Cobarrubias Joshua B. CPE 028-CPE41S1

**Final Case Study | Network Automation and Programmability**

# Objectives:

**Part 1:** Design a laboratory activity that discusses the three network topics excluding basic configuration, IP address, and show commands regarding network automation or network programmability.

**Part 2:** Use pyATS to test your network.

**Part 3:** Submit a laboratory activity documentation and video presentation of the FINAL CASE STUDY. Make sure that the CAMERA is ON when recording your video presentation.

**Part 4:** Create a GitHub repository of the FINAL CASE STUDY. Make sure to submit all codes, documentation, and video representation.

**Part 5:** Submit the link of your GitHub repository

1. **Intended Learning Outcome (ILOs):**
   * + - 1. Designing a basic topology
         2. Implementation of ACL and OSPF using ansible
         3. Setting a backup configuration using ansible
         4. Using Pyats to test the network
         5. Sending the files to GitHub repository
2. Diagram

   Description automatically generated**Topology:**
3. **Required Resources:**

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

**Addressing Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Device** | **Interface** | **IP Address** | **Subnet Masks** |
| R1 | S2/0 | 10.0.10.1 | 255.255.255.252 |
|  | F0/0 | 192.168.1.1 | 255.255.255.0 |
| R2 | S2/0 | 10.0.10.2 | 255.255.252.252 |
| PC1 | F0/0 | 192.168.1.2 | 255.255.255.252 |

