000v Host, Username, Password, and Ports.

IOS XE on CSR Recommended Code AlwaysOn

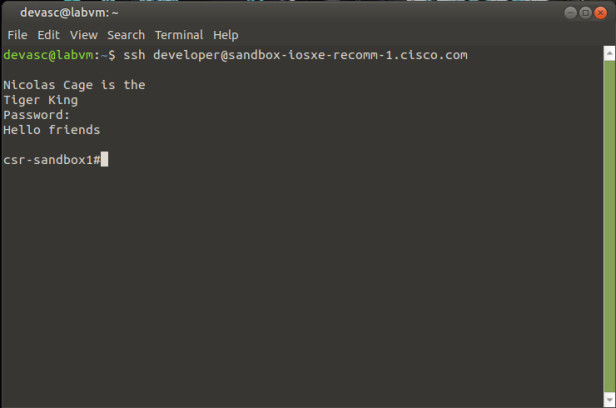


IOS XE on CSR Latest Code AlwaysOn1

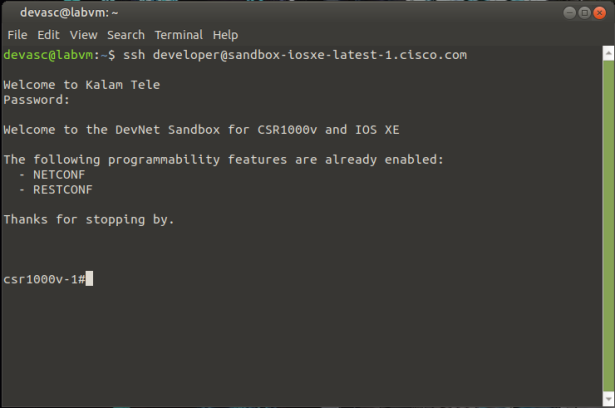


* 1. Verify interconnectivity among the CSR1000v virtual machines from the sandboxes, and the DEVASC LABVM virtual machine. Open a terminal in your DEVASC LABVM virtual machine. Use the Access Details found on each sandbox to perform SSH on each.

IOS XE on CSR Recommended Code AlwaysOn

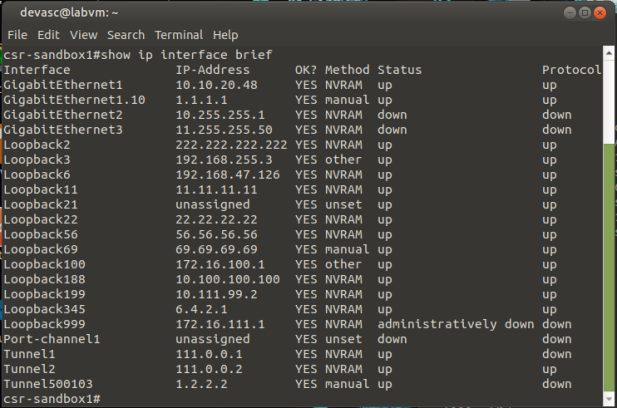


IOS XE on CSR Latest Code AlwaysOn1

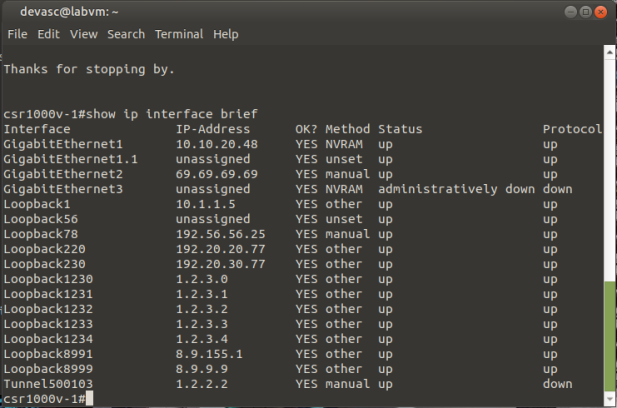


* 1. Perform router commands such as the “show ip interface brief” to further verify access to each sandbox.

IOS XE on CSR Recommended Code AlwaysOn

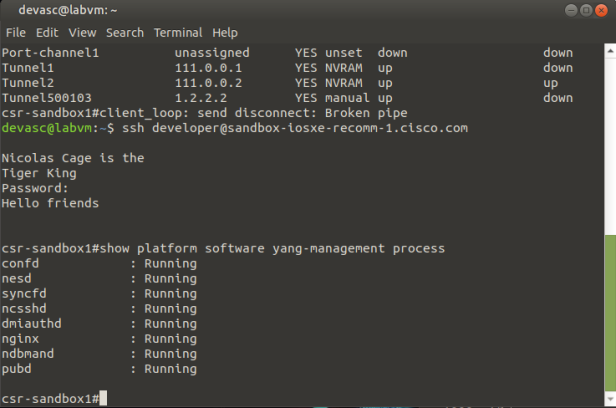


IOS XE on CSR Latest Code AlwaysOn1

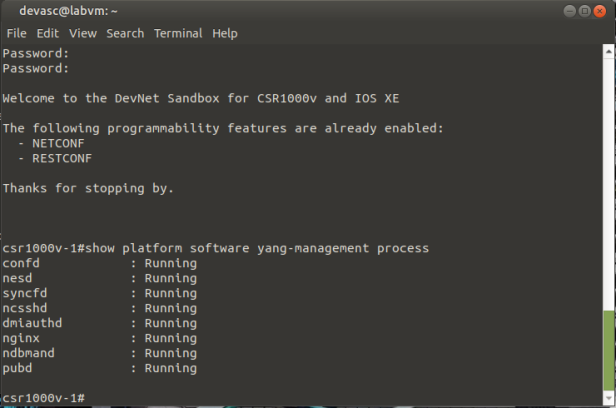


* 1. As a last stage in preparation for the RESTCONF for the sandboxes, recall and verify if each sandbox are running the RESTCONF daemons (Reference from 8.3.7 Lab Use RESTCONF to Access an IOS XE Device). Do this by entering the command “show platform software yang-management process”.

IOS XE on CSR Recommended Code AlwaysOn

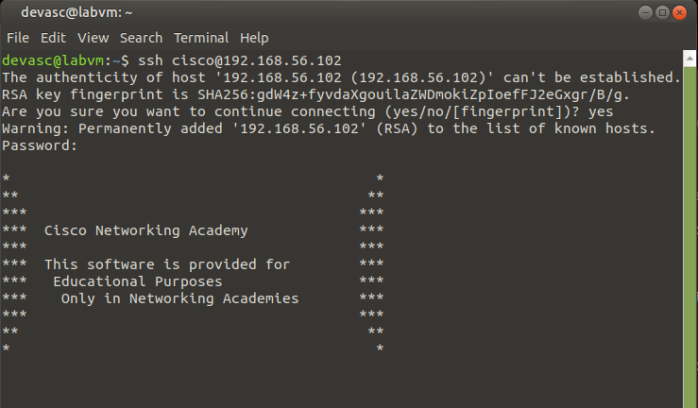


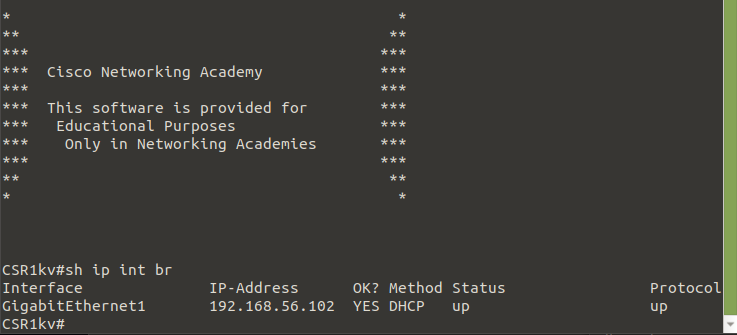
IOS XE on CSR Latest Code AlwaysOn1



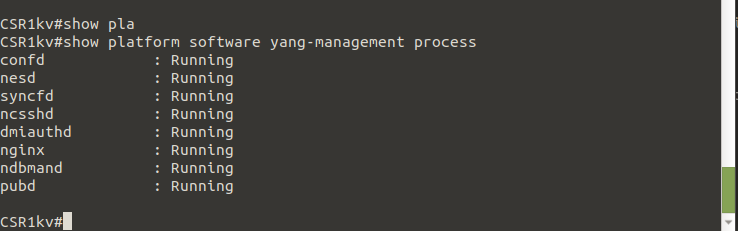
*Note: It is now verified that the DEVASC LABVM can connect with the CSR1000v Cisco DevNet sandboxes, and that each supports RESTCONF.*

