## Lab 25: Implement REST APIs with an SDN Controller

### Case Study

FiberConnect Ltd**.**, a leading telecom, delivers high-speed broadband and fiber-optic connectivity to both residential and enterprise customers. With a workforce of over 1,200 professionals, the company manages a growing and complex network infrastructure that spans multiple cities. As FiberConnect expanded its services, the company faced increasing difficulty maintaining consistent performance and reliability using its legacy networking approach. The traditional methods, based on manual configurations and isolated device management, no longer met the demands of a scalable and responsive network. This prompted FiberConnect to explore modern network automation techniques to streamline operations, improve visibility, and minimize downtime.

### Business Challenge

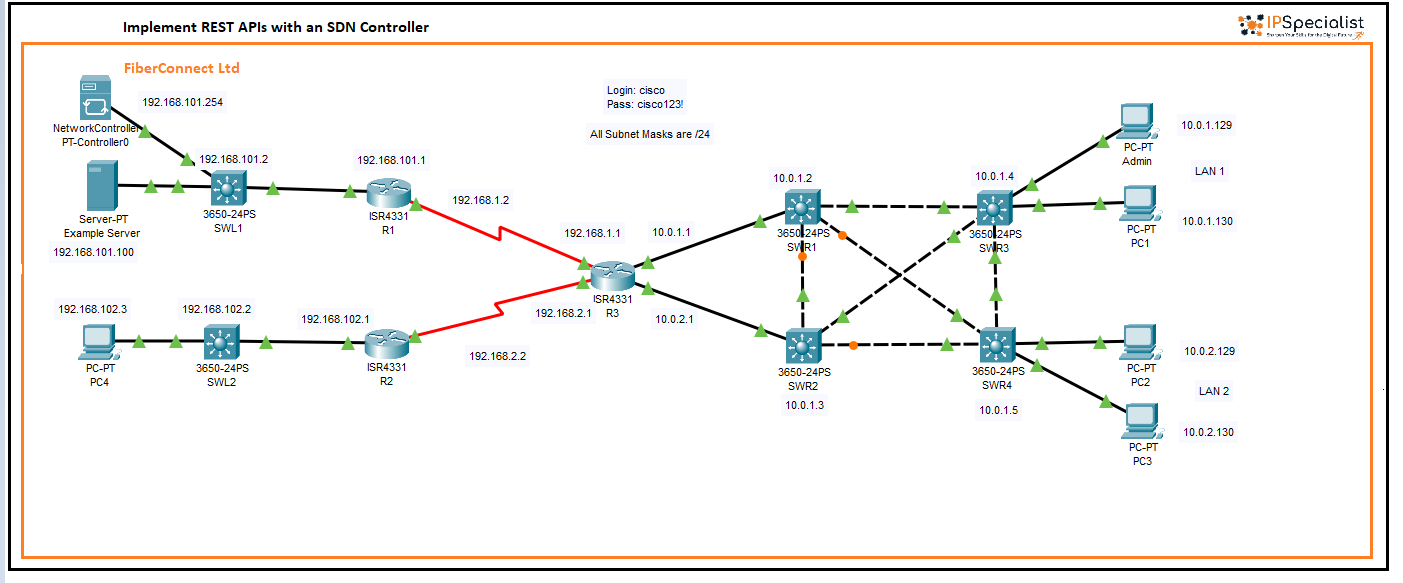
FiberConnect’s operations team encountered several key obstacles in their daily work. Network engineers were required to log into each router or switch individually to perform tasks, often relying on CLI-based configurations. This manual process was slow, error-prone, and lacked a unified way to enforce company-wide policies. Configuration backups and version tracking were inconsistently managed, making it difficult to recover from failures or perform security audits. In addition, there was no centralized monitoring system to provide real-time insights into the health of the network or detect anomalies. These limitations reduced operational efficiency, prolonged issue resolution, and hindered the company's ability to deliver consistent service to its customers.

### Solution

To modernize its network operations, FiberConnect implemented a centralized controller-based architecture using Cisco’s Software-Defined Networking (SDN) solution. By deploying a network controller within its environment, the company gained centralized access to all supported devices for tasks like discovery, configuration, and monitoring. The SDN controller enabled automated detection of network devices via Cisco Discovery Protocol (CDP), along with real-time visibility into device health, connectivity status, and network topology.

This approach eliminated the need for repetitive manual input and reduced the risk of misconfigurations. With automation in place, administrators could push DNS, NTP, and Syslog configurations across multiple devices simultaneously. The centralized dashboard also allowed quick access to logs, software versions, and operational data, significantly improving troubleshooting and network management. By embracing SDN-based automation, FiberConnect achieved greater scalability, faster issue resolution, and a more secure and standardized network environment.

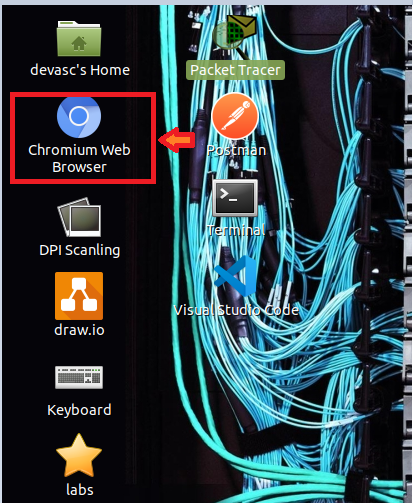
1. Verify External Connectivity to Packet Tracer
2. Request an Authentication Token with Postman
3. Send REST Requests with Postman
4. Send REST Requests with VS Code