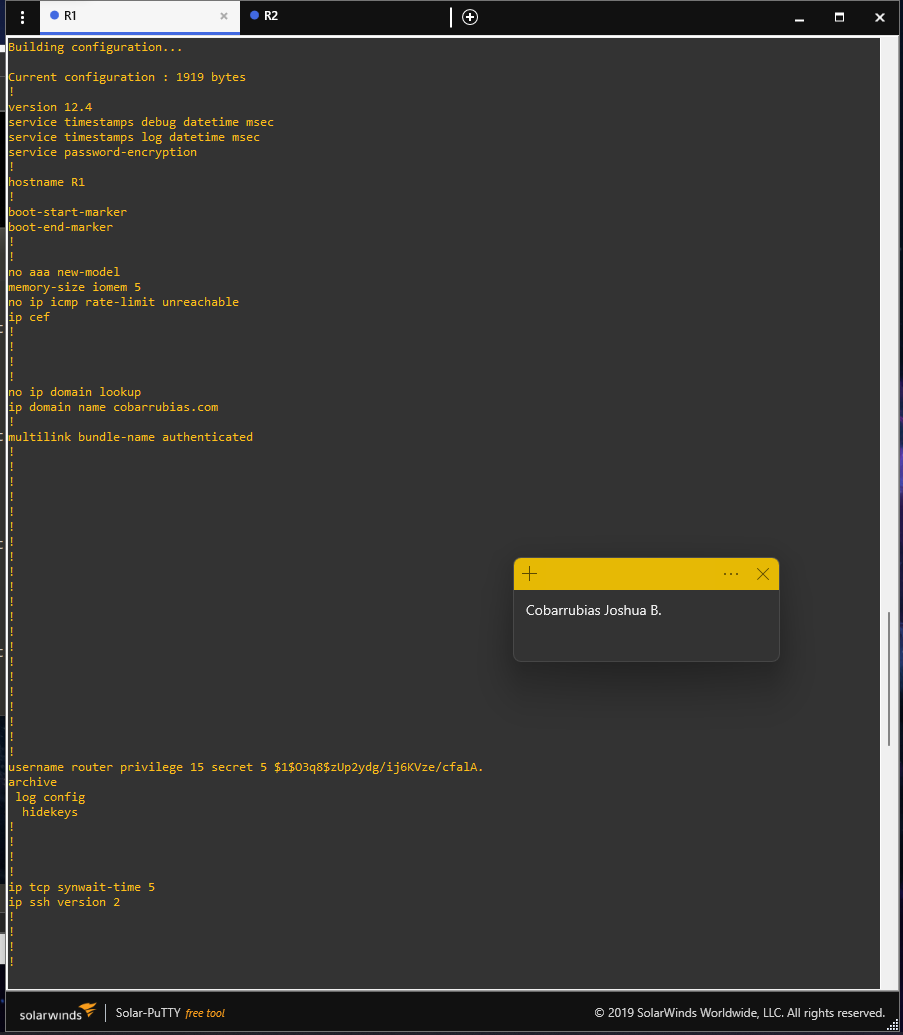
**Instructions**

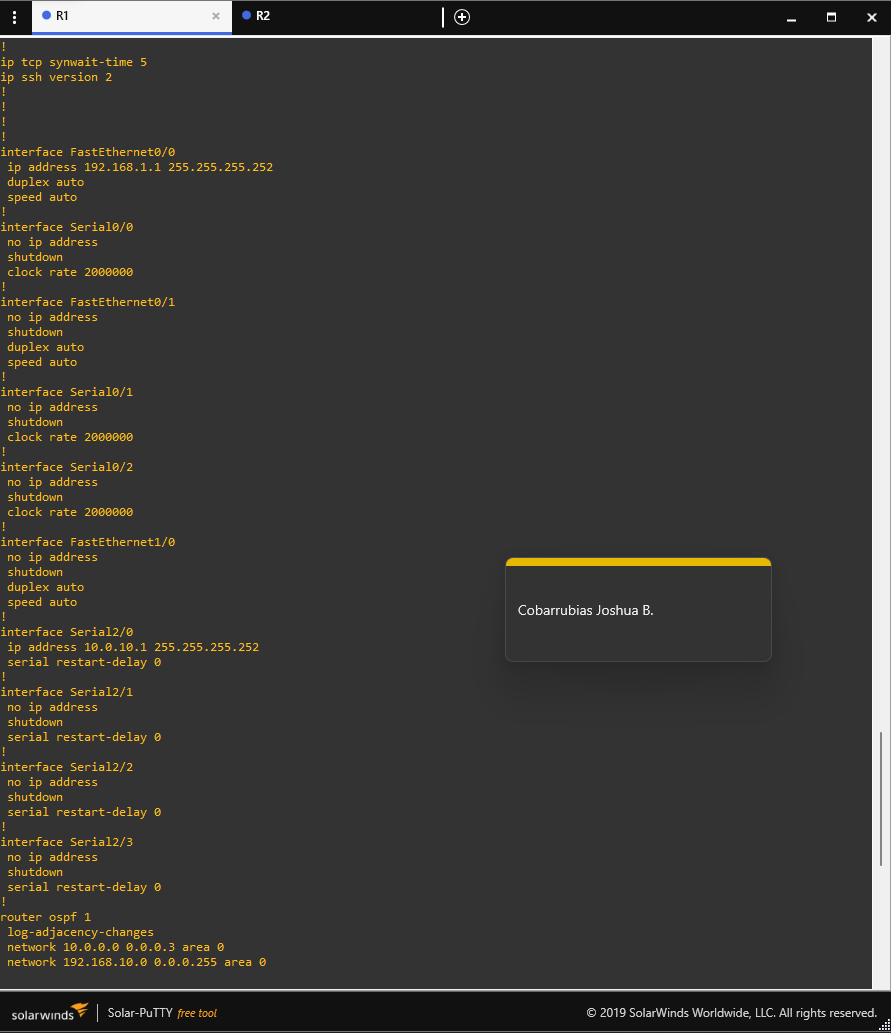
* 1. **Launch the DEVASC VM and GNS3**

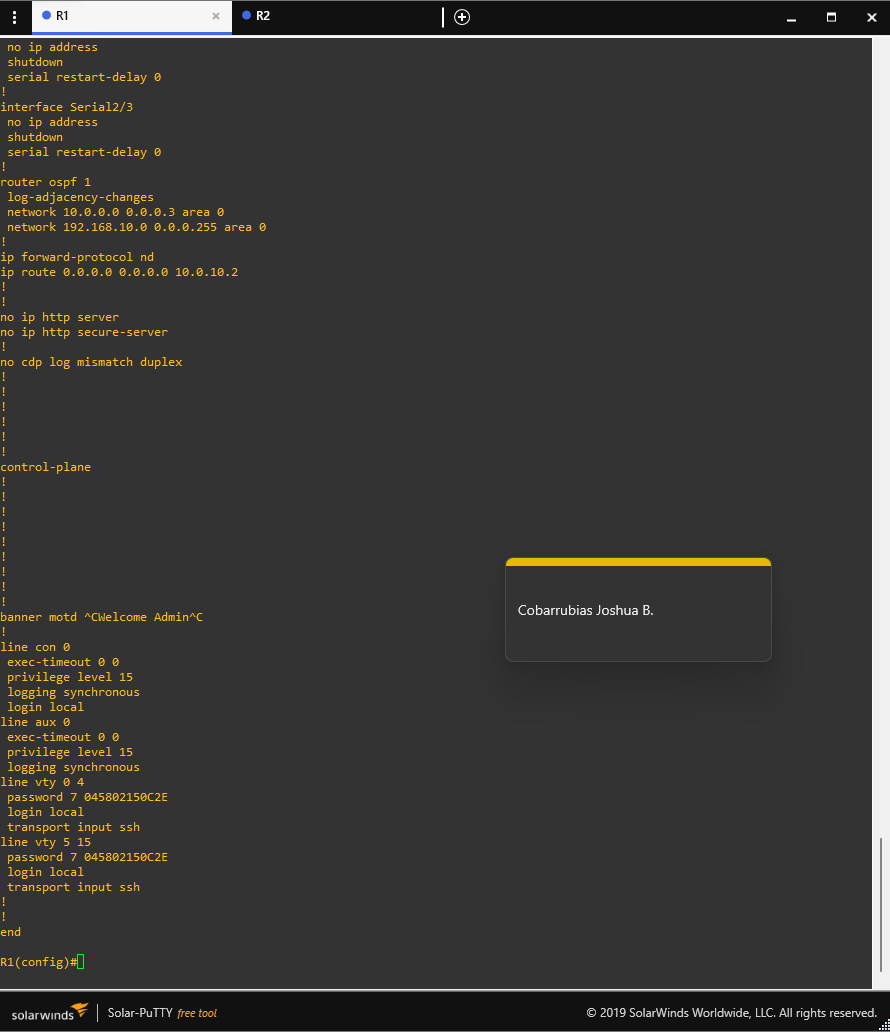
In this part, you launch two VMS. GNS3 and DEVASC.