* 1. The next step shows the same verification, this time with the locally connected CSR1000v. This should work since the previous lab activities require the interconnectivity between the DEVASC LABVM and the CSR1000v. Enter the command to ssh with the local CSR1000v then enter “cisco123!” as password. Then initialize a command to verify interfaces such as “show ip interface brief” as well.





* 1. Afterwards, also perform the command to verify if each the router is running the RESTCONF daemons with the command “show platform software yang-management process”.



*Note: All router connectivity and RESTCONF support are now verified. It is now time to prepare for verification of endpoints for configuring network topics for each of the routers through Postman, and then the actual RESTCONF.*

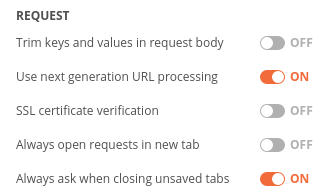
1. **Prepare Postman to find Network Topic (OSPF. IP Routing, and ACL) URI endpoints for RESTCONF API.**

Each network interface, configuration, and setup have their own URI endpoints. There are certain ways to check for endpoints. This can include reading documentations and by trial and error in traversing through the directory trees with Postman. In this part, Postman will be used to find the directory trees for each of the network topics with the help of documentations and forums. To demonstrate individual network configuration, one network topic will be configured on one router.

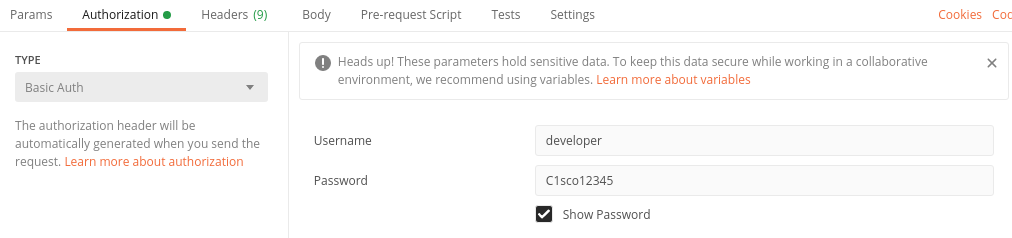
RESTCONF makes use of the YANG data model. It is a module that contain top-level hierarchy of nodes. Pretty much like a directory which can be displayed as a tree. It uses containers so that related nodes are grouped. These are then represented by the so-called leaf, with associated types. These will serve as a basis for the URI endpoint, which can be used for RESTCONF API requests. For more information about YANG Data Model, visit <https://www.netacad.com/sites/default/files/images/careers/Webinars/DevNet/devnet_session_7_networkprogrammability_yang_netconf_restconf.pdf>

*Note: The following sub-steps are already performed in previous lab activities; thus, it should work. These steps are included as a recap to ensure that the lab will perform correctly.*

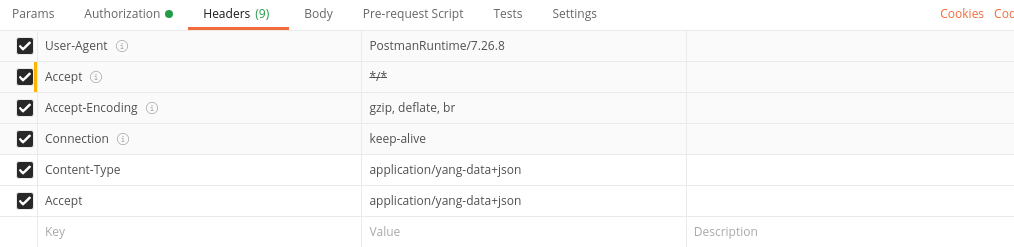
* 1. First, prepare Postman by disabling SSL certificate verification through the File > settings.



* 1. Then with Basic Auth selected under Authorization, enter the corresponding credentials of CSR1000v on both sandbox and locally. For the sandboxes use - **Username: developer; Password: C1sco12345**. Then for CSR1000v local use – **Username: cisco; Password: cisco123!**



* 1. Then, make sure to include these in the headers tab, most importantly the “Accept” to view the results as JSON.



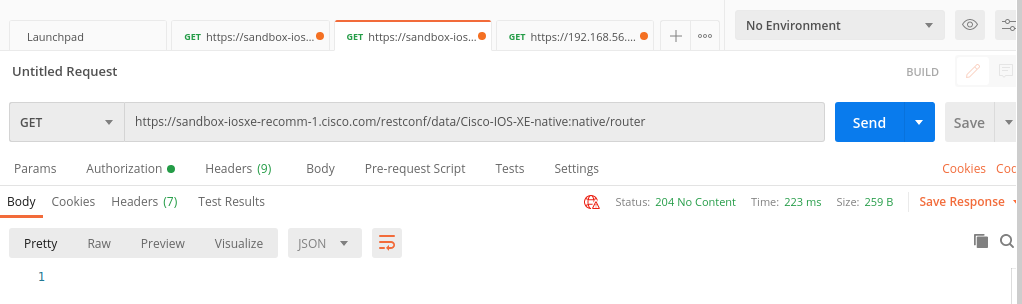
1. **NETWORK TOPIC 1: Configuring OSPF with the URI endpoint**
   1. Verify URI endpoint for OSPF, check the contents with Postman.

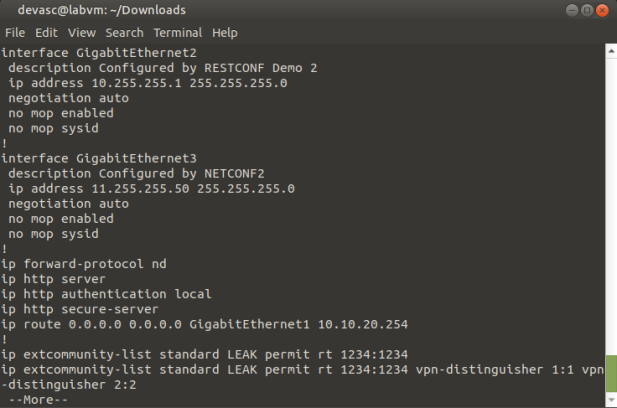
As mentioned, finding the URI endpoint can be done through trial and error, and with the help of documentations, and forums. Through this, a starting point for which part of the YANG data model to look at with ease is given. In this case, the OSPF URI endpoint will be examined. Here, the base URI starts with the Host address, under which contains the restconf module, that also contains the data which handles all nodes pertaining to the data module state, and other useful information. This then also contains the node for the Cisco-IOS-XE-native which contains all information regarding the Cisco-IOS-XE-native router. To get to the objective, the URI endpoint for the OSPF is discovered to be under the Cisco-IOS-XE-native:native/router. More information can be found at the sample program from <https://gist.github.com/e3prom/a35df51b6154d4862dd389e8a58a771b>

Thus, the URI for router OSPF is determined by:

"https://<HOST-ADDRESS> /restconf/data/Cisco-IOS-XE-native:native/router"

This can be verified by entering the address of the corresponding routers through Postman. As mentioned earlier, to demonstrate individual network configuration, any one router can be used for this specific network topic, which is OSPF. In this case, OSPF will be configured on IOS XE on CSR Recommended Code AlwaysOn, from Cisco DevNet sandbox.