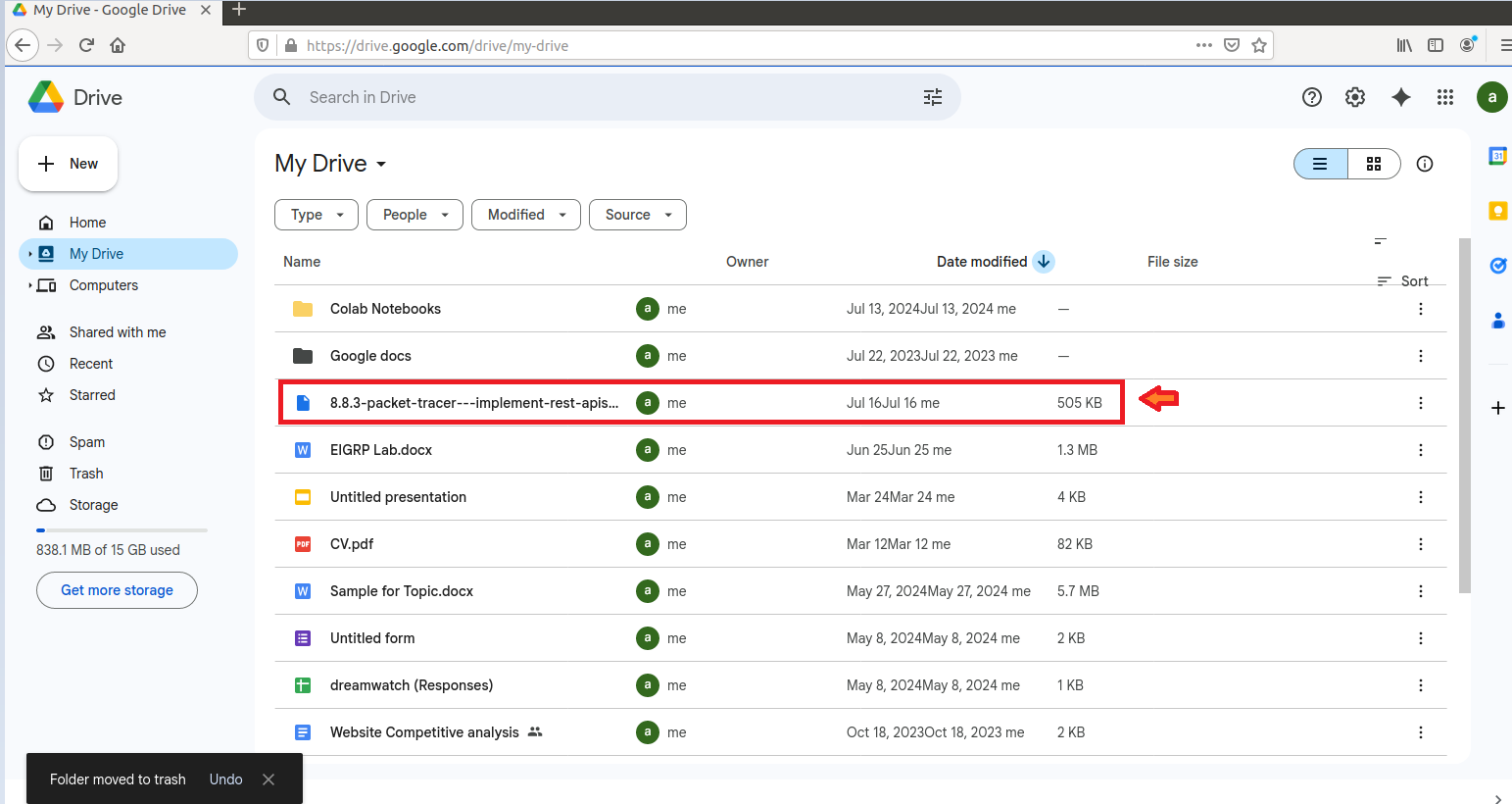
*Figure 25-01: Lab Topology*

**// Verify External Connectivity to Packet Tracer**

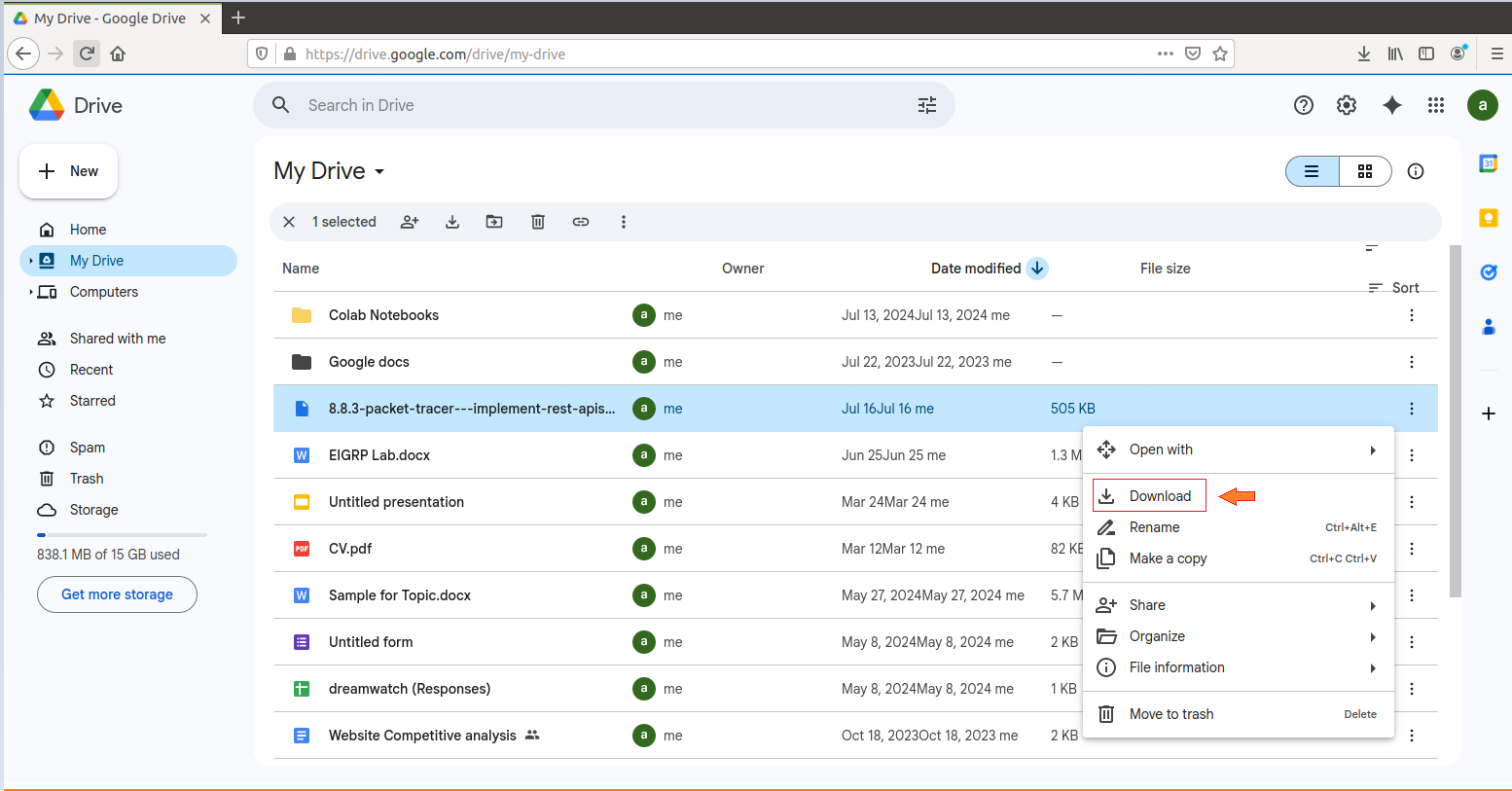
1. In this part of the activity, you will verify that Packet Tracer can be accessed by other applications within the DEVASC virtual machine (VM). To begin, ensure that the DEVASC VM is up and running. Within the VM environment, open the **Chromium browser** from the desktop or applications menu. Once the browser is open, navigate to your course curriculum, such as the Cisco NetAcad or Skills for All platform. This step confirms that the course content is accessible directly from within the VM, which is essential since the activity must be completed entirely inside the DEVASC VM. Support for accessing Packet Tracer from outside the VM is not provided. Within the DEVASC VM, access your course curriculum in the **Chromium browser**.