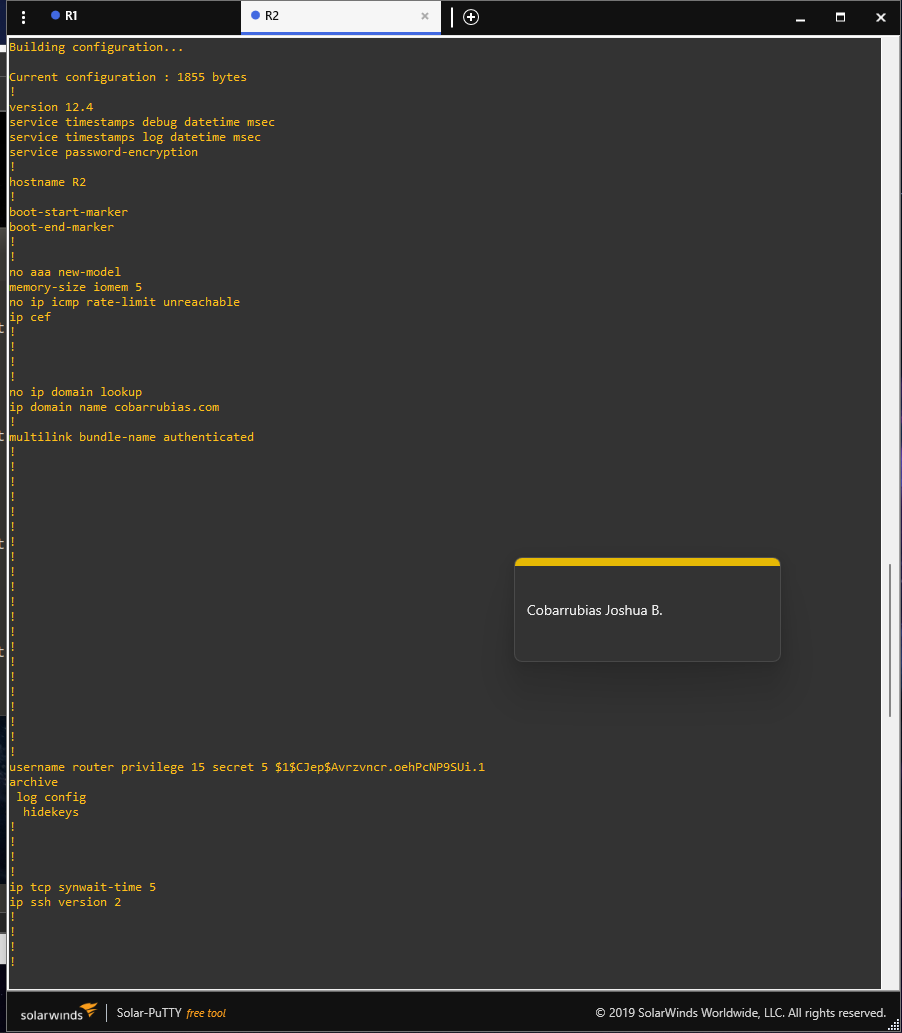
**Graphical user interface, text

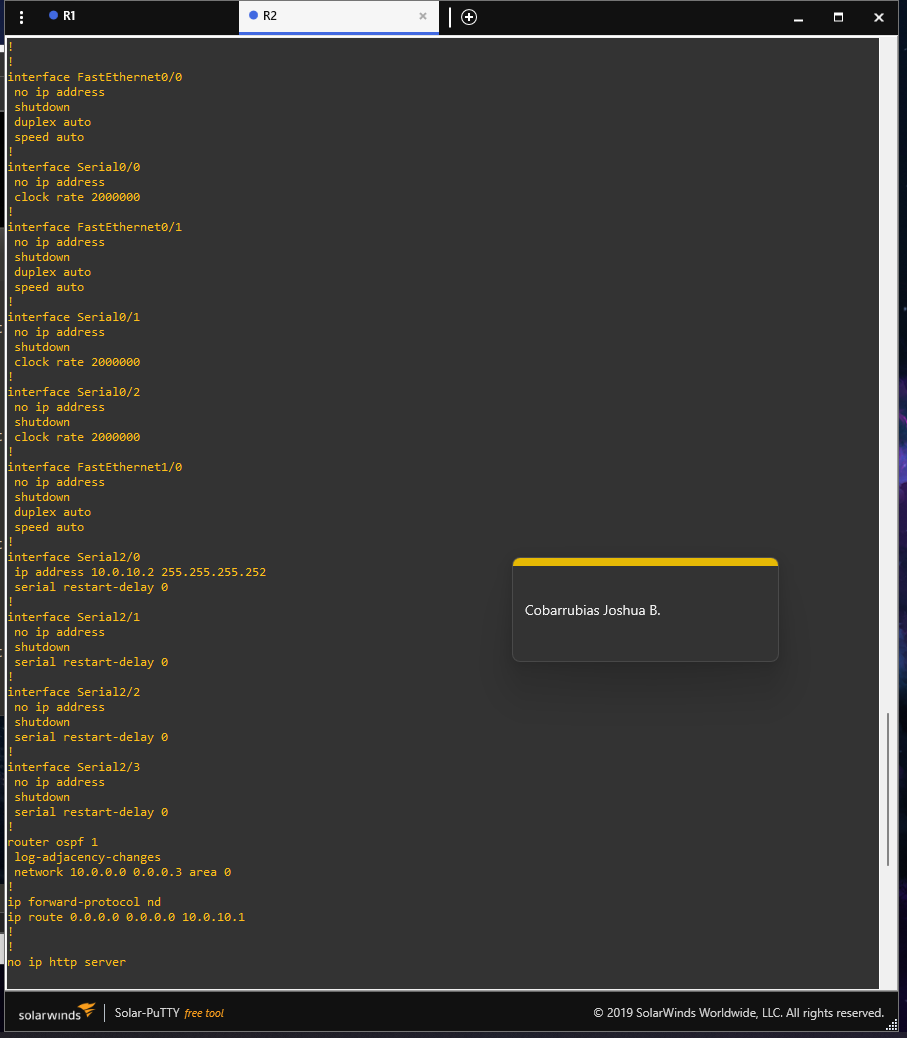
Description automatically generated**

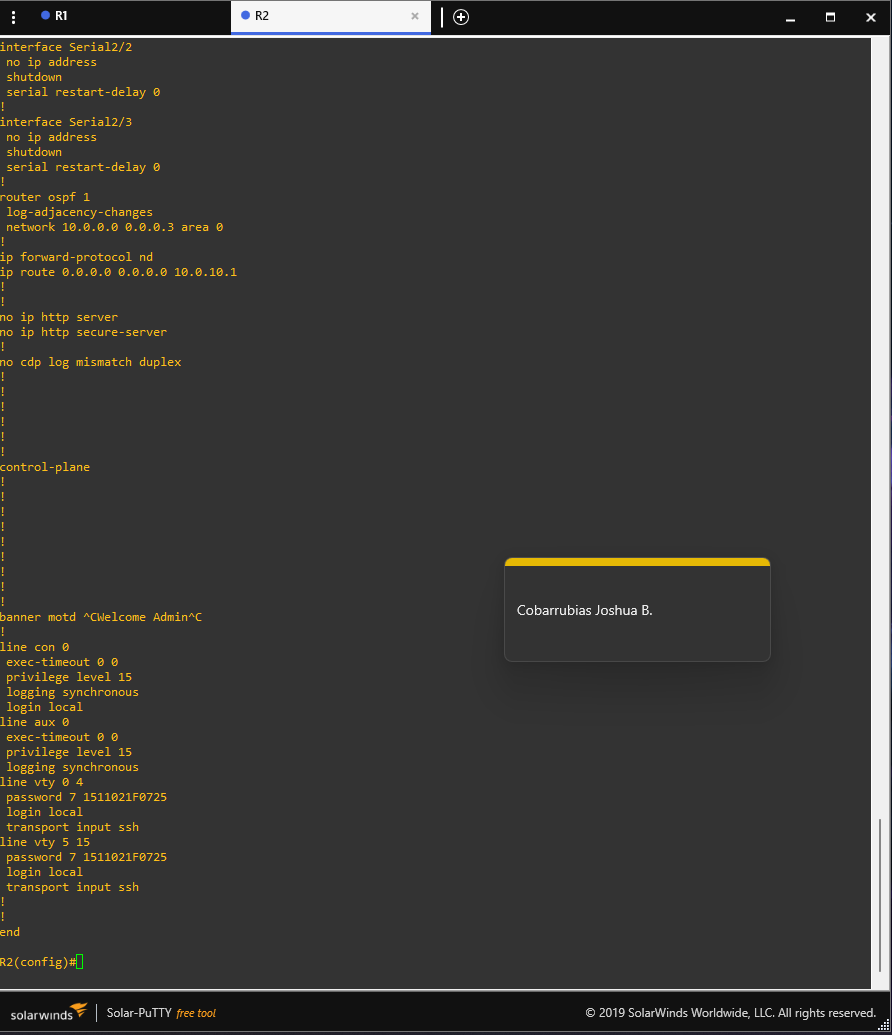
Connect and apply basic configuration on both routers and check if the DEVASC pc VM is connected on R1 and R2.





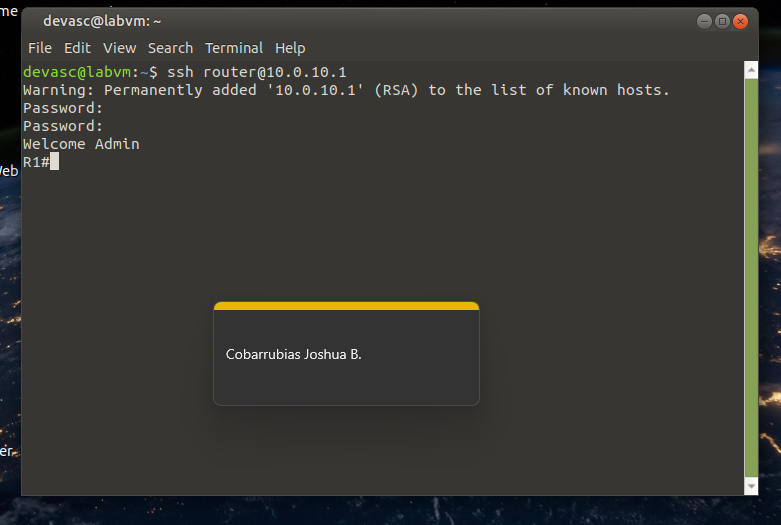


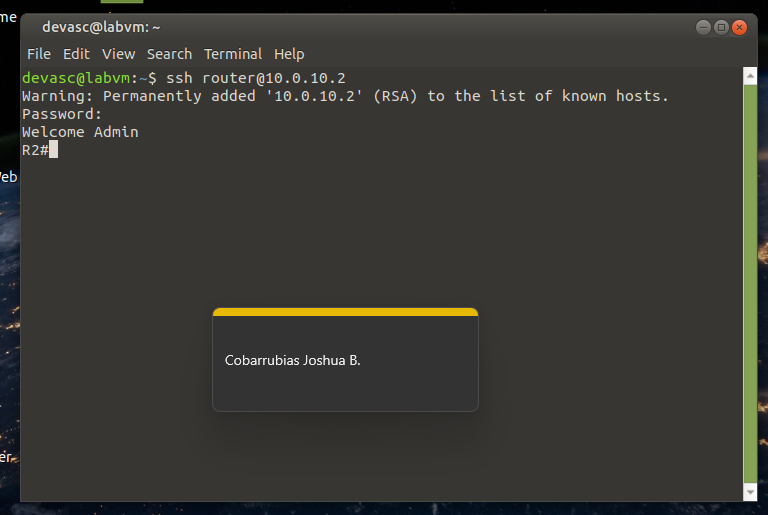




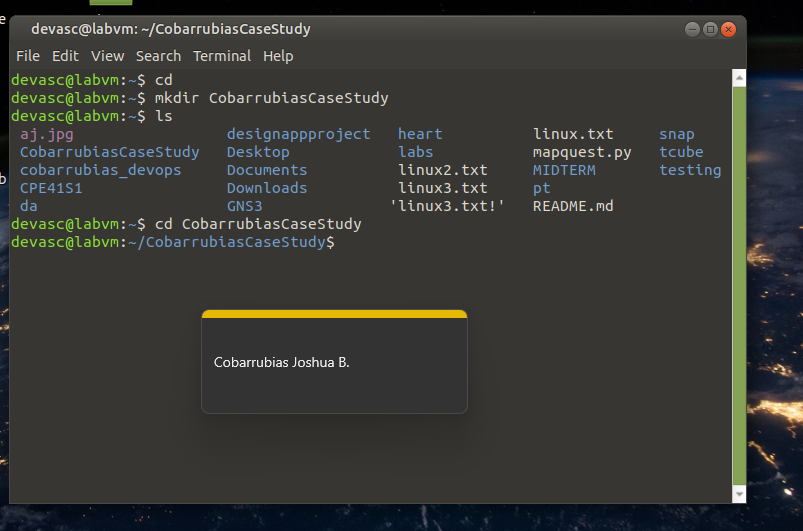
A screenshot of a computer

Description automatically generatedAfter setting up basic configuration and connecting R1 and R2 and DEVASC VM. We need to check if we could ping R1 and R2 and access them on the SSH.

Then will be checking if the SSH of R1 and R2 is accessible and fully operational.



After successfully accessing both routers using SSH we need to go to our DEVASC and create a project directory.



Then we need to open VSCode to open the project file using the directory we made CobarrubiasCaseStudy. Then will be creating a host file containing the ansible username and password on host file.