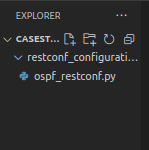




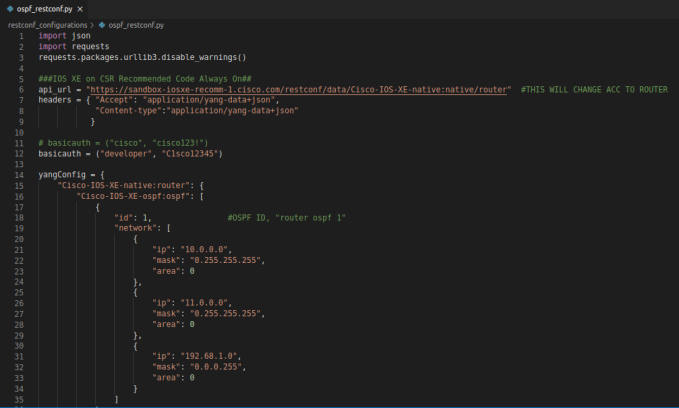
Notice that the router did not return a JSON data, and the Status: 204 No Content is present. This is normal, it indicates that there is a response which is just empty since there is no router OSPF configured on the router. Issuing the show run command on the router also shows that there is no OSPF configuration present which is usually found at the end of the interfaces information.

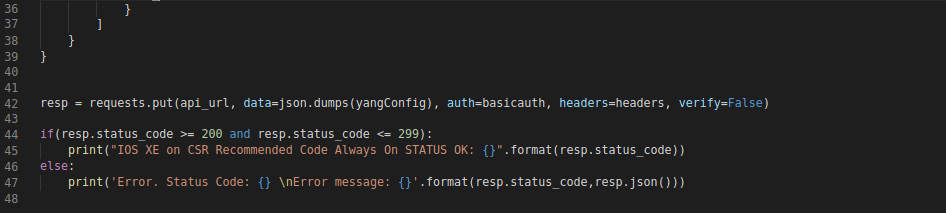
* 1. Use Python script with RESTCONF to configure OSPF.

Open VSCode. Create a folder named restconf\_configurations. Inside it, create a file called ospf\_restconf.py.



This will contain the script to put request the router OSPF configuration in the router.

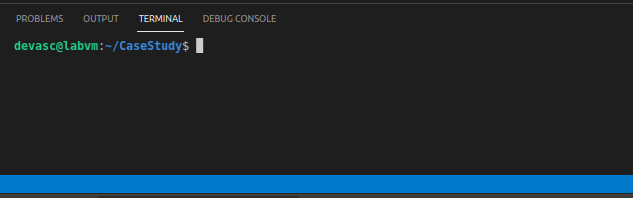




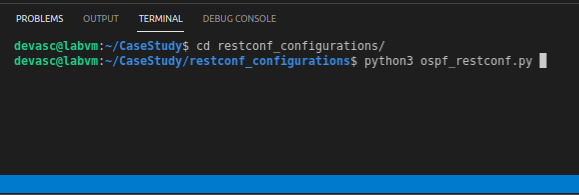
*Note: The code for all network topics have the same structure. Imports of the json, and requests modules are done. Then defined by api\_url, is the URI endpoint for where the configuration for the network topic is found. Headers are included to specify that json data is sent and accepted. Basicauth includes the username and password for the router. yangConfig is the model in json format that would be passed with the request. An if else statement then shows the response status, whether good or there is an error.*

* 1. Run the Python script and verify if it works, by checking through Postman

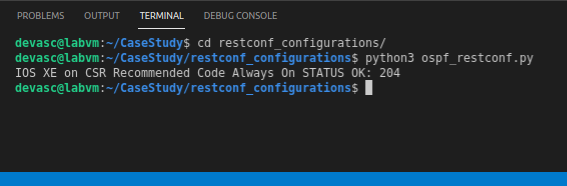
Open a New Terminal in VSCode.



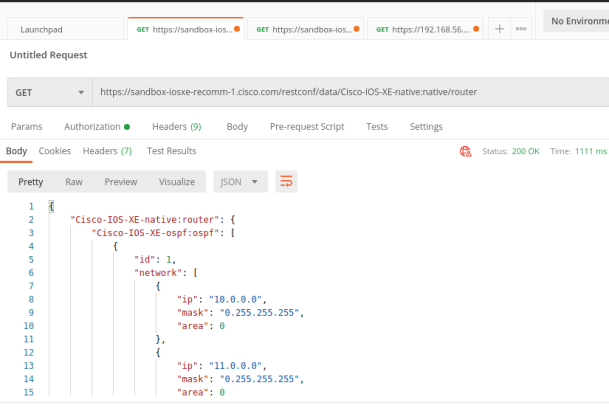
Navigate to the restconf\_configurations folder. Enter “Python3 ospf\_restconf.py”.

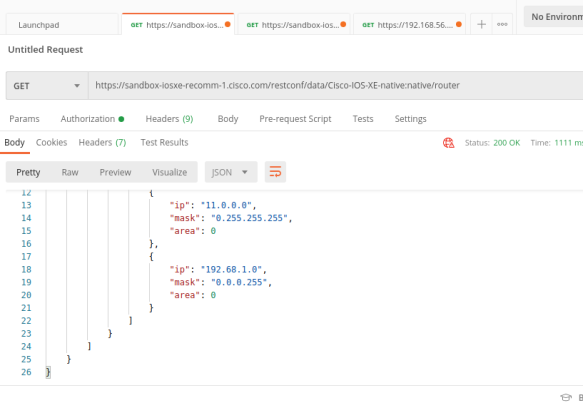


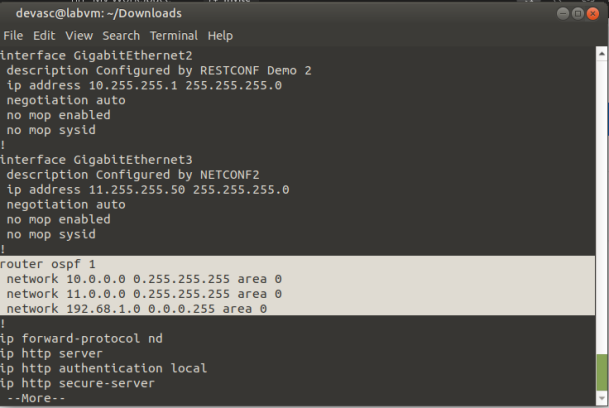
Then wait for a few seconds, the response should be similar to what is inside the script, in this case “IOS XE on CSR Recommended Code Always On STATUS OK: 204”



To see the results, go back to Postman GET request, then click SEND again. This time, it will show the configurations that have been made with the script.