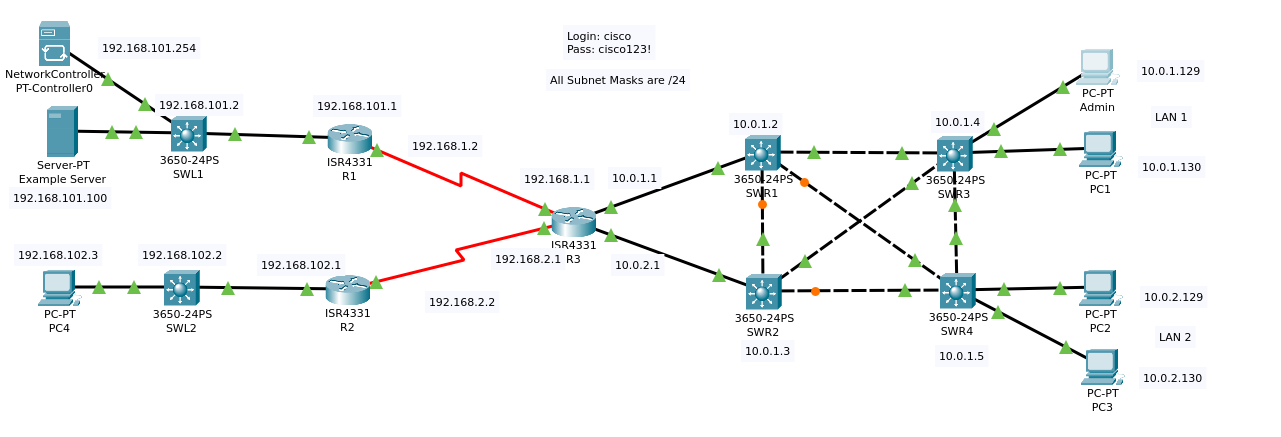
2. Navigate to the web page for this activity within the course curriculum inside the DEVASC virtual machine. Once you are on the activity page, locate the file titled "Packet Tracer – Implement REST APIs with an SDN Controller.pka" associated with the instructions.



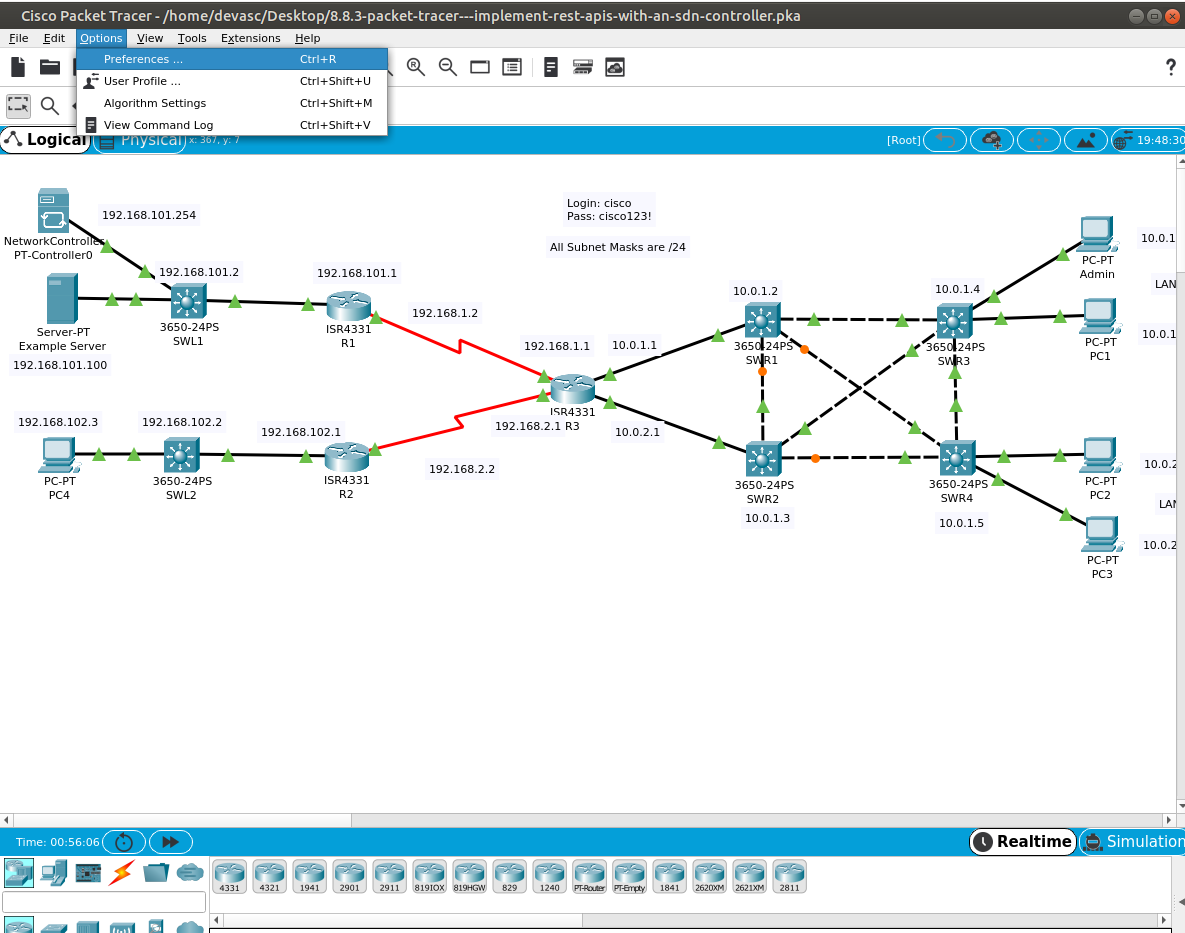
3. Click to **Download** the **.pka** file, and then open it directly in Packet Tracer once the download is complete.