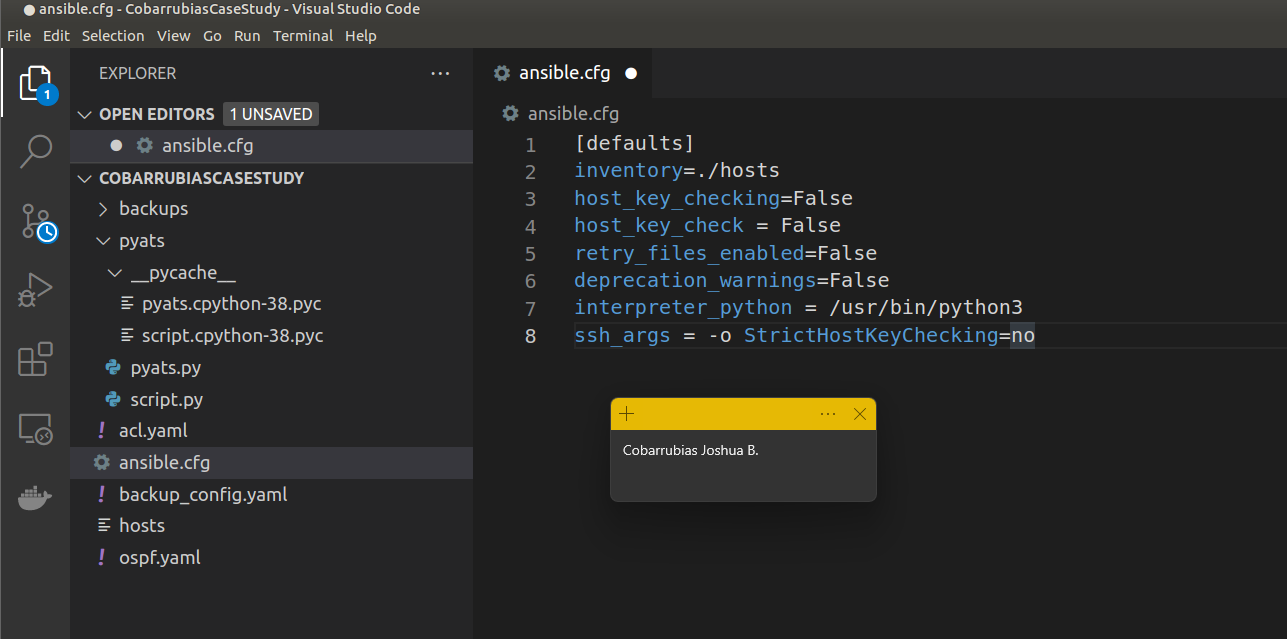
A screenshot of a computer

Description automatically generated

This will be the codes for the hosts file.

R1 ansible\_user=router ansible\_password=cisco ansible\_host=10.0.10.1 ansible\_connection=network\_cli ansible\_network\_os=ios ansible\_become=yes ansible\_become\_method=enable ansible\_become\_pass=cisco

R2 ansible\_user=router ansible\_password=cisco ansible\_host=10.0.10.2 ansible\_connection=network\_cli ansible\_network\_os=ios ansible\_become=yes ansible\_become\_method=enable ansible\_become\_pass=cisco

After creating an hosts file, we need to create an ansible configuration file on the directory naming ansible.cfg. 

This will be the codes for the ansible.cfg

[defaults]

inventory=./hosts

host\_key\_checking=False

host\_key\_check = False

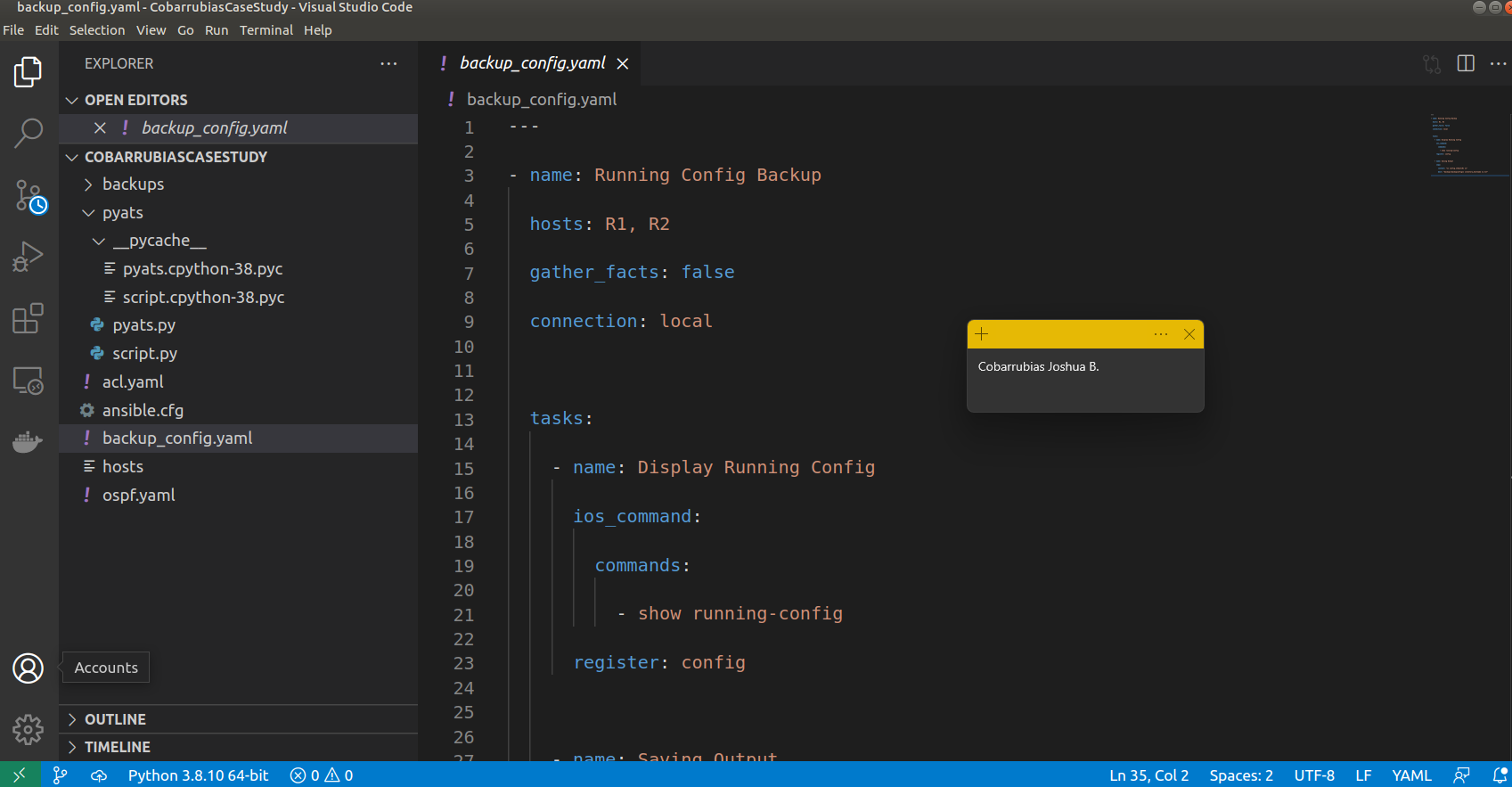
retry\_files\_enabled=False

deprecation\_warnings=False

interpreter\_python = /usr/bin/python3

ssh\_args = -o StrictHostKeyChecking=no

We will now create our yaml file naming backup\_config.yaml this will be the backup of our configuration on our R1 and R2.