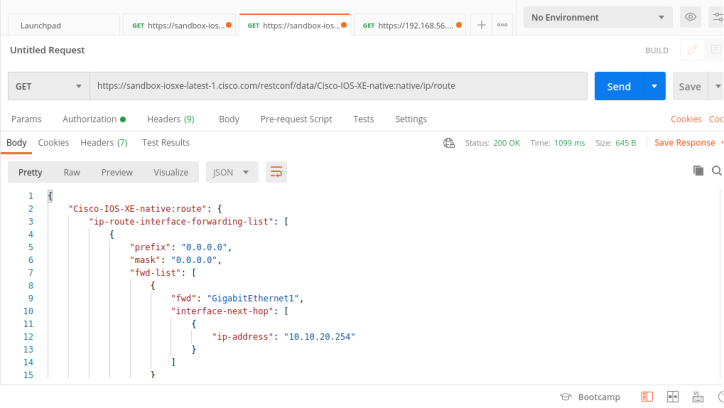
Also, looking at the running-configuration of the router shows that it exists now in the router. This concludes the Network Topic 1: OSPF, be automated using Python with RESTCONF

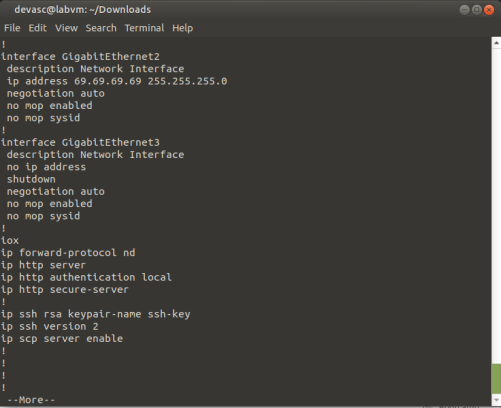
1. **NETWORK TOPIC 2: Configuring Static IP Routing with the URI endpoint**
   1. Verify URI endpoint for IP Routing, check the contents with Postman.

Just as said earlier, the URI endpoint is found out through trial and error, with the use of documentations. In this case, Static IP Routing, the URI endpoint is defined by:

“https://<HOST ADDRESS>/restconf/data/Cisco-IOS-XE-native:native/ip/route”

This can also be verified by entering the address of the router through Postman. As mentioned earlier, to demonstrate individual network configuration, one router is used for this time for Static Routing. In this case, IP Route will be configured on IOS XE on CSR Latest Code AlwaysOn1, from Cisco DevNet sandbox.

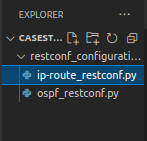




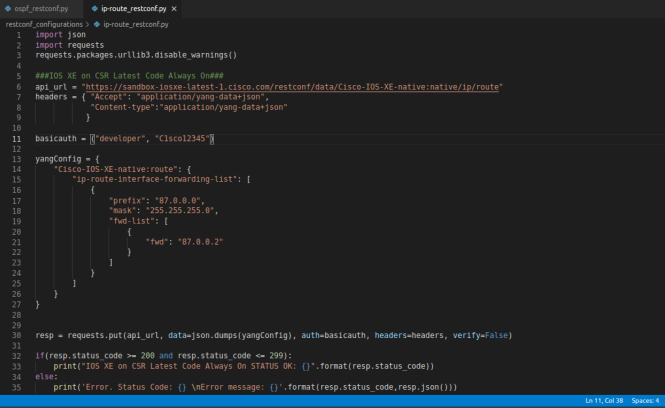
It can be observed that the response is 200: OK, with content returned. This is one of the caveats in using sandbox. Anyone can access, the routers anytime. Nonetheless, the configuration can still be done properly. Moreover, also looking at the running configuration of the router, there is no other configurations of ip route, this will be shown through the following steps.

* 1. Use Python script with RESTCONF to configure OSPF.

In the folder named restconf\_configurations, create a file called ip-route\_restconf.py.



This will contain the script to put request the router Static Routing configuration in the router.

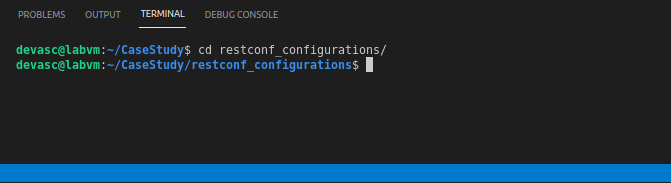


*Note: As noted before, the code have the same structure. To reiterate the json and requests modules have been imported. The URI endpoint for the configuration for the network subject is then defined by api url. To specify that json data is delivered and accepted, headers are included. The router's username and password are included in Basicauth. The model in json format that would be passed with the request is yangConfig. The answer status is then shown using an if else statement, whether it is good or bad.*

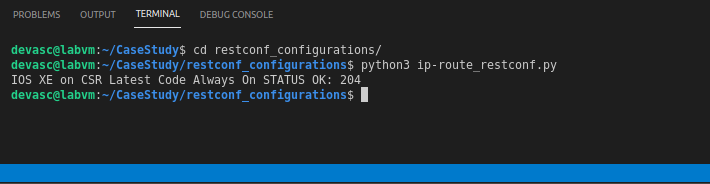
* 1. Run the Python script and verify if it works, by checking through Postman

Open a New Terminal in VSCode.

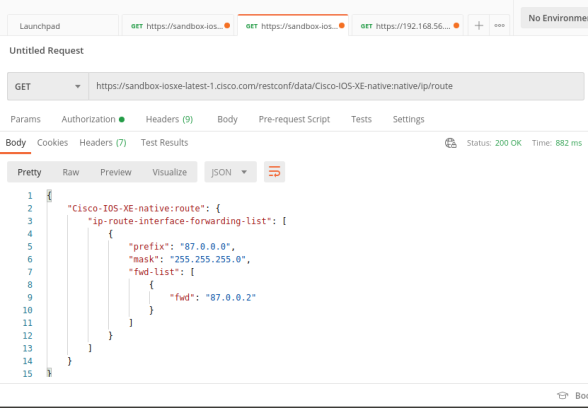
Navigate to the restconf\_configurations folder. Enter “python3 ip-route\_restconf.py”.

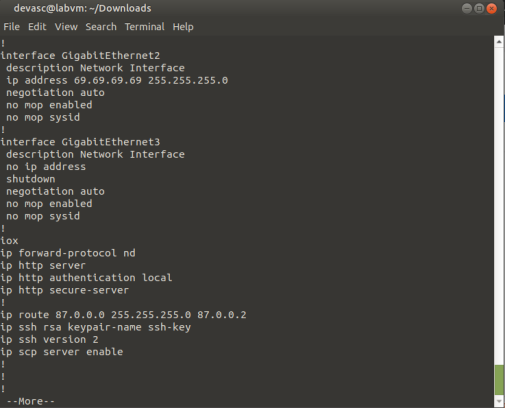


Then wait for a few seconds, the response should be similar to what is inside the script, in this case “IOS XE on CSR Latest Code Always On STATUS OK: 204”



To see the results, go back to your Postman GET request, then click SEND again. This time, it will show the ip route configurations that have been made with the script.

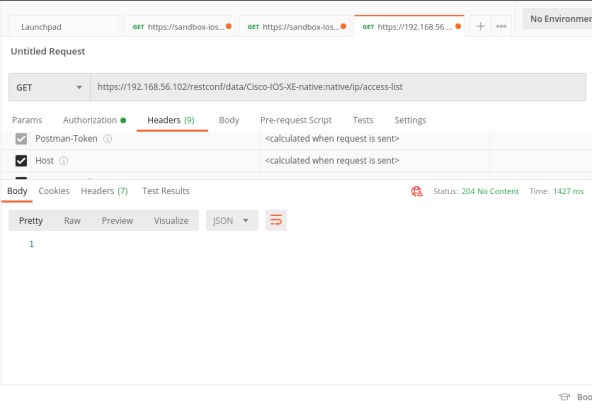


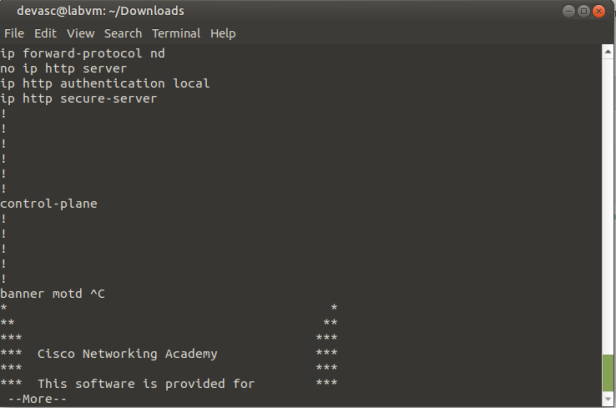


Also, looking at the running-configuration of the router shows that the static routing exists now in the router. This confirms that the script for Static Routing works well.