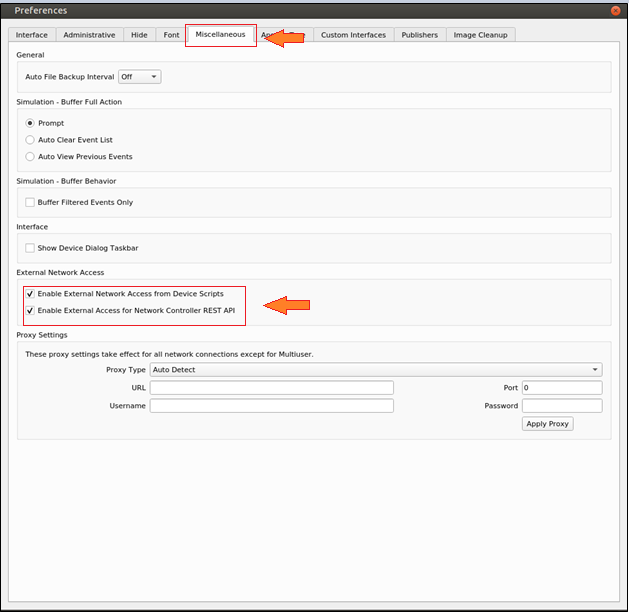
4. **Open** it directly in Packet Tracer. This will launch the activity and allow you to begin working on the practical lab within the DEVASC environment.



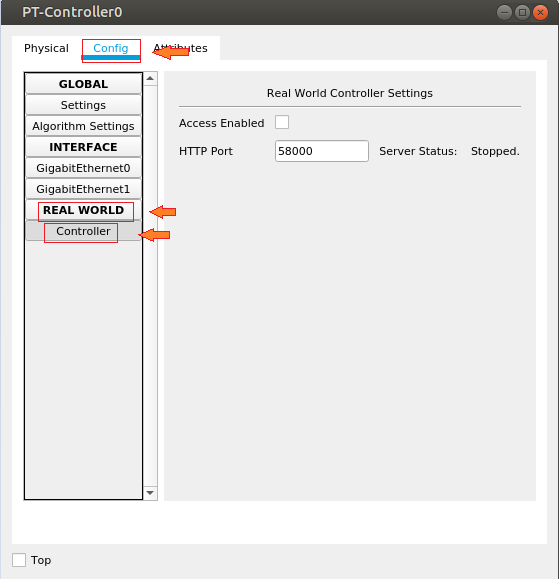
5. Click on **Options** then **Preferences.**



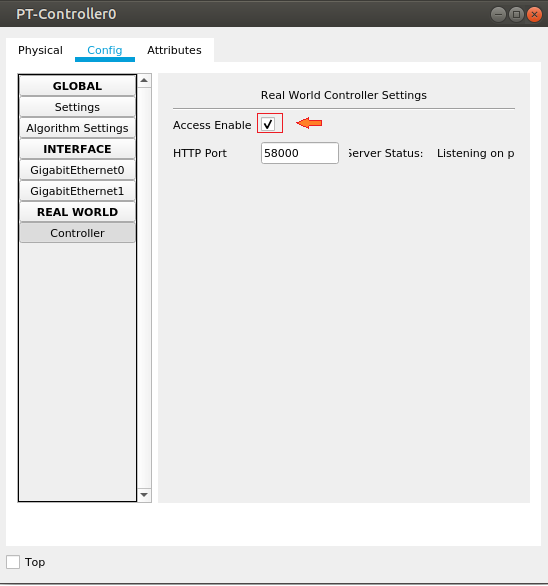
6. Go to **Miscellaneous**. Under External Network Access, make sure the option **Enable External Access for Network Controller REST API** is checked.