This will be the code for the backup\_config.yaml

---

- name: Running Config Backup

hosts: R1, R2

gather\_facts: false

connection: local

tasks:

- name: Display Running Config

ios\_command:

commands:

- show running-config

register: config

- name: Saving Output

copy:

content: "{{ config.stdout[0] }}"

dest: "backups/backupconfig{{ inventory\_hostname }}.txt"

We will now create our ansible configure file will be naming it ansible.cfg

Graphical user interface, text

Description automatically generated

This will be the code for the ansible.cfg

[defaults]

inventory=./hosts

host\_key\_checking=False

host\_key\_check = False

retry\_files\_enabled=False

deprecation\_warnings=False

interpreter\_python = /usr/bin/python3

# ssh arguments to use

ssh\_args = -o StrictHostKeyChecking=no

After creating the ansible.cfg we will now be creating our ACL configuration and naming it acl.yaml

Graphical user interface, text

Description automatically generated

This will be the code for the acl.yaml

---

- name: ACL FOR R1

hosts: R1

gather\_facts: false

connection: local

tasks:

- name: R1 ACL FOR R1

ios\_command:

commands:

- config terminal

- access-list 179 permit tcp 192.168.69.0 0.0.0.255 192.168.69.3 0.0.0.0

- access-list 179 permit udp 192.168.69.0 0.0.0.255 192.168.69.3 0.0.0.255

register: acl

- name: ACL FOR R2

hosts: R2

gather\_facts: false

connection: local

tasks:

- name: ACL SET FOR R2

ios\_command:

commands:

- config terminal

- access-list 186 permit tcp 192.168.2.0 0.0.0.255 192.168.2.3 0.0.0.0

- access-list 186 permit udp 192.168.2.0 0.0.0.255 192.168.2.3 0.0.0.255

register: acl

And for the last we will be creating our OSPF configuration and will be naming it ospf.yaml

Graphical user interface, text

Description automatically generated

This will be the code for the ospf.yaml

---

- name: OSPF FOR R1

hosts: R1

gather\_facts: false

connection: local

tasks:

- name: OSPF SETUP FOR R1

ios\_command:

commands:

- config terminal

- router ospf 1

- network 192.168.1.1 0.0.0.255 area 0

- network 10.0.10.1 0.0.0.3 area 0

- network 10.0.10.2 0.0.0.3 area 0

register: ospf

- name: OSPF FOR R2

hosts: R2

gather\_facts: false

connection: local

tasks:

- name: OSPF SETUP FOR R2

ios\_command:

commands:

- config terminal

- router ospf 1

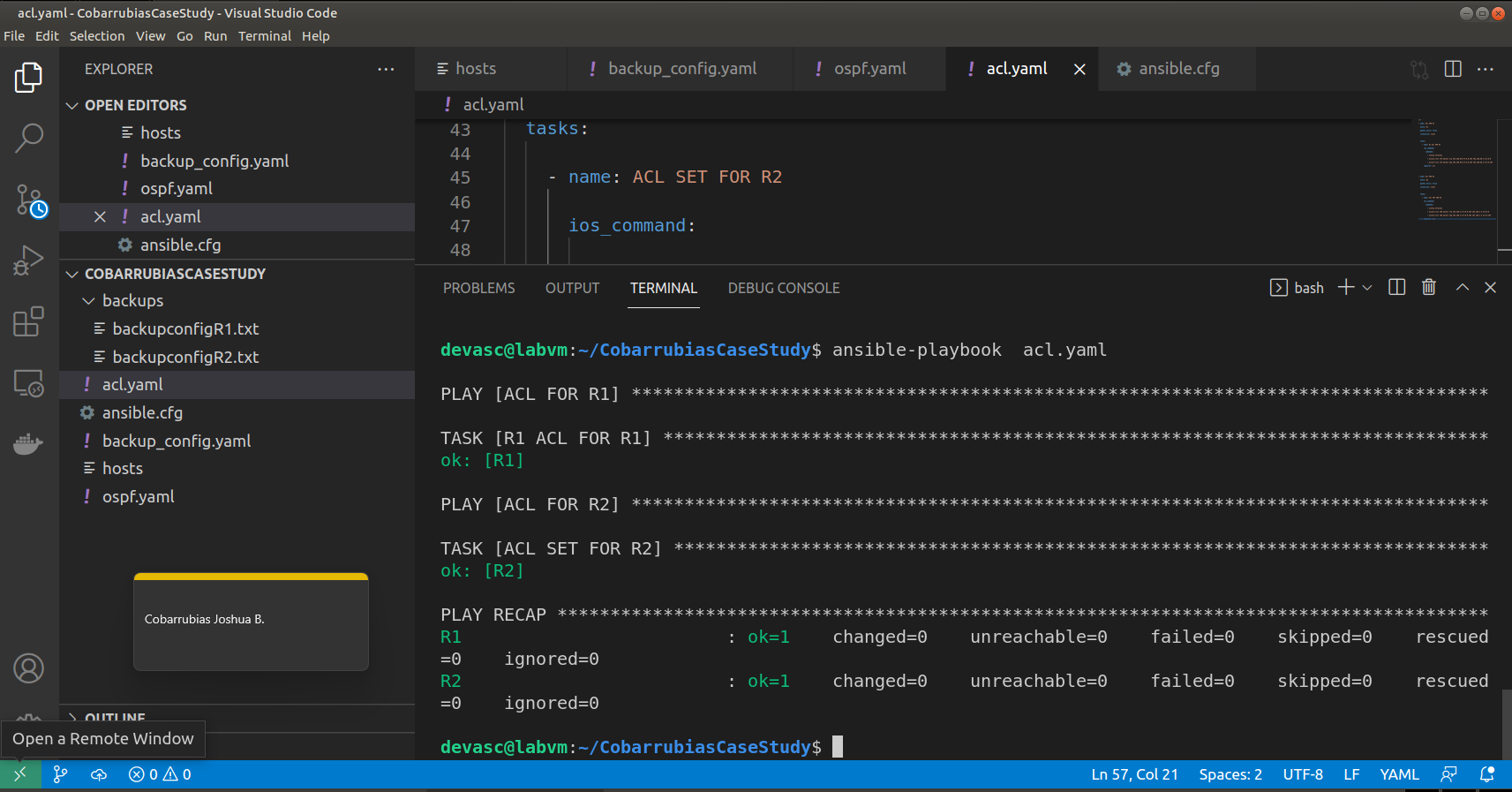
- network 192.168.1.1 0.0.0.255 area 0

- network 10.0.10.1 0.0.0.3 area 0

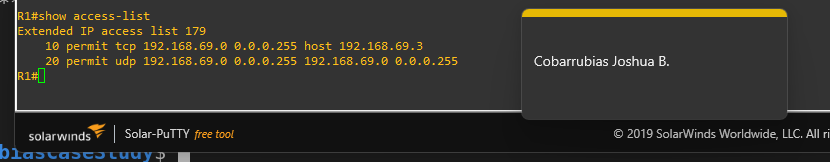
- network 10.0.10.2 0.0.0.3 area 0

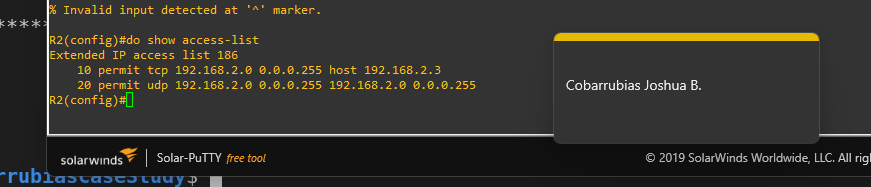
register: ospf

After creating all the files will be running it using ansible-playbook. The first thing will run is the acl.yaml we need to go to terminal and write ansible-playbook acl.yaml