1. **NETWORK TOPIC 3: Configuring ACL with the URI endpoint**
   1. Verify URI endpoint for IP Routing, check the contents with Postman.

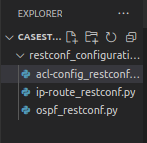
Similar with the router configured with OSPF, the router did not deliver any JSON data, and the status is Status: 204 No Content. This is normal; it implies that there is an empty response because the router does not have router ACL configured. When you execute the show run command on the router, you'll notice that there's no ACL configuration, which is generally found near the end of the interfaces list.



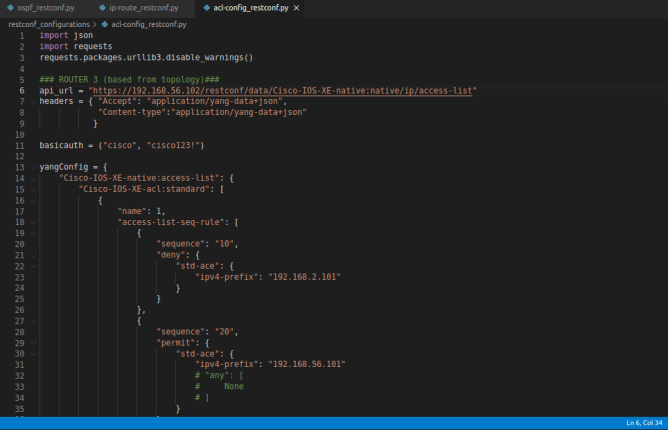


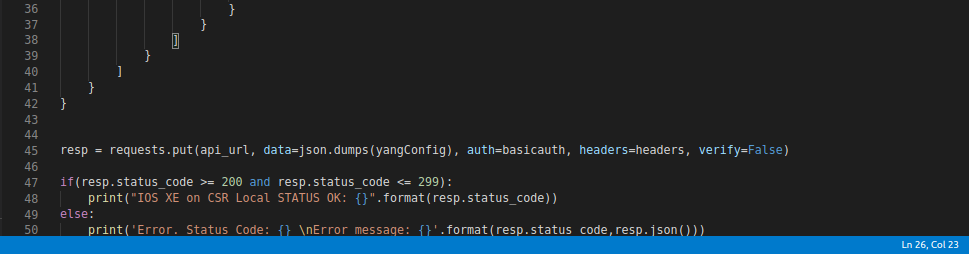
* 1. Use Python script with RESTCONF to configure OSPF.

In the folder named restconf\_configurations, create a file called acl-config\_restconf.py.



This will contain the script to put request the router OSPF configuration in the router.

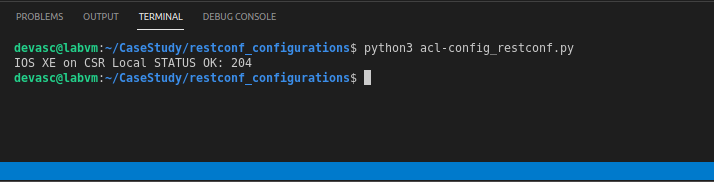




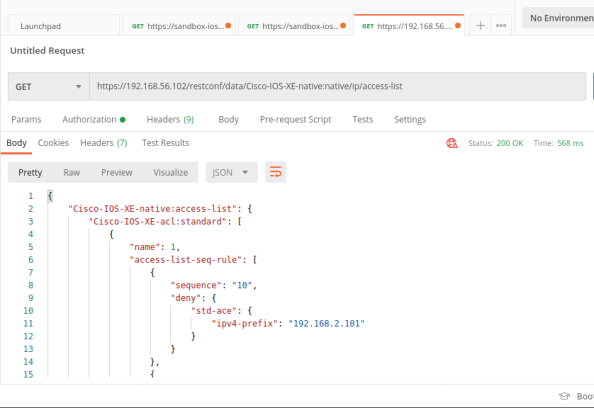
*Note: The script, again, has the same structure. To review, json and requests modules are included. The api url is then used to specify the URI endpoint for the network subject's setup. Headers are used to indicate how json data is sent and accepted. Basicauth includes the router's login and password. yangConfig is the json model that would be supplied with the request. An if else statement is then used to show if the answer is good or bad.*

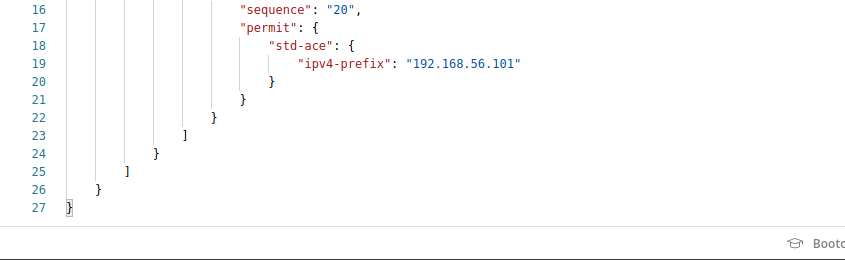
* 1. Run the Python script and verify if it works, by checking through Postman

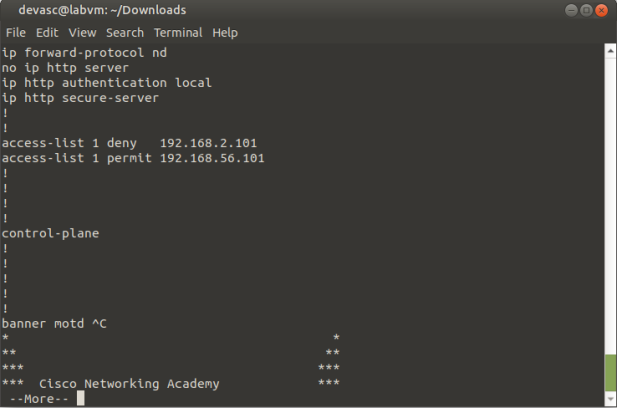
Open a New Terminal in VSCode. Navigate to the restconf\_configurations folder. Enter “Python3 acl-config\_restconf.py”. Then wait for a few seconds, the response should be similar to what is inside the script, in this case “IOS XE on CSR Local STATUS OK: 204”



To see the results, go back to your Postman GET request, then click SEND again. This time, it will show the configurations that have been made with the script.



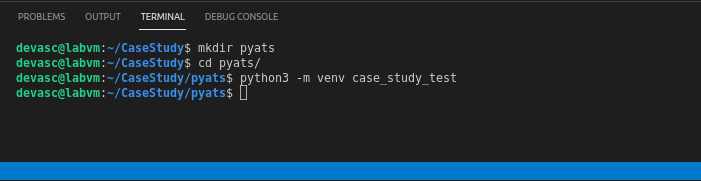


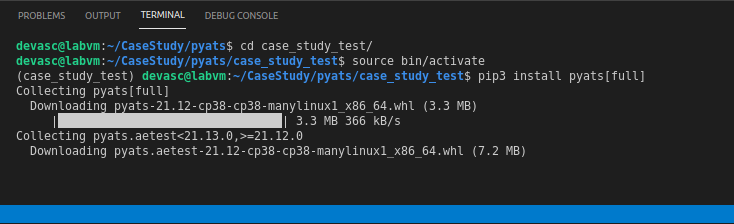


Then, looking at the running configuration of the router, the ACL can now be seen. This shows that the script for configuring ACL in the local router is successful.