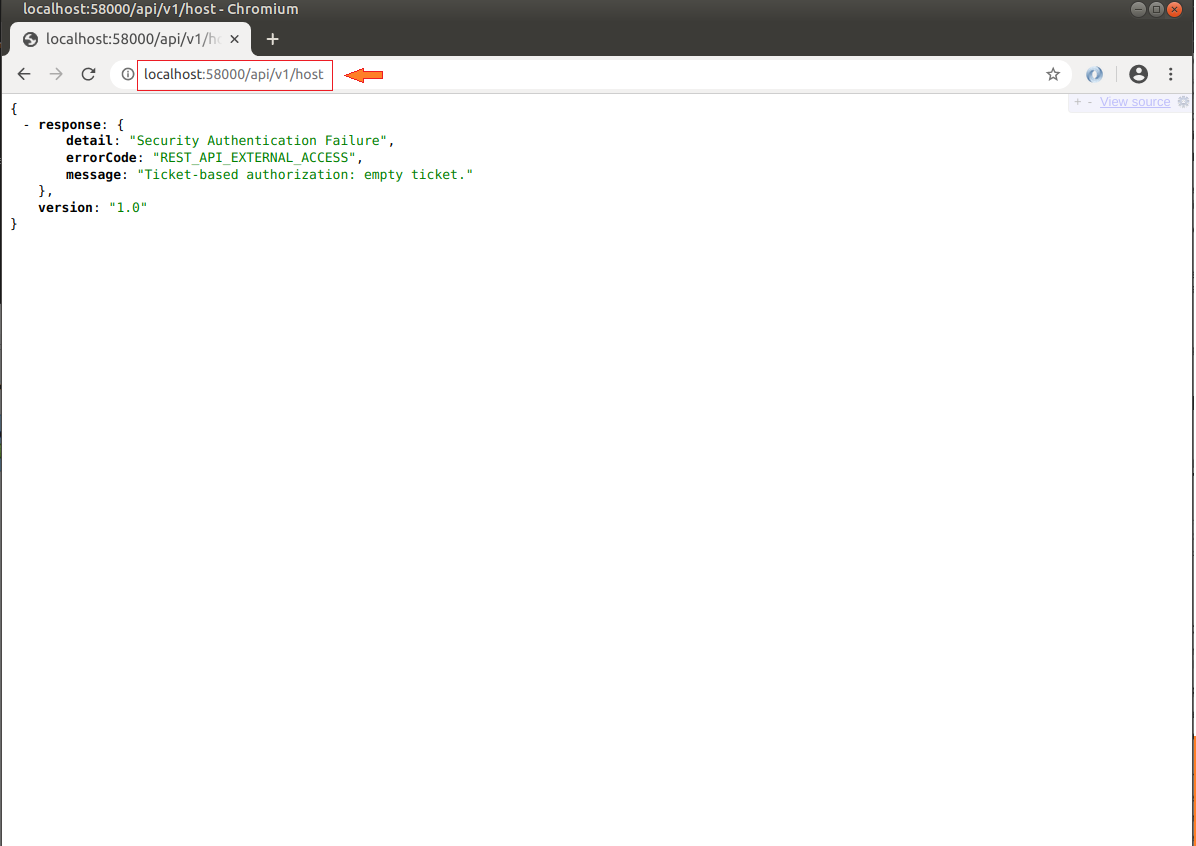
7. To begin, click on **PT-Controller0** and navigate to the **Config** tab. Locate the **"REAL WORLD"** section on the left-hand sideandclickon **Controller** to access its configuration settings.



8. Check **Access** **Enabled** and make a note of the port number, which is most likely **58000**. This is the port number you will need when externally accessing the Packet Tracer activity from Chromium, VS Code, and Postman later in this activity.