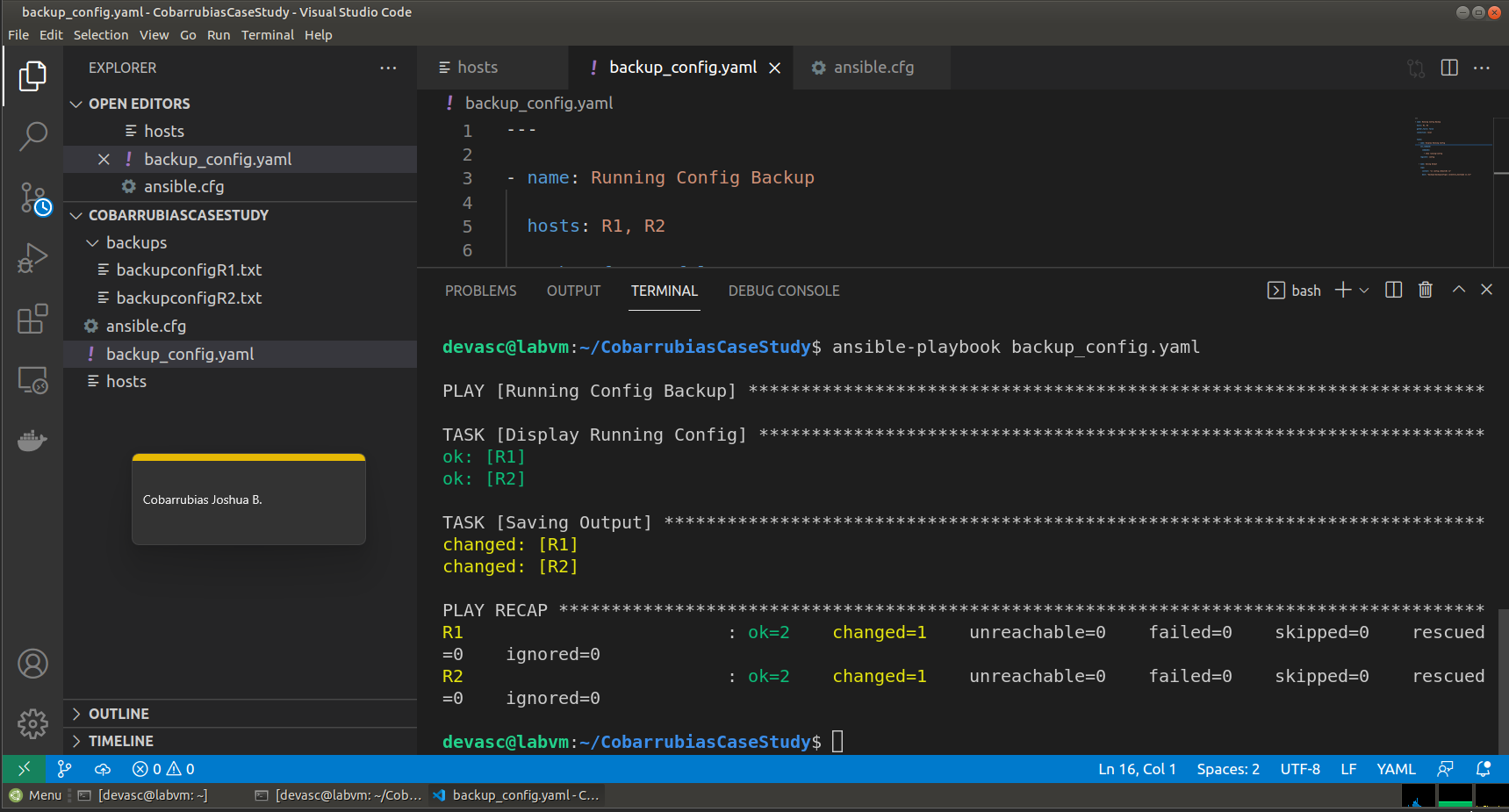
To check if the ACL is implemented we need to go to GNS3 console of R1,R2 and type show access-lists





As you can see the ACL was fully configured using ansible.

The next thing will run is our backup\_config.yaml using ansible by using this code on our terminal on the VSCode ansible-playbook backup\_config.yaml

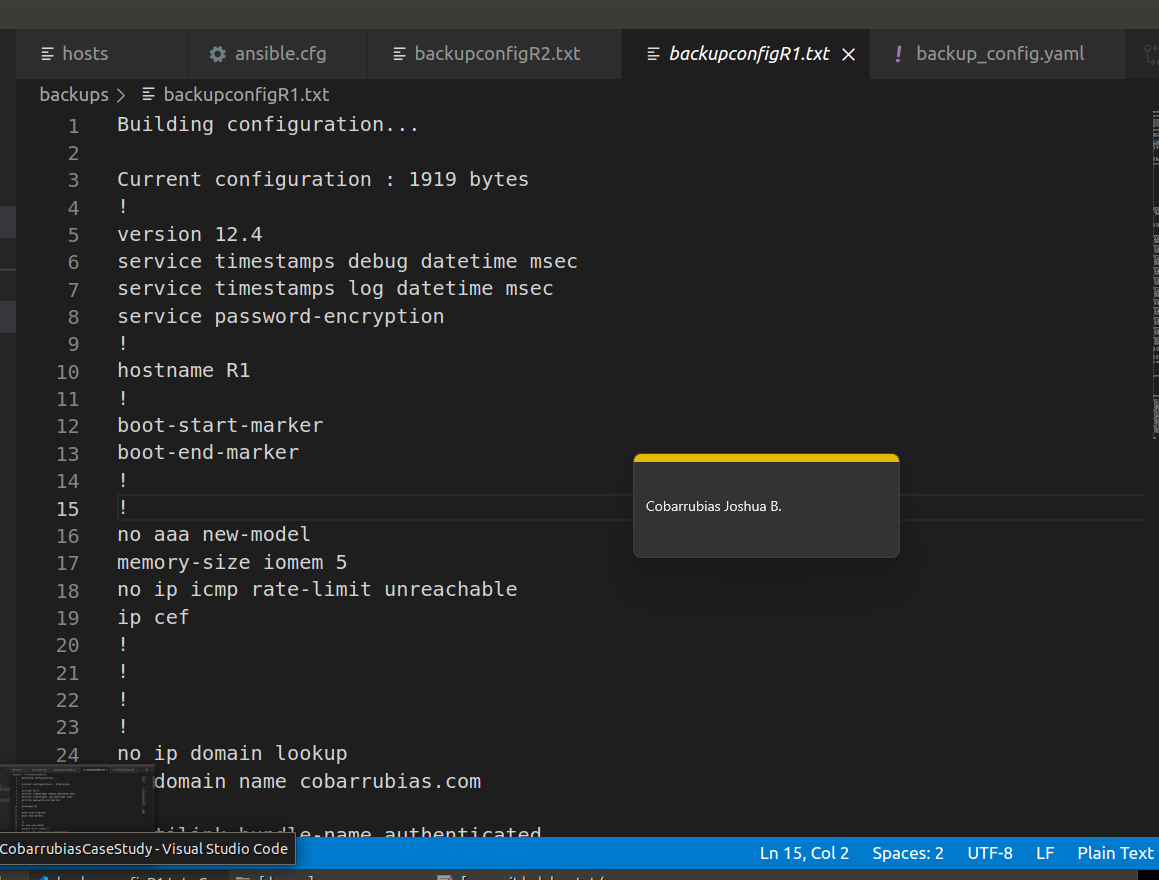


Then to check if the backup\_config.yaml was successfully using ansible we will check to backups folder if it contains our backups\_config by txt file.