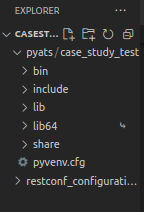
**Part 2: Testing the network topics with pyATS**

1. Create a directory and virtual environment for pyATS. then install the pyATS full module



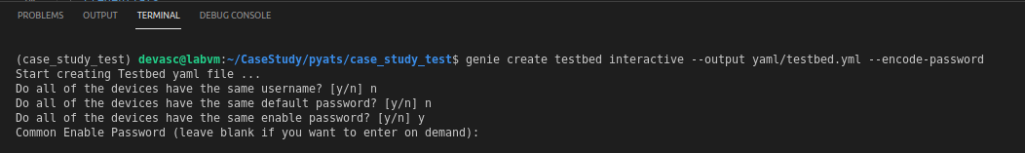


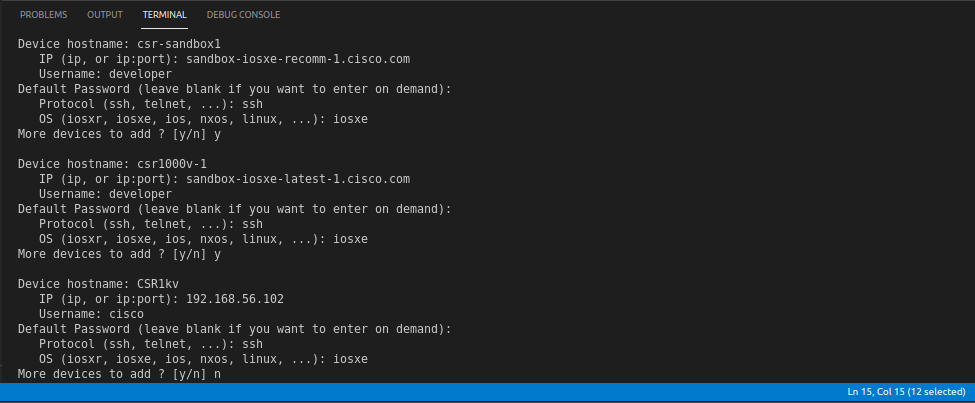
It will take time to complete, as it finishes, verify if the following structure of the folder will show up.



1. Create pyATS genie testbed for the three routers

Issue the: genie create testbed interactive --output yaml/testbed,yml --encode password. Sets of questions for configuration will show up. Enter the information as shown below





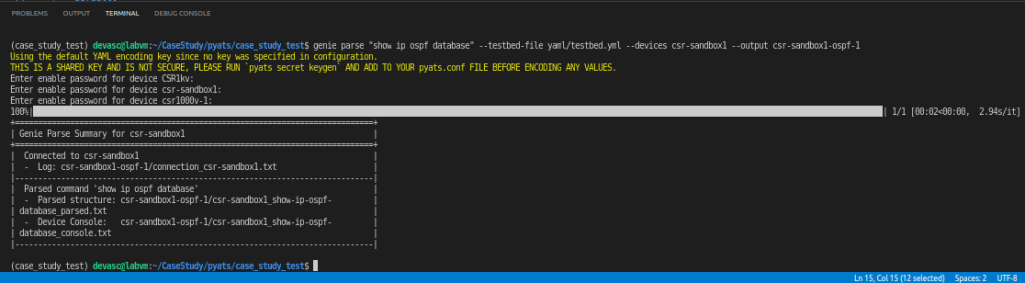
After doing so, the testbed.yml file will be created, with the following content. This will be used in conducting the pyATS with genie tests



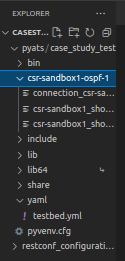
1. Implement pyATS genie with the three network topics

**OSPF**

Issue the: genie parse “show ip ospf database” --testbed-file yaml/testbed.yml --devices csr-sandbox1 –output csr-sandbox1-ospf-1. If successful, certain files will be produced, such as the command parsed.txt, as well as the command console.txt files. This will be used later in checking for difference

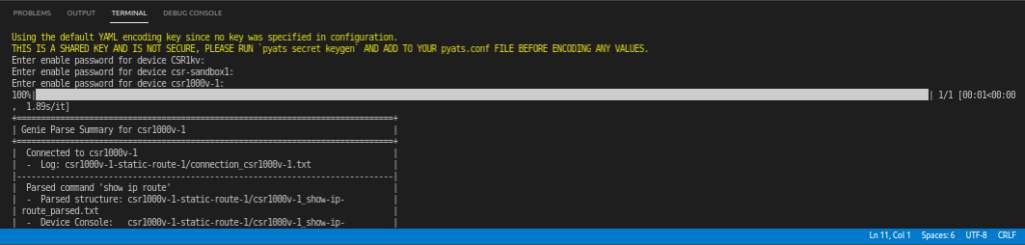


The folder csr-sandbox1-ospf-1 with the following files will then be observed

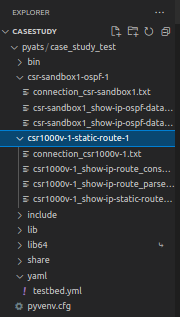


**STATIC ROUTE**

Issue the: genie parse “show ip route” --testbed-file yaml/testbed.yml --devices csr1000v-1 –output csr1000v-1-static-route-1. If it succeeds, the command parsed.txt, as well as the command console.txt files will be produced. This will be used later in checking for difference too.

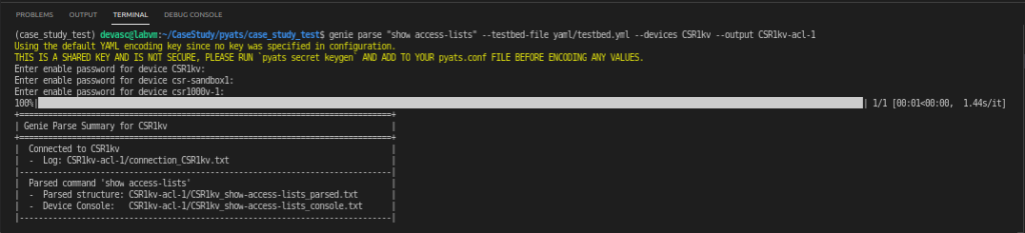


The folder csr-sandbox1-ospf-1 with the following files will then be observed in the panel.

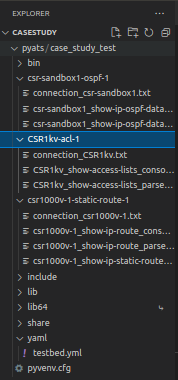


**ACL**

Issue the: genie parse “show access-lists” --testbed-file yaml/testbed.yml --devices CSR1kv –output CSR1k-acl-1. If it proceeds and succeeds, the command parsed.txt, and the command console.txt files will be produced too. This will be used later in checking for difference as well.



The folder CSR1k-acl-1 with the following files will then be observed created on the left panel

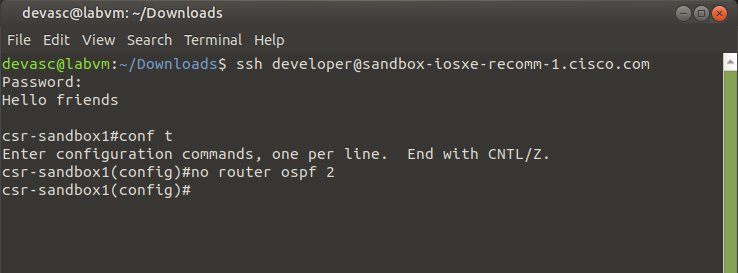


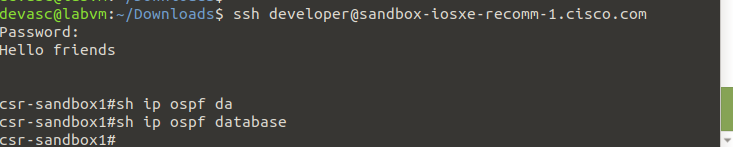
1. Modify some of the configurations for the three network topics

To check for the difference, example scenarios for each network topics and their own routers are made. In the first router (IOS XE on CSR Recommended Code AlwaysOn) with OSPF, the router ospf configuration is removed. In the second router (IOS XE on CSR Latest Code AlwaysOn1), the ip route is configured to route with a certain interface. In the third router (IOS XE on CSR Local), the ACL configuration is removed as well.