9. Open **Chromium** and go to the URL **http://localhost:58000/api/v1/host**. You should receive a response confirming the connection. This step verifies that external access to **Packet Tracer** and **PT-Controller0** is working correctly. Note that authorization requires a ticket, which you will obtain in the next part as an authorization token.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **// Request an Authentication Token with Postman**  1. Click **Admin > Desktop > Web Browser**.    2. Enter **192.168.101.254** and Log in to PT-Controller0 with **username cisco** and **password** **cisco123!.**  3. Click the **menu** next to the Cisco logo.    4. Choose **API Docs.**    5. The **API Docs** screen looks like this.    6. You can also access this same documentation from the **Help** menu. Click **Help > Contents.**    7.In the navigation pane on the left, scroll down approximately two-thirds of the way and click on **Network Controller API**. This section displays the same documentation that was available on PT-Controller0. In the API documentation, locate and click on **addTicket**, as you will use this endpoint in the next step to obtain an authorization token.    8.After reviewing the **addTicket** REST API method documentation, open **Postman** to begin testing the API request.    9. In the Launch area, click the **+ sign** to create a new Untitled Request. Click the down arrow and change the type from **GET** to **POST**.    10. Enter the URL **<http://localhost:58000/api/v1/ticket>.**    11. Below the URL field, click **Body**. Change the type to **raw**. Click the down arrow next to Text and change it to **JSON**. This change will also set the **Content-Type** HTTP Header to **application/json** that is required for this API call.    12. Paste the following JSON object into the Body field. Make sure your code is properly formatted   |  | | --- | | {  "username": "cisco",  "password": "cisco123!"  } |     13. Click the **Send** button to submit the **POST** request to **PT-Controller0**. Once the request is processed, you will receive a response. The content of this response may vary, but it will typically include a field labeled something like **serviceTicket** followed by an actual value unique to your session. This value represents your authentication token and will be used in subsequent API requests.   |  | | --- | | {  "response": {  "idleTimeout": 900,  "serviceTicket": "your\_serviceTicket",  "sessionTimeout": 3600  },  "version": "1.0"  } |     14. In Postman, start by clicking the **plus (+) icon** to open a new **Untitled Request** tab. In the URL field, enter **http://localhost:58000/api/v1/network-device**. Next, click on the **Headers** tab located just below the URL bar. Scroll to the last available row under the **KEY** column, click the empty field, and type **X-Auth-Token.** In the corresponding **Value** field, enter the actual service ticket you received from the authentication response. This token will authorize your request to access network device information.    15. Click the **Send** button to submit the **GET request** to the **PT-Controller0**. If your request is properly formatted and authorized, you should receive a response containing information about the network devices managed by the controller. The response will include details for all nine network devices in the network. Each device's data may include fields such as device ID, hostname, management IP address, type, and status.   |  | | --- | | {  "response": [  {  "collectionStatus": "Managed",  "connectedInterfaceName": [  "GigabitEthernet0/0/0",  "GigabitEthernet0",  "FastEthernet0"  ],  "connectedNetworkDeviceIpAddress": [  "192.168.101.1",  "192.168.101.254",  "192.168.101.100"  ],  "connectedNetworkDeviceName": [  "R1",  "NetworkController",  "Example Server"  ],  "errorDescription": "",  "globalCredentialId": "53046ecc-88c3-49f6-9626-ca8ab9db6725",  "hostname": "SWL1",  "id": "CAT1010BT47-uuid",  "interfaceCount": "29",  "inventoryStatusDetail": "Managed",  "lastUpdateTime": "1",  "lastUpdated": "2020-06-11 18:03:41",  "macAddress": "000C.CF42.2B11",  "managementIpAddress": "192.168.101.2",  "platformId": "3650",  "productId": "3650-24PS",  "reachabilityFailureReason": "",  "reachabilityStatus": "Reachable",  "serialNumber": "CAT1010BT47-",  "softwareVersion": "16.3.2",  "type": "MultiLayerSwitch",  "upTime": "3 minutes, 1 seconds"  },  {  "collectionStatus": "Managed",  "connectedInterfaceName": [  "GigabitEthernet1/0/1",  "Serial0/1/0"  ],  "connectedNetworkDeviceIpAddress": [  "192.168.101.2",  "192.168.1.1"  ],  "connectedNetworkDeviceName": [  "SWL1",  "R3"  ],  "errorDescription": "",  "globalCredentialId": "53046ecc-88c3-49f6-9626-ca8ab9db6725",  "hostname": "R1",  "id": "FDO1302XY2X-uuid",  "interfaceCount": "6",  "inventoryStatusDetail": "Managed",  "ipAddresses": [  "192.168.101.1",  "192.168.1.2"  ],  "lastUpdateTime": "1",  "lastUpdated": "2020-06-11 18:03:41",  "macAddress": "00D0.5852.527D",  "managementIpAddress": "192.168.1.2",  "platformId": "ISR4300",  "productId": "ISR4331",  "reachabilityFailureReason": "",  "reachabilityStatus": "Reachable",  "serialNumber": "FDO1302XY2X-",  "softwareVersion": "15.4",  "type": "Router",  "upTime": "3 minutes, 1 seconds"  },  {  "collectionStatus": "Managed",  "connectedInterfaceName": [  "GigabitEthernet1/0/1",  "GigabitEthernet1/0/1",  "Serial0/1/0",  "Serial0/1/1"  ],  "connectedNetworkDeviceIpAddress": [  "10.0.1.2",  "",  "192.168.1.2",  "192.168.2.2"  ],  "connectedNetworkDeviceName": [  "SWR1",  "SWR2",  "R1",  "R2"  ],  "errorDescription": "",  "globalCredentialId": "53046ecc-88c3-49f6-9626-ca8ab9db6725",  "hostname": "R3",  "id": "FDO13026087-uuid",  "interfaceCount": "6",  "inventoryStatusDetail": "Managed",  "ipAddresses": [  "10.0.1.1",  "10.0.2.1",  "192.168.1.1",  "192.168.2.1"  ],  "lastUpdateTime": "1",  "lastUpdated": "2020-06-11 18:03:41",  "macAddress": "00E0.B039.A39C",  "managementIpAddress": "192.168.2.1",  "platformId": "ISR4300",  "productId": "ISR4331",  "reachabilityFailureReason": "",  "reachabilityStatus": "Reachable",  "serialNumber": "FDO13026087-",  "softwareVersion": "15.4",  "type": "Router",  "upTime": "3 minutes, 1 seconds"  },  {  "collectionStatus": "Managed",  "connectedInterfaceName": [  "GigabitEthernet0/0/0",  "GigabitEthernet1/0/2",  "GigabitEthernet1/0/3",  "GigabitEthernet1/0/5"  ],  "connectedNetworkDeviceIpAddress": [  "10.0.1.1",  "10.0.1.4",  "10.0.1.5",  "10.0.1.3"  ],  "connectedNetworkDeviceName": [  "R3",  "SWR3",  "SWR4",  "SWR2"  ],  "errorDescription": "",  "globalCredentialId": "53046ecc-88c3-49f6-9626-ca8ab9db6725",  "hostname": "SWR1",  "id": "CAT101021Z6-uuid",  "interfaceCount": "29",  "inventoryStatusDetail": "Managed",  "lastUpdateTime": "1",  "lastUpdated": "2020-06-11 18:03:41",  "macAddress": "00E0.F915.E250",  "managementIpAddress": "10.0.1.2",  "platformId": "3650",  "productId": "3650-24PS",  "reachabilityFailureReason": "",  "reachabilityStatus": "Reachable",  "serialNumber": "CAT101021Z6-",  "softwareVersion": "16.3.2",  "type": "MultiLayerSwitch",  "upTime": "3 minutes, 1 seconds"  },  {  "collectionStatus": "Managed",  "connectedInterfaceName": [  "GigabitEthernet0/0/1",  "GigabitEthernet1/0/2",  "GigabitEthernet1/0/4",  "GigabitEthernet1/0/5"  ],  "connectedNetworkDeviceIpAddress": [  "10.0.2.1",  "10.0.1.5",  "10.0.1.4",  "10.0.1.2"  ],  "connectedNetworkDeviceName": [  "R3",  "SWR4",  "SWR3",  "SWR1"  ],  "errorDescription": "",  "globalCredentialId": "53046ecc-88c3-49f6-9626-ca8ab9db6725",  "hostname": "SWR2",  "id": "CAT1010JJ1H-uuid",  "interfaceCount": "29",  "inventoryStatusDetail": "Managed",  "lastUpdateTime": "1",  "lastUpdated": "2020-06-11 18:03:41",  "macAddress": "00E0.B060.5317",  "managementIpAddress": "10.0.1.3",  "platformId": "3650",  "productId": "3650-24PS",  "reachabilityFailureReason": "",  "reachabilityStatus": "Reachable",  "serialNumber": "CAT1010JJ1H-",  "softwareVersion": "16.3.2",  "type": "MultiLayerSwitch",  "upTime": "3 minutes, 1 seconds"  },  {  "collectionStatus": "Managed",  "connectedInterfaceName": [  "GigabitEthernet1/0/1",  "Serial0/1/1"  ],  "connectedNetworkDeviceIpAddress": [  "192.168.102.2",  "192.168.2.1"  ],  "connectedNetworkDeviceName": [  "SWL2",  "R3"  ],  "errorDescription": "",  "globalCredentialId": "53046ecc-88c3-49f6-9626-ca8ab9db6725",  "hostname": "R2",  "id": "FDO13022UJ0-uuid",  "interfaceCount": "6",  "inventoryStatusDetail": "Managed",  "ipAddresses": [  "192.168.102.1",  "192.168.2.2"  ],  "lastUpdateTime": "0",  "lastUpdated": "2020-06-11 18:03:42",  "macAddress": "0060.4797.3DA5",  "managementIpAddress": "192.168.2.2",  "platformId": "ISR4300",  "productId": "ISR4331",  "reachabilityFailureReason": "",  "reachabilityStatus": "Reachable",  "serialNumber": "FDO13022UJ0-",  "softwareVersion": "15.4",  "type": "Router",  "upTime": "3 minutes, 2 seconds"  },  {  "collectionStatus": "Managed",  "connectedInterfaceName": [  "GigabitEthernet0/0/0",  "FastEthernet0"  ],  "connectedNetworkDeviceIpAddress": [  "192.168.102.1",  "192.168.102.3"  ],  "connectedNetworkDeviceName": [  "R2",  "PC4"  ],  "errorDescription": "",  "globalCredentialId": "53046ecc-88c3-49f6-9626-ca8ab9db6725",  "hostname": "SWL2",  "id": "CAT101059L6-uuid",  "interfaceCount": "29",  "inventoryStatusDetail": "Managed",  "lastUpdateTime": "15",  "lastUpdated": "2020-06-11 18:03:27",  "macAddress": "0090.2155.BB91",  "managementIpAddress": "192.168.102.2",  "platformId": "3650",  "productId": "3650-24PS",  "reachabilityFailureReason": "",  "reachabilityStatus": "Reachable",  "serialNumber": "CAT101059L6-",  "softwareVersion": "16.3.2",  "type": "MultiLayerSwitch",  "upTime": "2 minutes, 47 seconds"  },  {  "collectionStatus": "Managed",  "connectedInterfaceName": [  "GigabitEthernet1/0/2",  "GigabitEthernet1/0/3",  "GigabitEthernet1/0/5",  "FastEthernet0",  "FastEthernet0"  ],  "connectedNetworkDeviceIpAddress": [  "10.0.1.3",  "10.0.1.2",  "10.0.1.4",  "10.0.2.129",  "10.0.2.130"  ],  "connectedNetworkDeviceName": [  "SWR2",  "SWR1",  "SWR3",  "PC2",  "PC3"  ],  "errorDescription": "",  "globalCredentialId": "53046ecc-88c3-49f6-9626-ca8ab9db6725",  "hostname": "SWR4",  "id": "CAT1010K0UR-uuid",  "interfaceCount": "29",  "inventoryStatusDetail": "Managed",  "lastUpdateTime": "15",  "lastUpdated": "2020-06-11 18:03:27",  "macAddress": "0060.5C0D.E4AE",  "managementIpAddress": "10.0.1.5",  "platformId": "3650",  "productId": "3650-24PS",  "reachabilityFailureReason": "",  "reachabilityStatus": "Reachable",  "serialNumber": "CAT1010K0UR-",  "softwareVersion": "16.3.2",  "type": "MultiLayerSwitch",  "upTime": "2 minutes, 47 seconds"  },  {  "collectionStatus": "Managed",  "connectedInterfaceName": [  "GigabitEthernet1/0/2",  "GigabitEthernet1/0/4",  "GigabitEthernet1/0/5",  "FastEthernet0",  "FastEthernet0"  ],  "connectedNetworkDeviceIpAddress": [  "10.0.1.2",  "10.0.1.3",  "10.0.1.5",  "10.0.1.130",  "10.0.1.129"  ],  "connectedNetworkDeviceName": [  "SWR1",  "SWR2",  "SWR4",  "Admin",  "PC1"  ],  "errorDescription": "",  "globalCredentialId": "53046ecc-88c3-49f6-9626-ca8ab9db6725",  "hostname": "SWR3",  "id": "CAT1010J4FO-uuid",  "interfaceCount": "29",  "inventoryStatusDetail": "Managed",  "lastUpdateTime": "15",  "lastUpdated": "2020-06-11 18:03:27",  "macAddress": "0050.0F7C.0C09",  "managementIpAddress": "10.0.1.4",  "platformId": "3650",  "productId": "3650-24PS",  "reachabilityFailureReason": "",  "reachabilityStatus": "Reachable",  "serialNumber": "CAT1010J4FO-",  "softwareVersion": "16.3.2",  "type": "MultiLayerSwitch",  "upTime": "2 minutes, 47 seconds"  }  ],  "version": "1.0"  } |     16. In Postman, **right-click** the tab for your host GET request and choose **Duplicate** Tab.    17. All information in the ticket is the same except for the URL. Simply change network-device to host: <http://localhost:58000/api/v1/host>.   |  | | --- | | {  "response": [  {  "connectedAPMacAddress": "",  "connectedAPName": "",  "connectedInterfaceName": "GigabitEthernet1/0/24",  "connectedNetworkDeviceIpAddress": "192.168.102.2",  "connectedNetworkDeviceName": "SWL2",  "hostIp": "192.168.102.3",  "hostMac": "00E0.F96C.155B",  "hostName": "PC4",  "hostType": "Pc",  "id": "PTT08108MO8-uuid",  "lastUpdated": "2020-06-11 22:49:32",  "pingStatus": "SUCCESS"  }  // Additional devices omitted for brevity  ],  "version": "1.0"  } | |