Graphical user interface, text, application

Description automatically generated

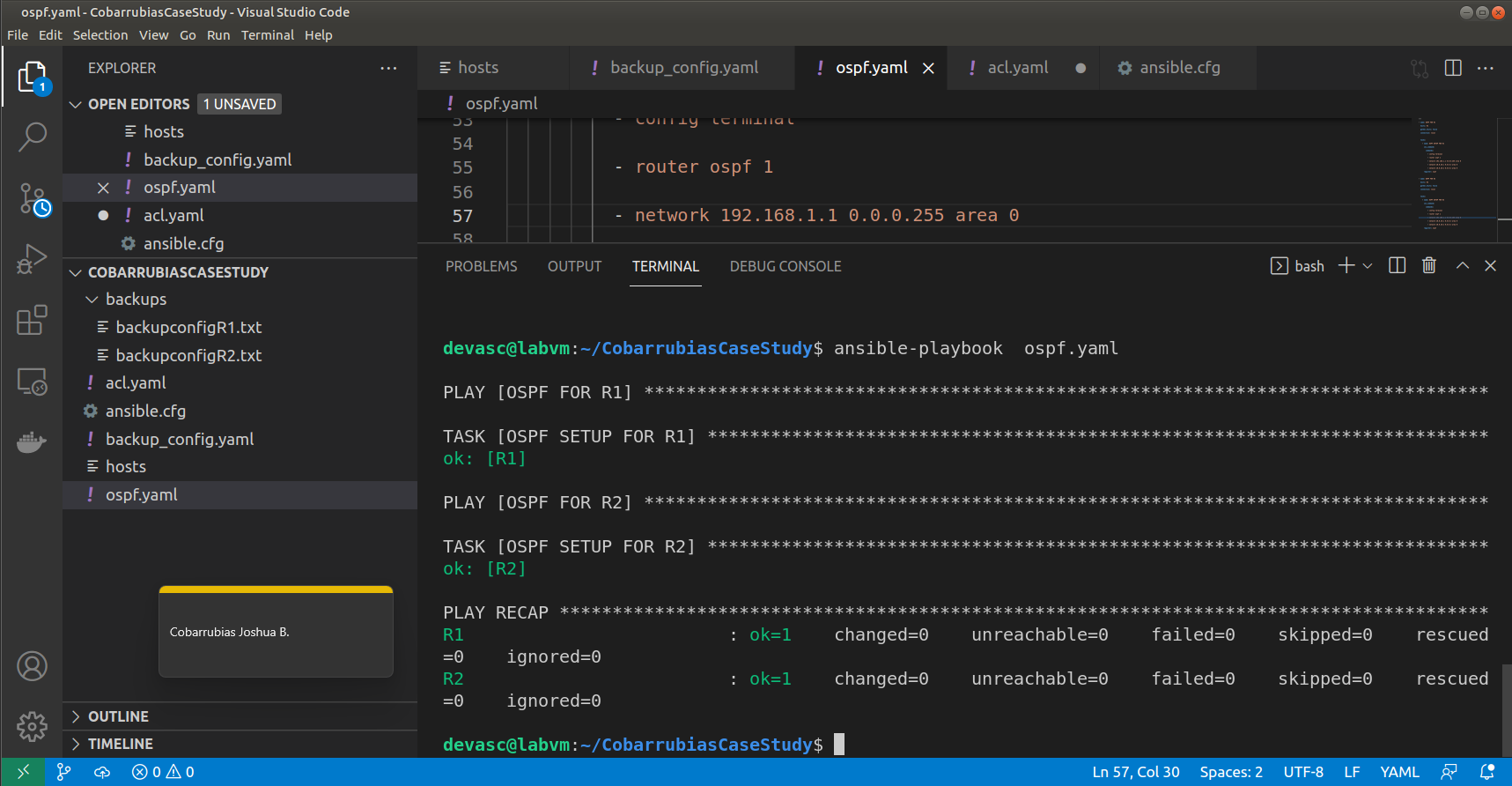
And will be checking if the backupconfigR1.txt file has our configuration same to backupconfigR2.txt

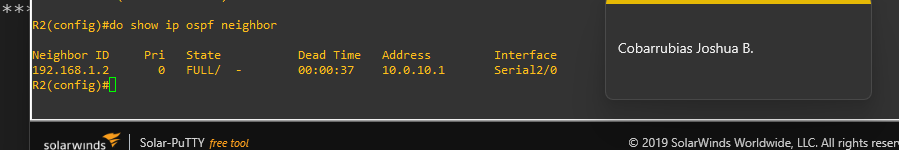
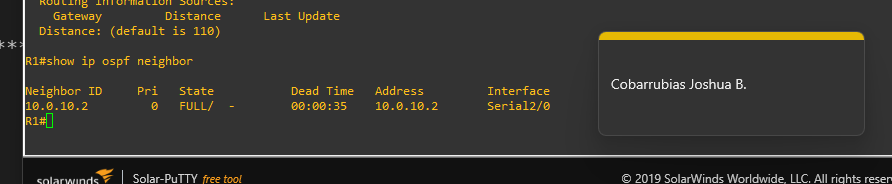


Graphical user interface, text

Description automatically generated

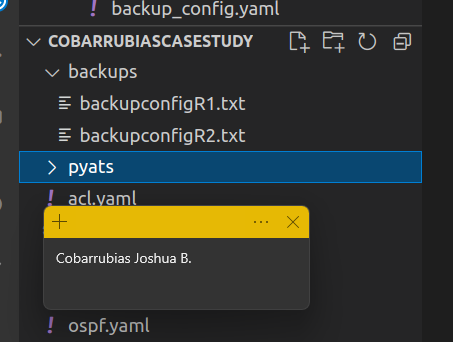
After creating the ACL and backing up the configuration of the routers R1 and R2. We will now create our OSPF on our routers R1 and R2. So, we already made the files for OSPF we will now only test it. By going to the terminal on VSCode and inputting ansible-playbook ospf.yaml

So, as you can see it was successful using ansible now to check if OSPF was successfully implemented we need to go to GNS3 and go to R1 and R2 console and type in show ip ospf neighbor.



So, as you can see OSPF was implemented on both routers R1 and R2 because now it has a neighbor ID of an OSPF.

Now for the final we will now create our pyats by creating a script to test our network. First, we need to create an pyats folder inside our directory.



After creating a folder to our directory, we need to create a pyats.py for testing our network.

Graphical user interface, text

Description automatically generated

This will be the code for pyats.py

import os

from pyats.easy py import run

def main():

test\_path = os.path.dirname(os.path.abspath(\_\_file\_\_))

testscript = os.path.join(test\_path, 'script.py')

run(testscript=testscript)

After creating the pyats.py we will need to create another python file naming it script.py

Text

Description automatically generated

This will be the code for script.py

import logging

from pyats import aetest

log = logging.getLogger(\_\_name\_\_)

class common\_setup(aetest.CommonSetup):

""" Common Setup section """

@aetest.subsection

def sample\_subsection\_1(self):

""" Common Setup subsection """

log.info("Aetest Common Setup ")

@aetest.subsection

def sample\_subsection\_2(self, section):

""" Common Setup subsection """

log.info("Inside %s" % (section))

log.info("Inside class %s" % (self.uid))

class tc\_one(aetest.Testcase):

""" This is user Testcases section """

@aetest.setup

def prepare\_testcase(self, section):

""" Testcase Setup section """

log.info("Preparing the test")

log.info(section)

@ aetest.test

def simple\_test\_1(self):

""" Sample test section. Only print """

log.info("First test section ")

@ aetest.test

def simple\_test\_2(self):

""" Sample test section. Only print """

log.info("Second test section ")

@aetest.cleanup

def clean\_testcase(self):

""" Testcase cleanup section """

log.info("Pass testcase cleanup")

class tc\_two(aetest.Testcase):

""" This is user Testcases section """

@ aetest.test

def simple\_test\_1(self):

""" Sample test section. Only print """

log.info("First test section ")

self.failed('This is an intentional failure')

@ aetest.test

def simple\_test\_2(self):

""" Sample test section. Only print """

log.info("Second test section ")

@aetest.cleanup

def clean\_testcase(self):

""" Testcase cleanup section """

log.info("Pass testcase cleanup")

class common\_cleanup(aetest.CommonCleanup):

""" Common Cleanup for Sample Test """

@aetest.subsection

def clean\_everything(self):

""" Common Cleanup Subsection """

log.info("Aetest Common Cleanup ")