**OSPF**

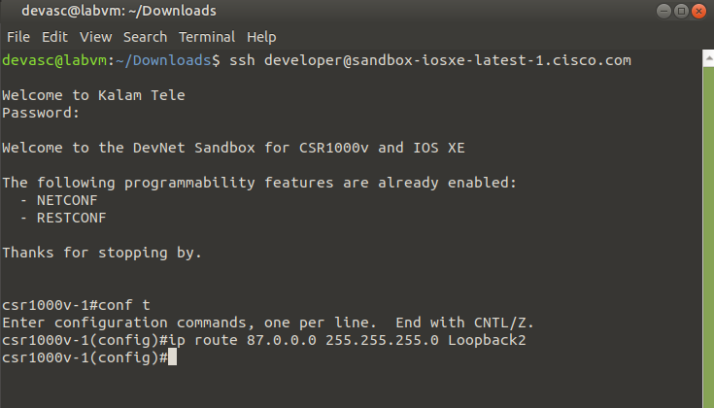
Try removing the OSPF configuration in the router by issuing the no router ospf 2 (which is the router ospf configured in the python script). Then press enter. This should remove the sample router ospf configuration from the router, verify with the “show ip ospf database”.

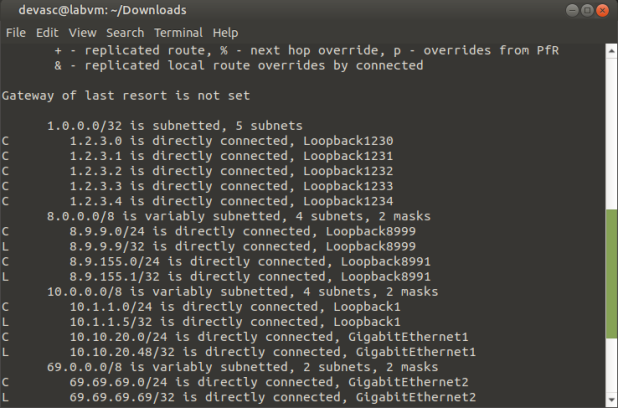




**STATIC ROUTE**

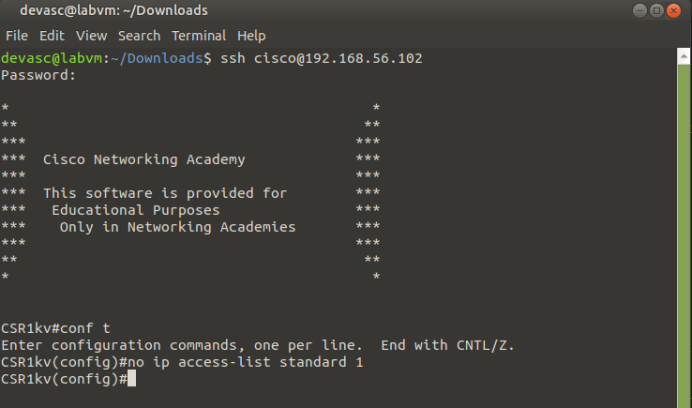
Use an existing interface to route with the ip routing specified (can be different, but better if similar with the one in the python script). Verify the modification by issuing the “show ip route” command.

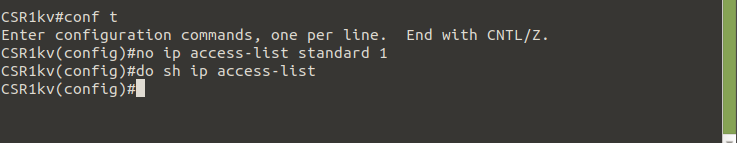




**ACL**

Try also, removing the ACL configuration by issuing “no ip access-list standard 1” which is the ACL configured with the script. Then verify if removed, using the “show access-lists” command.

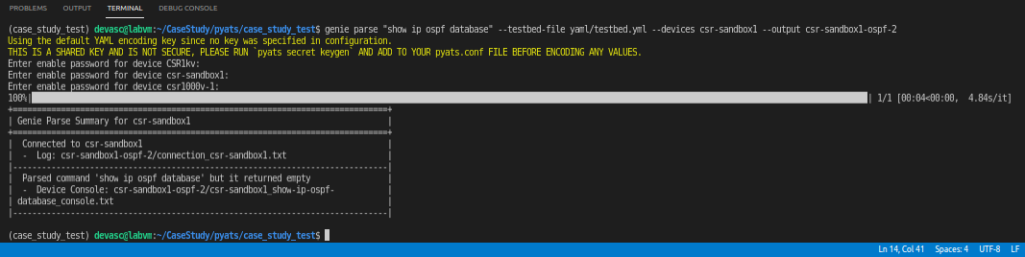




1. Implement pyATS genie with the three network topics after modification

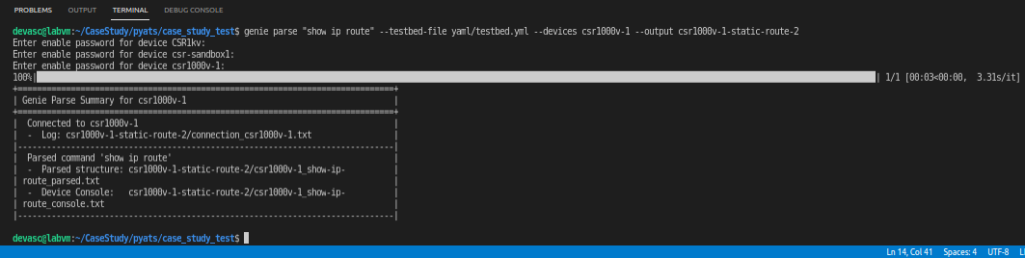
**OSPF**

Repeat the step in creating a testbed yaml file for the network topic. But this time, use: genie parse “show ip ospf database” --testbed-file yaml/testbed.yml --devices csr-sandbox1 –output csr-sandbox1-ospf-2. This adds another folder with 2 as the last character. Do the same for the other network topics. It can be observed in this network topic that the returned value is empty. This is good since the configuration was removed for testing afterwards.



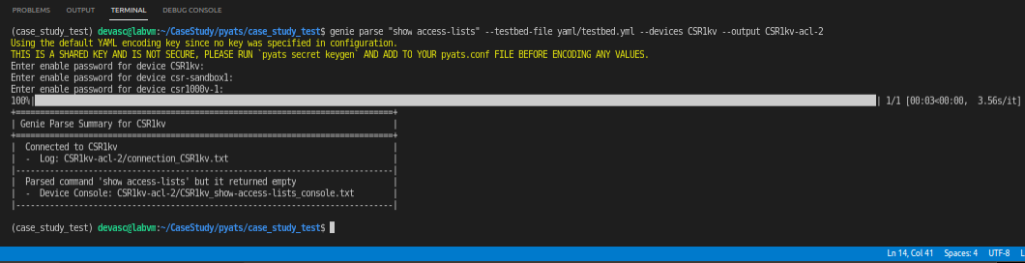
**IP ROUTE**

Here, issue the: genie parse “show ip route” --testbed-file yaml/testbed.yml --devices csr1000v-1 –output csr1000v-1-static-route-2 command. This returns the parsed and console text files, since what was done is to add ip routing.