|  |  |  |
| --- | --- | --- |
| **// Send REST Requests with VS Code**  1. In vs code, click on **File** and then click on **Open Folder.**    2. Navigate to the **devnet-src/ptna** directory.    3. Click on **OK.**    4. In the **Explore** pane on the left, you will see three Python scripts listed:   * 01\_get-ticket.py * 02\_get-network-device.py * 03\_get-host.py   Review the code in each script. You will notice that both **02\_get-network-device.py** and **03\_get-host.py** depend on a valid service ticket to authenticate API requests. These scripts contain a placeholder, **your\_serviceTicket**, which needs to be replaced with the actual ticket value returned by the controller.  To obtain a valid service ticket, run the **01\_get-ticket.py script**. This script sends a Post request to the **/api/v1/ticket** endpoint using your login credentials. Once executed, it returns a service ticket that you can copy and use to replace the placeholder in the other two scripts. Running this first script helps demonstrate how authentication works within the network controller API workflow.    5. Open **Terminal** and then select **New Terminal**.    6. Run the **01\_get-ticket.py** to see output similar to the following.    7. Replace **your\_serviceTicket** value in **02\_get-network-device.py** with the value Packet Tracer gave you.    8. Earlier, when the network device API was called through Postman, it returned a comprehensive list of all nine network devices, including every available detail for each device. In contrast, the **02\_get-network-device.py** script is designed to display only specific pieces of information that are relevant to the programmer’s needs, namely, the hostname, platformId, and managementIpAddress. To view this filtered output, open your terminal and run the script by executing **python3 02\_get-network-device.py**. This will produce a cleaner and more focused summary of the network devices, highlighting only the essential details.   |  | | --- | | import json  import requests  api\_url = "http://localhost:58000/api/v1/network-device"  headers={"X-Auth-Token": "NC-148-691bc022166c46a08d79-nbi"}  resp = requests.get(api\_url, headers=headers, verify=False)  print("Request status: ", resp.status\_code)  response\_json = resp.json()  networkDevices = response\_json["response"]  for networkDevice in networkDevices:  print(networkDevice["hostname"], "\t", networkDevice["platformId"], "\t", networkDevice["managementIpAddress"]) |     9. Similarly, the programmer designed the **03\_get-host.py** script to display only selected information for each of the six host devices connected to the network. Change **your \_service\_ ticket** with the ticket number given.    10. Replace **your\_serviceTicket** value in **03\_get-host. py** with the value Packet Tracer gave you. Rather than printing all the available data returned by the API, the script focuses on specific key values that are most relevant, such as hostname, IP address, and MAC address.   |  | | --- | | import json  import requests  api\_url = "http://localhost:58000/api/v1/host"  headers={"X-Auth-Token": "NC-148-691bc022166c46a08d79-nbi"}  resp = requests.get(api\_url, headers=headers, verify=False)  print("Request status: ", resp.status\_code)  response\_json = resp.json()  hosts = response\_json["response"]  for host in hosts:  print(host["hostName"], "\t", host["hostIp"], "\t", host["hostMac"], "\t", host["connectedInterfaceName"]) | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **// Send REST Requests Inside Packet Tracer**  1.In Packet Tracer, click the **Admin** **PC** and then click the **Programming** tab.    2.To begin, ensure that no project is currently open in the IDE. If prompted with a message stating, **There** **is currently no project**, click on the **New** option. In the project creation dialog, enter **REST APIs** as the project name and select **Empty - Python** as the template. Click **Create** to generate your new Python project with a blank **main.py** script.   3. To modify the scripts to run inside Packet Tracer. In VS Code, copy the code for 03\_get-host.py. In the Admin > Programming tab, double-click the main.py script to open it. Paste the code in the main.py script.   4.Change the **api\_url.** Replace **localhost:58000/api/v1/host** with **192.168.101.254/api/v1/host.**   |  | | --- | | import json  import requests  api\_url = "http://localhost:58000/api/v1/host"  headers={"X-Auth-Token": "NC-148-27c19db7847e4b188cf1-nbi"}  resp = requests.get(api\_url, headers=headers, verify=False)  print("Request status: ", resp.status\_code)  response\_json = resp.json()  hosts = response\_json["response"]  for host in hosts:  print(host["hostName"], "\t", host["hostIp"], "\t", host["hostMac"], "\t", host["connectedInterfaceName"]) |     5.Edits are automatically saved. Nowclick **Run.** Packet Tracer output does not exactly simulate what you see in the **Linux command line**. However, you should see similar output as shown below.   |  | | --- | | import json  import requests  api\_url = "http://192.168.101,254/api/v1/host"  headers={"X-Auth-Token": "NC-148-27c19db7847e4b188cf1-nbi"}  resp = requests.get(api\_url, headers=headers, verify=False)  print("Request status: ", resp.status\_code)  response\_json = resp.json()  hosts = response\_json["response"]  for host in hosts:  print(host["hostName"], "\t", host["hostIp"], "\t", host["hostMac"], "\t", host["connectedInterfaceName"]) |     6.Copy and paste **02\_get-network-device.py** into the **main.py.** Change the **URL** and run it.   |  | | --- | | import json  import requests  api\_url = "http://localhost:58000/api/v1/network-device"  headers={"X-Auth-Token": "NC-148-27c19db7847e4b188cf1-nbi"}  resp = requests.get(api\_url, headers=headers, verify=False)  print("Request status: ", resp.status\_code)  response\_json = resp.json()  networkDevices = response\_json["response"] |     7.Edits are automatically saved**.** Click **Run.** Packet Tracer output does not exactly simulate what you see in the **Linux command line**. However, you should see similar output as shown below.   |  | | --- | | **Change to the following**  import json  import requests  api\_url = "http://192.168.101.254/api/v1/network-device"  headers={"X-Auth-Token": "NC-148-27c19db7847e4b188cf1-nbi"}  resp = requests.get(api\_url, headers=headers, verify=False)  print("Request status: ", resp.status\_code)  response\_json = resp.json()  networkDevices = response\_json["response"]  for networkDevice in networkDevices:  print(networkDevice["hostname"], "\t", networkDevice["platformId"], "\t", networkDevice["managementIpAddress"])  for networkDevice in networkDevices:  print(networkDevice["hostname"], "\t", networkDevice["platformId"], "\t", networkDevice["managementIpAddress"]) | |