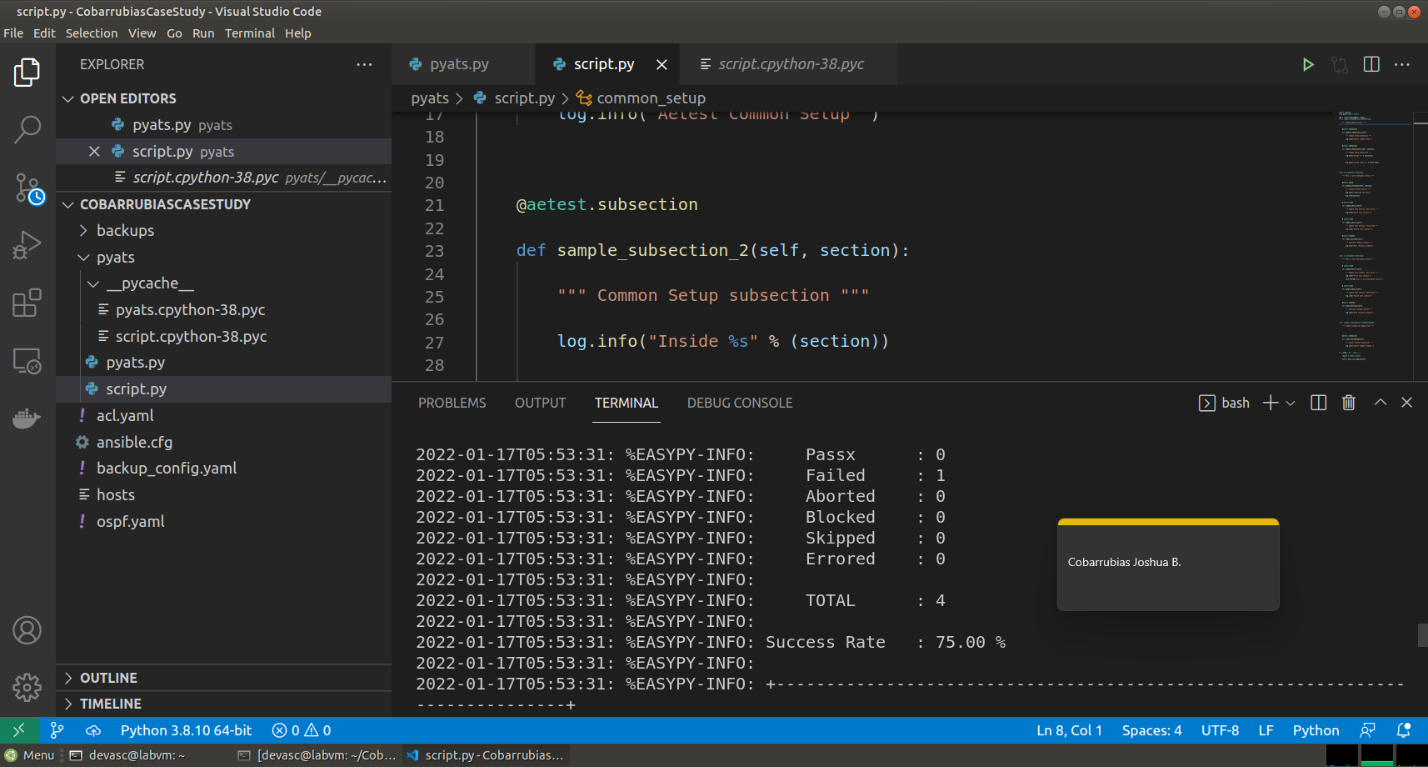
if \_\_name\_\_ == '\_\_main\_\_':

result = aetest.main()

aetest.exit\_cli\_code(result)

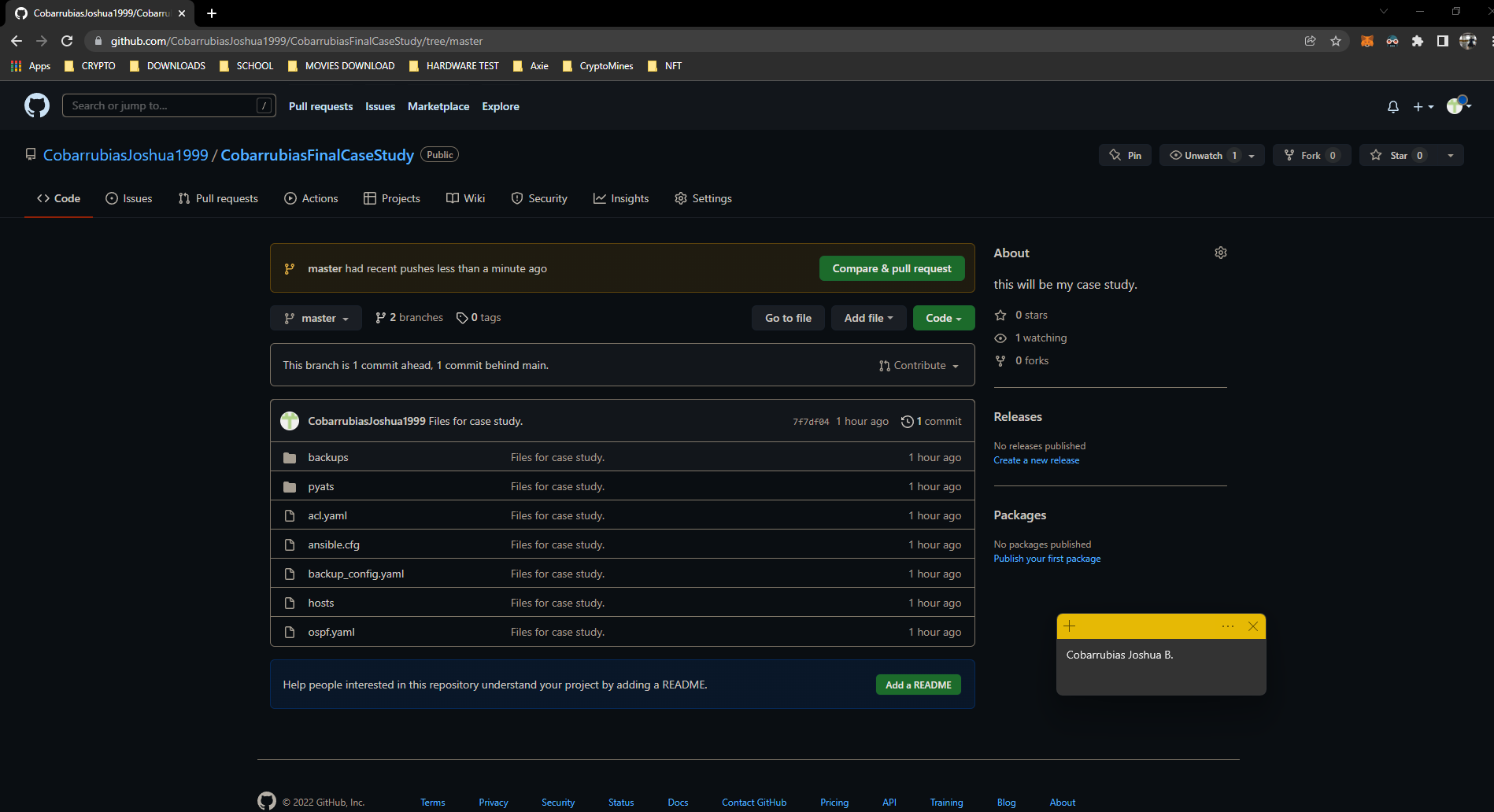
Now after coding the script.py we will now test the scripty using python that would run a basic test. The command will be pyats run job/pyats\_job.py

pyats run job pyats/pyats.py



So as you can see it was a success of testing the network using our job and script by pyATS.

Now we will now create a repository to our GitHub and commit and push our work for our case study.



1. **Conclusion:**
2. In conclusion this case study shows us how to utilize an ansible for configuration backups, ACL implementation, and other tasks. OSPF is used on the network’s router. The use of the fundamentals of ansible and understanding its significance is beneficial. Were automating the commands we use to configure our routers. Knowing how to put those into actions. Manually configuring network topics and using ansible will go a long way toward saving time when configuring a network. Finally, the laboratory exercise teaches us how to use pyATS and genie to test the network functionality. As well as whether the parameters are appropriately applied. And I also realized that all our activities or laboratory activities was helpful I based all my work in this case study on my past laboratories and implemented it here in my case study. I learned a lot in this class and will surely explore more so I can deeper my knowledge on network automation.

**Links:**

GitHub: <https://github.com/CobarrubiasJoshua1999/CobarrubiasFinalCaseStudy.git>

**Honor Pledge for Graded Assignment**

“I affirm that I have not given or received any unauthorized help on this assignment, and that this work is my own.”