**ACL**

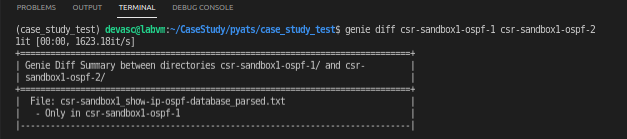
Then: genie parse “show access-lists” --testbed-file yaml/testbed.yml --devices CSR1kv –output CSR1k-acl-2 for ACL. Similar with the OSPF, the test returned empty. This is because the configuration was removed as well, which will be tested afterwards.



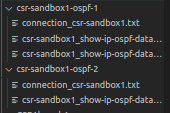
1. Do genie diff of the original testbed, and the new ones after modification.

**OSPF**

In OSPF use: genie diff csr-sandbox1-ospf-1 csr-sandbox1-ospf-2. This will perform comparisons. In this case, there are no differences detected, instead it only sees the csr-sandbox1\_show-ip-ospf-database\_parsed.txt file in csr-sandbox1-ospf-1. This is because, the new testbed after modification states that the value it returned is empty. This means that the modification (removing the router ospf 2) is successful, and that it is detected with pyATS with genie.

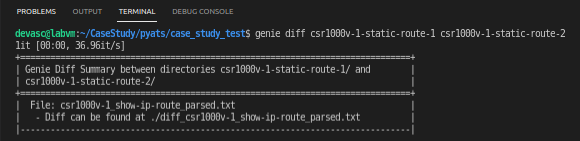


It can be further observed in the panel that there is no csr-sandbox1\_show-ip-ospf-database\_parsed.txt in the one after configuration has been made.

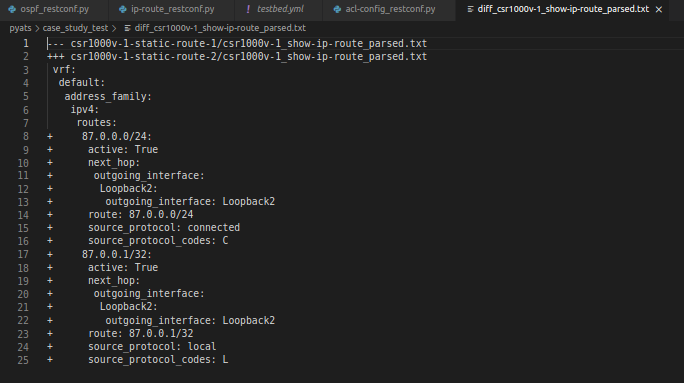


**IP ROUTE**

For Static IP Route Use: genie diff csr1000v-1-static-route-1 csr1000v-1-static-route-2. This compares the original testbed configuration, and the one after modification. As a difference is found, it recorded it in the diff\_csr1000v-1\_show-ip-route\_parsed.txt

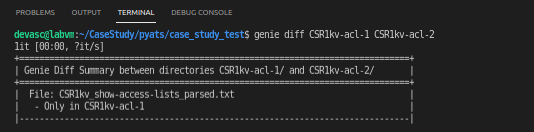


Furthermore, the changed can be seen in the text file produced. It reflected the addition of the ip route 87.0.0.0 255.255.255.0 Loopback2. (There are other configurations due to further testing prior the screenshot)

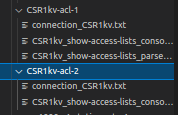


**ACL**

In ACL use: genie parse "show access-lists" --testbed-file yaml/testbed.yml --devices CSR1kv --output CSR1kv-acl-2. Similar with the comparison in OSPF, since the configuration itself is removed, the return value would be empty. Thus, the output file is only seen in the folder CSR1kv-acl-1.



It can be verified by looking at the panel and looking at the folders. The text file containing the parsed configuration can be seen in the CSR1kv-acl-1 folder only.

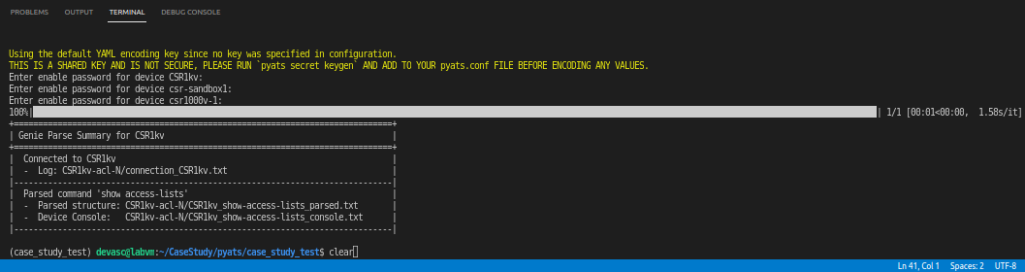


1. Rerun Sample configuration, and output pyATS genie

In this step, the configuration python script for ACL is rerun. It is the preferred router for this step since it is local, and that there are no imminent changes, unlike that of the sandbox that can be modified anytime by anyone. This makes the local router be beneficial for this step. Issue again the python3 acl-config\_restconf.py. Then wait for it to complete.

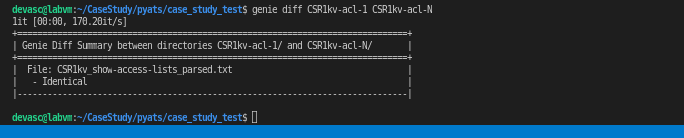
Image

Again, conduct the parse with pyATs with genie, but this time use: genie parse "show access-lists" --testbed-file yaml/testbed.yml --devices CSR1kv --output CSR1kv-acl-N. This will produce the folder with “-N” as the last characters.



1. Genie diff with the original testbed, and the testbed after rerun of the ACL python script.

Issue the command: genie diff CSR1kv-acl-1 CSR1kv-acl-N. It can be observed that the output is Identical. This goes to show that the configuration is successful. Running the python script configuration will yield the same exact results. This is important in case of scenarios such as abrupt changes have occurred.



Conclusion:

Network automation and programmability is an important concept in DevOps. In this case, configuring router network topics using Python scripts with RESTCONF is given focus. As I was doing the case study, I found out what they describe as the power of RESTCONF API. Given that what I did in this case study is simple network configuration and verifying by doing show commands, there is a lot more to explore with RESTCONF. This means that more URI endpoints can be discovered, which covers the network topics we have previously encountered during the previous Cisco courses. Although finding out the URI endpoints was the most tedious task that I have encountered, since there are no direct documentations that point out the exact URI endpoints for each router configuration. Thus, I also found the importance of reading different documentations, and forums, not only in configuring networks, but with programming or coding itself. I was quite dismayed when the CSR1000v did not function properly with GNS3. I was hoping to make an actual simulation of networks, where the routers are interconnected as well as Host PCs. However, through this, I also discovered more of the power of RESTCONF API by configuring routers in the DevNet Sandbox which are deployed in the cloud. It worked as if the routers are connected locally with my DEVASC-LABVM. I was able to configure all with python scripts through RESTCONF API in the same way for each router. Moreover, I was able to perform pyATS with genie with all the configurations made in the router with the python scripts. I was expecting that pyATS will work well only on the locally deployed CSR1000v, but it turns out, even the CSR1000v routers deployed through the DevNet Sandbox can be tested as well. Doing these, I learned how important pyATS with genie is. It does aid greatly in checking or verifying network configurations, rather than manually checking by issuing show commands in the router manually. It automates the issuance of the show commands, and even compares it for you. With all these, I am aiming to test out further capabilities of RESTCONF API, if I get more interested in pursuing network and having a capable device. But nonetheless, I learned about configuring networks RESTCONF API